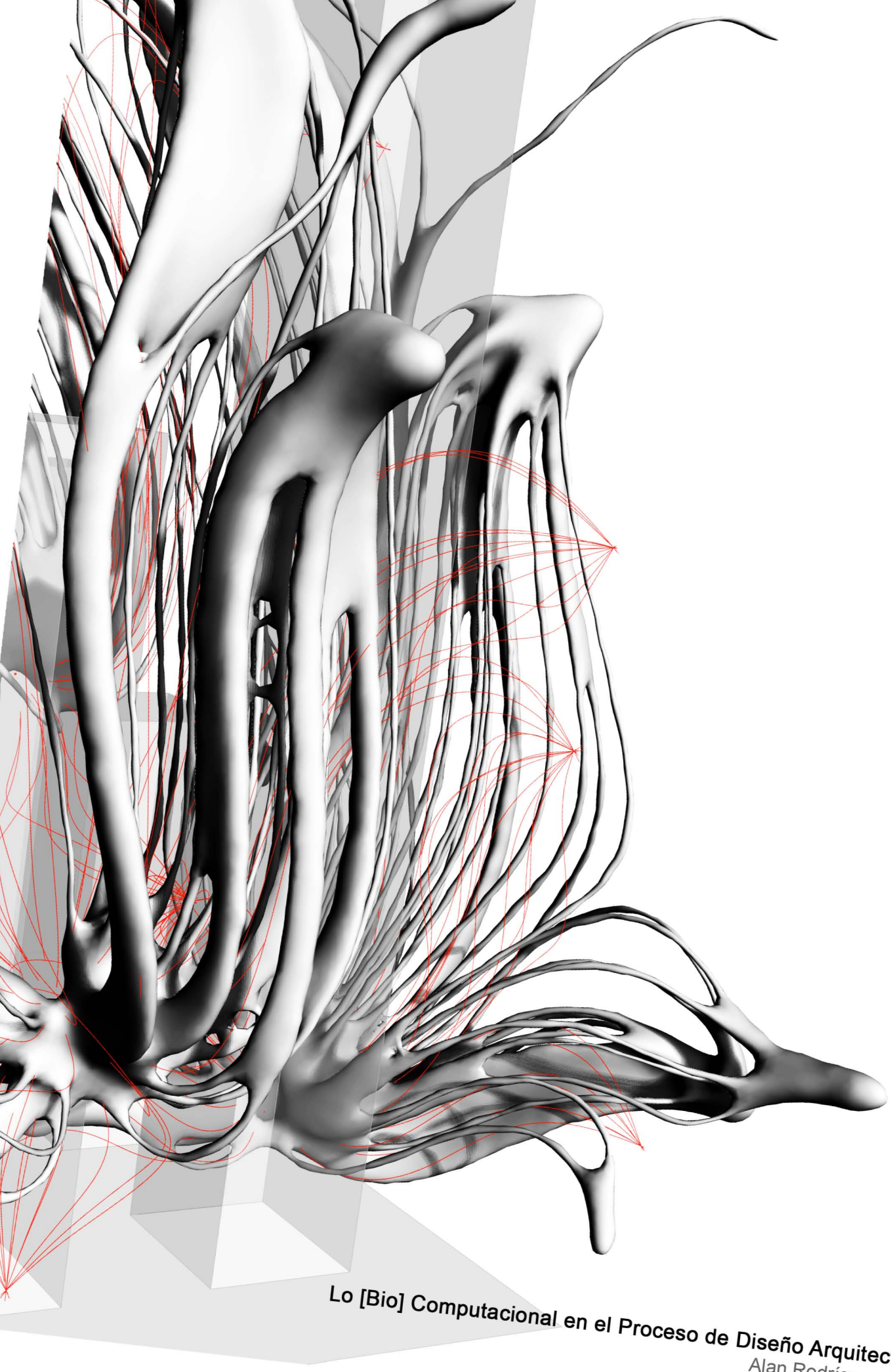


Lo [Bio] Computacional en el Proceso de Diseño Arquitectónico

Alan Rodríguez Carrillo

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Universidad Nacional Autónoma de México
Facultad de Arquitectura

Tema:

Lo [Bio]Computacional en el Proceso de Diseño Arquitectónico

Tesis para obtener el título de: **Arquitecto**

Presenta:

Alan Rodríguez Carrillo

Sinodales:

Doctor en Arquitectura: Ronan Bolaños Linares
Maestro en Arquitectura: Luis Eduardo de la Torre Zatarain
Maestra en Urbanismo: Claudia Gabriela Ortiz Chao

Fecha: 17 de noviembre de 2016

Índice

Introducción

Fundamentación

Capítulo 1 - La ingeniería computacional en los procesos de diseño 04

1.1 – Ivan Sutherland y el Software Sketchpad 04
1.2 – Problemas con la flexibilidad 06
1.3 – La crisis del software y la curva Böhm 09
1.4 – El parametricismo es paramétrico 12
1.5 – La ingeniería del software: cuerpo de conocimiento 22
1.6 – Conclusiones 42

Capítulo 2 - Programación Lógica Gráfica 44

2.1 – Los paradigmas en la programación 44
2.2 – Programación lógica gráfica – desambiguación 47
2.3 – El diagrama algorítmico gráfico como proceso en el diseño 50
2.4 – Conclusiones 53

Capítulo 3 - Programación Lógica Textual 55

3.1 – Ver, programar, diseñar 55
3.2 – Introducción a *Python* 65
3.3 – La programación textual en el diseño 76
3.4 – Conclusiones 82

Capítulo 4 - Caso de estudio Lógico – Gráfico 84

4.1 – La crisis de la forma 84
4.2 – El camino de la naturaleza 89
4.3 – La evolución material aplicada al diseño 120
4.4 – La herramienta digital computacional en la estructuración material 128
4.5 – Estrategias de la naturaleza computarizadas 138
4.6 – Problemas de investigación: el diseño como segunda naturaleza 140
4.7 – Proceso de diseño: fundamentos biológicos – matemáticos 146
4.8 – Enfoques aplicados al proceso de diseño 157
4.9 – Conclusiones 215

Capítulo 5 - Caso de estudio Lógico – Textual 220

5.1 – Trans 221
5.2 – Espacio desconocido: La estructura espacial de las ilusiones 222
5.3 – Corpóreo, Real e Híper 226
5.4 – Ciudad [In]Visible 230
5.5 – Flujo 238
5.6 – Enfoques aplicados al proceso de diseño 239
5.7 – Conclusiones 284

Conclusión General 287

Bibliografía

Trabajos citados 291
Índice de ilustraciones 300

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Introducción

El objetivo de esta investigación es analizar, comprender y explorar los procesos bio - computacionales, generalizando lo que la palabra significa, en los procesos del diseño arquitectónico.

Con el fin de direccionar este objetivo, este documento se enfoca en lo que Robert Woodbury (2010, 66) ha identificado como la “razón común” entre lo programable y el diseño en el ámbito del modelado logarítmico y paramétrico, un espacio donde actualmente las “pequeñas conexiones explícitas” existen en esta rama del conocimiento. En esta tesis se considera cómo las prácticas desde la ingeniería del software pueden conectarse con la práctica y procesos en el diseño arquitectónico, el cual he estado aplicando durante mi corta experiencia en la producción de modelos arquitectónicos y de simulación de la naturaleza. En los primeros capítulos de este documento, se generaliza y analizan conceptos clave para el entendimiento y comprensión de los temas a tratar. He seleccionado dos proyectos en los capítulos 4 y 5 con el propósito de poner en práctica los conceptos antes expuestos. Los capítulos que finalizan este documento exponen las conclusiones a partir del análisis de dos casos de estudio, con el objetivo de considerar cómo el proceso de diseño en arquitectura puede influir e informar al arquitecto o diseñador para crear modelos paramétricos y asociativos que sean considerados como consecuencias de un proceso individual de diseño y establezcan las relaciones que existen en el ámbito arquitectónico y computacional específico.

Un límite que tengo en el presente objetivo es el de explorar solo dos casos de estudio y su problemática particular, ya que solo se tratarán dos tipos de modelado digital. No todos los arquitectos han empleado o han sido conscientes de las técnicas computacionales empleadas que influyen de manera directa sobre su proceso de diseño particular o colectivo, el modelado paramétrico involucra solo una rama de conocimientos teóricos y técnicos referidos a lo computacional entre muchas otras. La flexibilidad de elección y adaptación computacional en los procesos de diseño en la disciplina arquitectónica pueden también ser alcanzados a través de otros métodos disponibles como el *Building Information Modelling* (BIM). Los arraigados defensores de BIM contienden y fundamentan la capacidad de este tipo de software en reducir la reedición de trabajo (lo cual crea flexibilidad), por la creación de repositorios centralizados de datos que todas las representaciones del proyecto suponen: cambiar los datos solo una vez y teóricamente todo se actualizará (Eastman. 2012, 15 – 26).

Esta investigación se enfocará sobre dos cuestiones computacionales relacionadas a un proceso de diseño arquitectónico particular, la programación lógica visual y la programación lógica textual, debido a la experiencia con estas dos ramas de conocimiento computacional y la oportunidad de presentar un caso de estudio por cada una. Sin embargo, me parece pertinente no solo focalizar este trabajo de investigación a mis trabajos de diseño y no ser antagonista a otros procesos. Desde que los arquitectos frecuentemente integran procesos de modelado digital en conjunto, las mejoras a la práctica de estos procesos de representación y generación formal digital se podrían manifestar por sí solos implementando características paramétricas, asociativas, algorítmicas, entre otras, así como otras representaciones flexibles como BIM.

La metodología a aplicar para el desarrollo de los temas será de forma descriptiva, analítica y demostrativa, preponderando el entendimiento y cuestionamiento de posturas especializadas en las disciplinas correspondientes.

Los criterios de selección de materiales para la investigación se realizarán de acuerdo con la relevancia, profundización de los temas a tratar, nivel de especialización y critica recibida por los medios y los profesionales involucrados en los temas.

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Nota sobre el lenguaje y términos

Dentro del objetivo que definirá la dirección de la tesis, y a través de este documento, frecuentemente emplearé el uso del término de *"Modelo paramétrico", "Ingeniería del software" y "Lo computacional".* Los tres términos son conflictivos. Para ayudar a clarificar mis intenciones explicaré en esta sección por qué he incluido estos términos.

Los arquitectos asignan un rango de significados a la frase de *"Paramétrico"* y *"Modelo Paramétrico"*. Un gran número de definiciones han sido anticipadas y asignan un valor meramente basadas en clasificaciones de estilo. Neil Leach ha hecho una aclaración respecto a este desacuerdo, el cual se encuentra tan intensamente arraigado y mal definido, declarando lo siguiente: "*Pocas personas en el occidente que trabajan a la vanguardia de la computación enfocada al diseño arquitectónico usan el término paramétrico"* (Leach y Schumacher 2012). Sin embargo, he elegido usar el término en este documento porque el término *paramétrico* tiene un significado histórico muy preciso. El número de definiciones contemporáneas son una ilustración de como la concepción moderna del término *modelado paramétrico* ha sido cambiado. Exploro y explico esos cambios en los siguientes sub-capítulos. En las siguientes secciones explico las definiciones y las utilizaré en esta tesis: un modelo paramétrico es una lista de ecuaciones que expresan un modelo geométrico como funciones y sistemas explícitos de un número de variables numéricas, ambientales, entre otras.

La frase "Ingeniería del software" ha sido elegida deliberadamente para ser provocativa, escrita y definida por los autores que han inventado el término en un encuentro del comité de Ciencia de la OTAN en 1968 (Naur y Randell 1968, 13). La intención original fue fundamentar la práctica del proceso de manufactura del software en un fundamento teórico similar a otras disciplinas de la ingeniería (Naur y Randell 1968, 13). Muchas personas involucradas en el tema han argumentado desde el inicio que la ingeniería es una disciplina inapropiada para basar la identidad de la programación sobre esta. Esto ha llevado a Tom DeMarco (2005, 95) declarar "la ingeniería del software es una idea, cuyo tiempo ha llegado y se ha ido". Otros han comentado que la manufactura del software es más que una disciplina de diseño (Brooks 2010), o un oficio (Seibel 2009), o un arte (Knuth 1968). Esto queda fuera del ámbito de mi investigación para resolver este debate de más de 40 años. En esta tesis utilizo la frase *"Ingeniería del software"* no porque es una analogía adecuada para lo que los programadores hacen, sino más bien porque la ingeniería de software es un término aun ampliamente utilizado para designar el conjunto de conocimientos sobre la producción de software.

Por último, he escogido el uso del término "*Lo computacional*" por el significado neutro que trae la interpretación personal directa, con el fin de describir aquellos procesos que involucran cualquier medio de representación, modelado y visualización en el diseño arquitectónico a través de una computadora. Es un campo interdisciplinario entre el modelado basado en reglas y el modelado estadístico del lenguaje natural desde una perspectiva digital. Dicho modelado no se centra en ninguna de las áreas de la lingüística en particular, sino que es un campo interdisciplinario, en el que participan lingüistas, informáticos y programadores especializados en inteligencia artificial, psicólogos cognoscitivos y expertos en lógica, biólogos, usuarios, arquitectos, entre otros.

Fundamentación

Los diferentes procesos de diseño en al ámbito de la arquitectura hacen referencia a un contexto histórico, político, social y económico. La arquitectura responde a las demandas de su época, además de proporcionar un significado acorde a cada contexto humano.

El constante proceso de globalización y la información masiva en los medios, han influido de manera significativa en el hacer arquitectónico, expresado por medio de diferentes posturas por parte de autores, las cuales, a lo largo de la historia contemporánea, han retomado análisis y reinterpretaciones de épocas, a través de los significados particulares y colectivos.

En el estricto sentido de la palabra, un proceso de diseño puede ser válido o no, ya que puede operar o no en una comunidad. El ser humano en la actualidad está inmerso en el mundo de la comunicación y la información masiva, es importante hacer una reflexión sobre este fenómeno en lo que respecta a la arquitectura y en específico a los procesos del diseño que involucran los medios de producción abstractos y para problemas particulares de población, densidad, climáticos, habitabilidad, entre otros.

Un proceso de diseño arquitectónico, es una reinterpretación de los agentes externos que influyen de manera física y psicológica en el ser humano, reflejados a través de la arquitectura. Dichos agentes que toman un papel relevante en los procesos del diseño arquitectónico, no se pueden excluir, ya que son los generadores primarios de cualquier teoría, además de ser los fundamentos conceptuales en los cuáles se basa un determinado objeto particular.

Lo computacional ha sido relevante en al ámbito de la arquitectura, no solo como medio de representación y expresión gráfica, sino como una herramienta que involucra conexiones entre el humano y la computadora, a través del cerebro, el movimiento, el cuerpo y el control de las manos en los procesos del diseño arquitectónico. En la actualidad, hacer caso omiso de este fenómeno es dar la espalda al mundo actual, en sociedades humanas cada vez más mecanizadas y computarizadas, como consecuencia de los grandes sistemas de producción en el ámbito tecnológico. El proceso de diseño arquitectónico tiene que responder a estos fenómenos, no solo a través del uso de computadoras para una tarea en específico; de manera específica, los programas se han convertido en herramientas imprescindibles del ser humano que actúan por órdenes directas de los movimientos motrices instruidos por las conexiones neuronales del cerebro. Es un acontecimiento que diariamente se ve, pero no se observa, la interacción de un humano con un dispositivo electrónico.

Los avances matemáticos, físicos, computacionales, biológicos, entre otros, no pueden de ser ajenos a la arquitectura y a sus procesos de producción formal, teórica y técnica - constructiva, la respuesta consiente del cuerpo a través del movimiento de las manos es y ha sido el medio primario de generación, expresión y producción en un proceso de diseño particular, y es en la actualidad, un agente que ha sido modificado en su función por medio de empleo de robots y equipos mecánicos que simplifican las tareas de la construcción en diferentes escalas, además de disminuir los costos económicos de personal y tiempos de producción en un proyecto específico, sea este de una pequeña o gran escala.

El hombre, el entorno natural y la máquina como tres entidades, se encuentran en constante relación en la actualidad, es lo que acontece en el presente de sociedades globales y sensibles a la información, acontecimientos que deben de ser tomados en cuenta por su influencia indirecta en los procesos de diseño particulares en la práctica arquitectónica.

Capítulo 1 - La ingeniería computacional en los procesos de diseño

1.1 – Ivan Sutherland y el software Sketchpad

Figura 1. En una fotografía del Laboratorio Lincoln (1964) Timothy Johnson dibuja una línea con la pluma de tinta de luz.

Timothy Johnson (figura 1) arrastra un lápiz a través del parpadeo de la pantalla y, por un breve tiempo, se ve como si estuviera dibujando. Para la mayoría de personas en 1964, esta demostración de Ivan Sutherland con el software Sketchpad será la primera vez que han visto a alguien dibujar e interactuar con una computadora al mismo tiempo.

Las acciones de Johnson inicialmente se asemejan a un diseñador en un tablero de dibujo, la tinta es reemplazada con la luz, la impresión de la computadora haciendo similitud a la mesa de dibujo solamente cuando Johnson crea un trapecio deforme y roto (figura 2) y desafiante afirma "no debería ser requerido un objeto completo para dibujar la forma exacta para iniciar" (Laboratorio Lincoln 1964, 09:00). A continuación, oscurece la pantalla para llegar a un botón a su izquierda (figura 3). Cuando Johnson vuelve a sentarse él revela, con un toque de magia, que el trapecio se ha transformado en un rectángulo (figura 4). El juego de manos que subyace a este mecanismo digital es cuando Johnson parece estar dibujando algo que es en realidad la comunicación de un conjunto de relaciones geométricas en el equipo. Estas relaciones forman un tipo específico de relaciones en el programa de computadora, conocida hoy en día como un modelo paramétrico. Los cambios en el modelo o en sus variables de entrada pueden propagarse a través de las funciones explícitas en el equipo para cambiar el modelo de salida de forma automática, lo que permite a los trapecios transformarse en rectángulos, y permite a los diseñadores empezar a dibujar antes de saber la forma exacta.

Cuando Sutherland creó Sketchpad, era optimista al pensar que los diseñadores ganarían flexibilidad mediante la transformación de un trabajo, que un ponente hizo con una hoja de papel al trabajo de un ingeniero de software (Timothy Johnson) hizo con una computadora lo suficientemente grande como para requerir su propio techo (TX-2). Sin embargo, cincuenta años más tarde, los arquitectos se sientan en las computadoras y cada vez es más común ver exactamente lo que Johnson estaba haciendo hace tantos años. Es un trabajo difícil. En esta tesis analizaré que, si bien los modelos paramétricos ofrecen la

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

flexibilidad necesaria para adaptarse a los cambios y procesos en el diseño arquitectónico, hay una tendencia para que los cambios rompan modelos paramétricos de forma inesperada. Estos problemas se parecen a los problemas de ingenieros de software cuando se encuentran con incógnitas de programación; problemas que han llevado a los diseñadores ya en 1993 a declarar que el modelado paramétrico es "más similar a la programación que al diseño convencional" (Weisberg 2008, 12 - 16). Sin embargo, a pesar de las comunes preocupaciones de cambios debido a las capacidades, en la actualidad, y de acuerdo con Robert Woodbury (2010, 66), "existe poca conexión explícita" entre la práctica del modelado paramétrico, la práctica de la ingeniería de software y lo computacional.

En esta tesis busco conectar estas tres prácticas considerando si los modelos paramétricos se pueden hacer más flexibles a través de la aplicación de los conocimientos de ingeniería de software y considerando aspectos computacionales de representación, visualización y en particular a los procesos del diseño arquitectónico con problemáticas particulares.

1.2 – Problemas con la flexibilidad

El diseño es un viaje de trazos que siempre se encuentran cambiando en sus representaciones. “El diseñador se pone en marcha…” (Nigel Cross 2006, 8 - 10), argumenta con el fin de “explorar, para descubrir algo nuevo, en lugar de regresar con un ejemplo de más ya familiar”. Muchas otras personas comparten el punto de vista de Cross que el diseño no es un simple salto dentro de una solución premeditada sino, mejor dicho, un viaje desordenado que requiere de incertidumbre y es caracterizado por iteración (Schön 1983; Lawson 2005). La clave facilitadora de este proceso son las representaciones externas, las cuáles sirven como punto de reflexión a lo largo del camino (Schön 1983). Cross formula que esas representaciones son necesarias desde que “Al diseñar parece, que es difícil conducir únicamente por procesos internos mentales”. Todavía las representaciones toman mucho tiempo para producirlas, y toman mucho tiempo modificarlas. Debido a lo anterior, un cambio inevitable ocurre durante la etapa de diseño, el arquitecto o diseñador necesariamente gasta tiempo cambiando o creando nuevas representaciones.

Desde el inicio más complicado, la digitalización de la representación, modelos y visualizaciones en la arquitectura se han concernido con los cambios en la facilidad para las representaciones del proceso de diseño y el producto virtual arquitectónico. Cuando Ivan Sutherland creó el software Sketchpad, utilizó un tiempo muy amplio para considerar que “el cambio de una parte crítica del modelo resultará automáticamente en cambiar las partes apropiadas del sistema particular” (Sutherland 1963, 22). En palabras actuales, este procedimiento podría ser descrito como paramétrico, lo que significa que la geometría, como parte relacionada en un sistema particular, es una función explícita de una lista de parámetros definidos, cuando menos, la parte crítica. Timothy Johnson demostró, que un diseñador, utilizando el Sketchpad, podría cambiar su pensamiento en la relación entre objetos, la parte crítica y el software Sketchpad podría adaptar automáticamente los objetos, las partes relacionadas, para satisfacer su relación. Por ejemplo, un arquitecto podría decidir que dos lados de un trapecio podrían ser ortogonales (como se muestra en la figura 3). Luego modifica el parámetro controlando las relaciones de líneas, y estos cambios se filtran a través de funciones explícitas en el motor paramétrico del software Sketchpad para propiciar la remodelación automática de las líneas que se han encontrado ortogonales y paralelas, el trapecio es ahora un rectángulo. Al hacer modelos geométricos aparentemente fáciles de cambiar, el software Sketchpad y la introducción del diseño asistido por computadora prometía reducir las horas de trabajo empleadas para cambiar manualmente aspectos del dibujo o crear nuevas representaciones.

Esto es ahora más de 50 años después del software Sketchpad y las computadoras reemplazando las mesas de dibujo que una vez imitaron. Muchos nuevos caminos de generación de modelos paramétricos han sido desarrollados y adaptados a los procesos del diseño arquitectónico: desde el ya histórico software de modelado 3D (CATIA, SolidWorks, y Pro/Engineer), a guiones visuales de programación de datos y geometría (Grasshopper, GenerativeComponents y Houdini) y ambientes de programación textual (las interfaces de scripting y la mayoría de programas CAD). El aspecto más común de todos esos ambientes de modelado paramétrico y las relaciones que provocaron es la transformación de todas las partes relacionadas en un sistema. Esto es en la actualidad un camino muy popular para crear y modificar modelos digitales. Cincuenta años después de Sketchpad, Robert Aish y Robert Woodbury (2005, 151) dicen que todavía esperan noticias de que el modelado paramétrico podrá “reducir el tiempo y esfuerzo requerido para el cambio y la reutilización”.

Figura 2. Un trapezoide dibujado en el software Sketchpad.

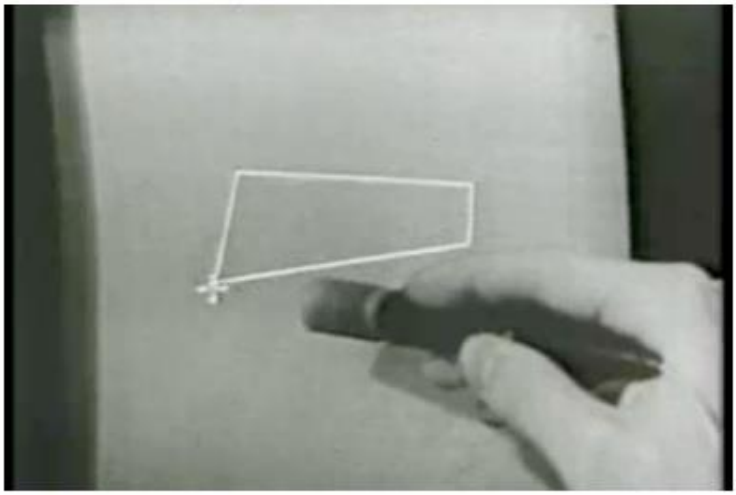
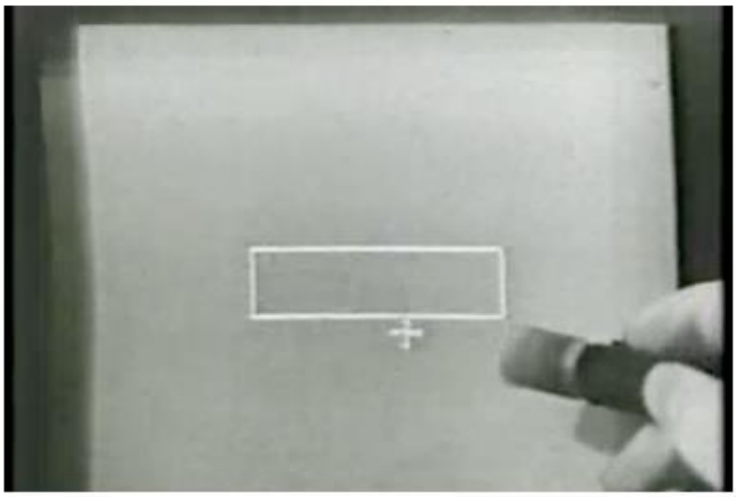


Figura 3. Timothy Johnson alcanzando un botón (fuera de cámara) para activar el motor de cambio paramétrico del software.

Figura 4. La función explícita del modelo crea ángulos ortogonales entre los lados del trapecio, transformándolo en un rectángulo.



Mientras todo lo anterior alude a pensar en la flexibilidad de la representación del diseño suficiente para reducir los tiempos invertidos en la remodelación y los cambios automáticos. Frecuentemente he encontrado que los modelos que he producido han sido tan enredados, que pueden ser acomodados o ser modificados por el simple hecho de cambiar o no algo trivial. Muchas veces tengo que comenzar nuevamente. Creo que no soy la única persona con experiencia en esta problemática. Veo el trabajo de algunos estudiantes y profesionales en la misma situación, escucho con frecuencia a algunos colegas reclamar este problema acerca de los errores de modelado y leo en primera instancia, cuentas en internet de arquitectos detallando y discutiendo el mismo problema.

- Cuando la topología de un proyecto cambia el modelo (paramétrico) generalmente necesita ser reconstruido. - *David Gerber 2007, p. 205*

- La reedición de gráficas relacionales o remodelación completa, son también aspectos comunes. - *Jane Burry 2007, p. 622*

- Un diseñador dice que quiere mover y torcer este muro, pero no anticipa que antes necesita un parámetro que pueda realizar un desplazamiento de la geometría y no hay lugar para realizar el cambio. Este aspecto muchas veces no es descifrado con anterioridad y muchas veces se inicia nuevamente. - *Rick Smith 2007, p. 2*

- El modelado paramétrico podrá requerir un esfuerzo adicional, podría incrementar la complejidad de las decisiones en diseño local e incrementar el número de objetos de atención que tendrán que ser pagados una vez finalizado el proceso de construcción. - *Robert Aish y Robert Woodbury 2005, p. 151*

- Los cambios requeridos por un equipo de diseño particular eran de una naturaleza compleja muy alta que el esquema de modelo paramétrico no podía cubrir sus demandas. Ellos tuvieron que replantear su idea inicial. - *Dominik Holzer, Richard Hough y Mark Burry 2007, p. 639*

- Si un cambio crítico es realizado, no hay otra solución que destruir completamente el modelo final y reiniciar en la decisión crítica del cambio. - *Mark Burry 1996, p. 78*

Estos autores han demostrado colectivamente que los modelos paramétricos empleados en la práctica tienen una propensa inflexibilidad, la cual en ocasiones conduce al modelo a producir errores y destruirse (las citas anteriores y sus implicaciones serán examinadas en el subcapítulo 1.4). Con frecuencia el único camino aceptable y cómodo es reconstruir la parte conflictiva del modelo. En el mejor de los casos, esto causa un retraso inesperado en el proceso de diseño.

En el caso más radical, el diseñador se encuentra disuadido de hacer el cambio y terminar con un diseño que no ha sido producido en el más estricto proceso de producción paramétrica y ha sido creado como consecuencia de las limitantes del modelado paramétrico, en conocimiento y entendimiento. A pesar de más de 50 años en refinamiento, mejoramiento y el actual incremento en la adopción del modelado paramétrico, los modelos de creación y procesos inflexibles todavía causan retrasos y dudas en la práctica arquitectónica, debido a la difícil colaboración, y a los diseñadores desalentados a partir de hacer cambios en sus procesos de trabajo.

1.3 – La crisis del software

En los años de 1960, alrededor del tiempo en el que Ivan Sutherland estaba creando Sketchpad, un gran número de proyectos de software fracasaron. Estas fallas "condujeron a grandes compañías (como IBM) al punto del colapso" que posteriormente fue nombrado como el premio *"Turing"* cuyo el ganador en ese año fue Niklaus Wirth (2008, 33) por su *Breve historia de la ingeniería del software*[1]. Las características más radicales de las fallas es que ocurrieron durante un periodo de progreso sustancial en la computación; un periodo dónde nuevos lenguajes de programación de tercera generación fueron desarrollados y utilizados en grandes procesadores con el fin de incrementar exponencialmente las velocidades (Wirth 2008, 33). Todavía, a pesar de esos avances, los proyectos fueron llegando significativamente como paquetes actualizados, sin modificaciones importantes y en algunos casos, algunos fueron abandonados. Un ejemplo notable es el proyecto del sistema IBM / 360, dirigido por Frederick Brooks, el cual en 1964 fue uno de los grandes proyectos de software jamás realizados. El tamaño del proyecto fue posible desde que las grandes computadoras habían llegado "muchos ordenadores de una gran magnitud y poder" (Dijkstra 1972, 861) pero el tamaño del proyecto también implicaba grandes dificultades relativas a la programación. Esos problemas no podían ser resueltos por la simple contratación de más programadores. "Añadiendo el poder del hombre para poder utilizar el software de manera tardía, hacía el proyecto tardío" dice Brooks (1975, 25), reflexionado sobre su dirección del sistema 360 en las memorias del libro, *The Mythical Man-month*. Al final, el sistema más ambicioso de unificación, el IBM / 360, así como muchos otros proyectos de software en los años 1960, llegó tarde y costó millones de dólares más que lo previsto.

Y en esta etapa comenzó la crisis del software, un periodo donde la esperanza ardía debido a los rápidos avances en la computación y que fue roto; no por las velocidades de procesamiento descarrilándose desde su improbable incremento, sino debido a la "aplastante complejidad de nuestro propio hacer" (Dijkstra 1997, 63); al cambio de simplificar el software escrito y programable. Wirth lo observa como "hemos reconocido de manera tardía que la programación es una tarea muy complicada, y que los problemas más difíciles no son triviales, siempre y cuando – o porque – las computadoras sean más rápidas y poderosas". La oración anterior refleja de algún modo la situación actual en la disciplina arquitectónica, dónde las vastas implementaciones en el modelado paramétrico en las décadas pasadas han expuesto las dificultades en la creación simple de un modelo paramétrico en los procesos de diseño particulares. En gran parte el mismo camino podría censurar a los arquitectos por las fallas a los cambios anticipados en un modelo paramétrico inflexible; los programadores temen a la cognición humana, no al poder de la computadora, este podría ser el factor de límite en la aplicación de la computación al proceso de diseño. Esta idea llamó mucho la atención, tanto que en la asamblea de la OTAN en 1968 se organizó un equipo de científicos para "arrojar más luz sobre los muchos problemas actuales del software" (Naur y Randell 1968, 14).

La conferencia de la OTAN referida a la Ingeniería del Software fue un momento decisivo. Entre las muchas discusiones de si alguien había muerto o enfermado por la crisis del software[2] se habló de "cambio de horarios del cuerpo humano en los procesos vitales primarios, un gran número de insectos y un producto inflexible difícil de manejar" (Naur y Randell 1968, 122). Esos problemas describen, palabra por palabra, los complicados cambios de muchos arquitectos cuando utilizan modelos paramétricos en procesos de diseño arquitectónicos en problemas particulares. En respuesta a estas dificultades, la inclinación de la conferencia de la OTAN fue reunir datos en lugar de confiar en la intuición.

El término de Ingeniería del Software se originó desde el título de la conferencia, el cual es un intento provocativo para “implicar la necesidad de manufactura de software para estar basado en los tipos de fundamentos teóricos y disciplinas prácticas, las cuáles son irracionales en las ramas establecidas de la ingeniería”. (Naur y Randell 1968, 13). Al respecto, la disciplina de Ingeniería del Software se origina como respuesta directa a la crisis del software, un intento para superar la crisis a través de un razonamiento entendible en la manufactura del software.

La curva Böhm

Barry Böhm no asistió a la conferencia de 1968 de la OTAN, pero claramente le influyó. A medida que los asistentes a la conferencia lo habían hecho, Böhm advirtió en 1972 que el software "llegaba a menudo tardío y poco fiable, y que los costos estaban subiendo" (Whitaker 1993, 300). Esto fue considerado una "impactante afirmación para el tiempo" (Whitaker 1993, 300) y la Fuerza Aérea de los Estados Unidos de América, que habían encargado el estudio, se negó a publicar los resultados, que "rechazaban fuertemente”. Böhm regresó cuatro años más tarde con un documento que lleva el mismo título que la conferencia de la OTAN: Ingeniería del Software (Böhm 1976). En este trabajo Böhm expone algo similar una vez más y produce más gráficos que muestran que el software se fue volviendo cada vez más caro que el hardware que se desarrollaba. Sin embargo, el documento es quizá más conocido por otro gráfico que contiene, una ilustración que ha llegado a ser conocida como la curva de Böhm (figura 5). La Curva de Böhm observa que a medida que un programa de ordenador se vuelve más desarrollado y complejo, también se hace más difícil de cambiar. Esta fue la misma observación que había hecho sobre proyectos de arquitectura en ese mismo año.

Los datos de las previsiones de la curva de Böhm advierten que hacer un cambio de última hora en un proyecto de software cuesta cien veces más que hacer el mismo cambio en el inicio del proyecto. En efecto, un proyecto de software - como un proyecto de arquitectura – se hace sustancialmente menos flexible con el tiempo y, como resultado, la capacidad del programador para hacer cambios disminuye en gran medida por el aumento del costo del cambio.

1 Mientras que las computadoras son una invención relativamente reciente, su rápido desarrollo ha dejado atrás una inmensa historia. En este capítulo sólo tocar dos aspectos de esta historia: la crisis del software y el costo del cambio de curva de Böhm. Para una historia más completa recomendaría empezando por Wirth (2008) Breve historia de la Ingeniería del Software, que hace referencia a una serie de escritos principales. Por desgracia, parece que nadie ha escrito un libro completo sobre la historia de la ingeniería de software - tal vez debido al tamaño y la velocidad de la industria - así que más allá del papel de Wirth, las mejores fuentes tienden a ser los libros y artículos publicados a partir del período, como Brook (1975) The Myth-man month. Numerosas guías de la mejor literatura se pueden encontrar en línea.

2 Las computadoras en 1968 fueron "cada vez más integradas en las actividades centrales de la sociedad moderna" (Naur y Randell 1968, 3) y muchos en la conferencia estaban preocupados de que los fallos de software llegarían a perjudicar a los que ahora se basan las computadoras.

Figura 5. La curva de Böhm en una representación anterior a la curva final. Prestar atención a la impresión logarítmica que hace Böhm de los datos.

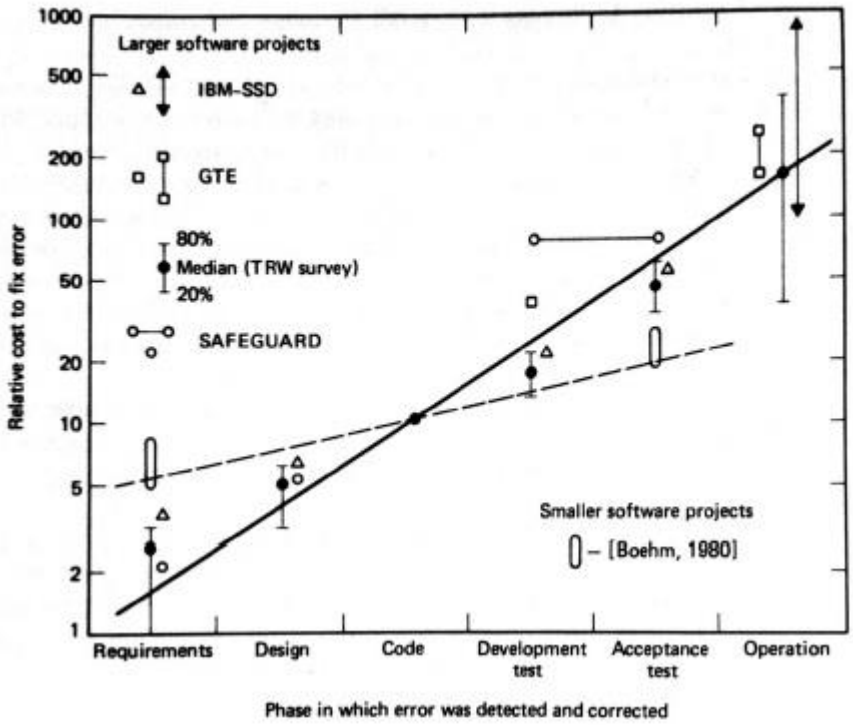
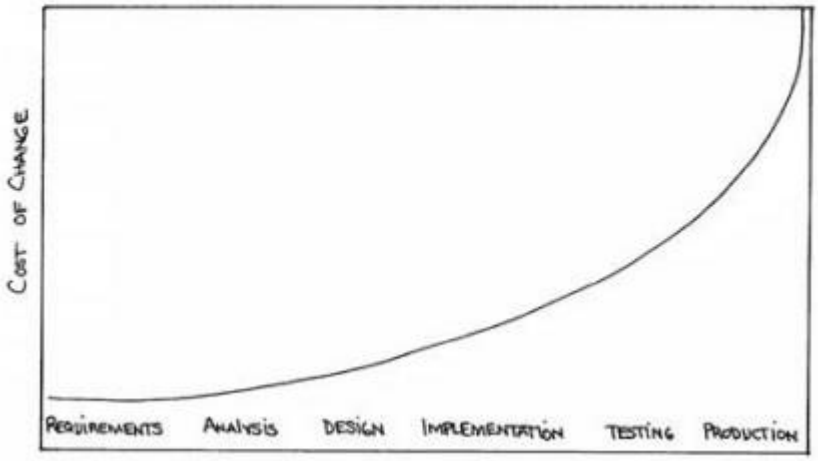


Figura 6. La curva de Böhm impresa en una escala linear.



1.4 – El parametricismo es paramétrico

El parametricismo es una noción deliberadamente provocativa, alegando que el "posmodernismo y el deconstructivismo eran medios episodios transitivos" (Schumacher 2010, 43) y que el diseño paramétrico sería el "gran nuevo estilo después de modernismo" (Schumacher 2009, 16) identifica al parametricismo con una lista de *"Heurística negativa"* parecido a "primitivas geométricas rígidas y vacías" y con una "yuxtaposición negativa". Compensa la anterior afirmación con el conjunto de *"Heurística positiva"* incluyendo "considerar todas las formas a ser paramétricamente maleables" y "diferenciación gradual (en varios rangos)" (Schumacher 2009, 16).

Schumacher ilustra su heurística parametricista prácticamente en los exclusivos proyectos de Zaha Hadid. En una revisión de proyecto, tuve la oportunidad de cuestionar a Patrick Schumacher sobre la falta de proyectos y su discurso teórico en otros arquitectos y sus posturas de diseño, Schumacher – el cual tiene un doctorado en filosofía – comento "Yo soy un arquitecto practicante, pero antes soy un teorista". Por esta razón, Schumacher no quiere decir que construye modelos paramétricos en su práctica como arquitecto. Schumacher no escribe nada relacionado con utilizar modelos paramétricos y no puedo encontrar evidencia que Schumacher crea modelos paramétricos con Zaha Hadid. Mejor dicho, la práctica de Schumacher consiste en gran medida de la revisión de otros arquitectos y su producción en relación a modelos paramétricos. Considerando la perspectiva de Schumacher, es algo comprensible que él decía "la emergencia de un nuevo estilo en la época… es más importante que las innovaciones metodológicas y de procedimiento a través de técnicas específicas computacionales" (Leach y Schumacher 2012). Después de todo, son las salidas estilísticas y no la metodología y el procedimiento lo que Schumacher ve. En este sentido Schumacher no se diferencia de muchos otros teóricos que también han definido el modelo paramétrico en términos y consecuencias de lo que se produce con un modelo computacional. En respuesta a la pregunta ¿Qué es el modelado paramétrico? El parametricismo representa una posición extrema, de la que muchos arquitectos gustan distanciarse, sin embargo, todavía es una posición en la cual han llegado muchos otros focalizándose en definiciones de modelado paramétrico y en su práctica.

Modelado contra diseño

En contraste con Schumacher, mi tesis se focaliza sobre las metodologías e innovaciones de procedimientos relacionados con el modelado paramétrico computacional en lugar de centrarse en enunciar la emergencia de un nuevo estilo de época del diseño paramétrico y computacional. El discurso alrededor del diseño paramétrico (si el diseño paramétrico es tomado como un verbo para describir el proceso de diseño con un modelo paramétrico, o como un sustantivo para describir los resultados de este proceso) parece poco probable llegar a una solución en un futuro próximo, si se necesita.

Dejaré lo anterior para que otros puedan debatir como el diseño paramétrico encaja en la cultura más amplia de la arquitectura; por ahora hay una urgente necesidad de comprender los retos tecnológicos que presentan los modelos paramétricos. Encontrar maneras de hacer un modelo paramétrico más flexible puede tener ramificaciones en términos de diseño arquitectónico, pero las ramificaciones inmediatas serán para las multitudes de los arquitectos que utilizan actualmente los modelos paramétricos en su práctica. Por lo tanto, mi investigación se centra casi exclusivamente en el modelado computacional escrito y gráfico, contemplando un debate particular en torno a las implicaciones de diseño.

1.4.1 - Los cristales de James Dana

El término *paramétrico* se originó en las matemáticas, pero aquí no comenzaré un debate de cuando inicialmente los diseñadores y arquitectos comenzaron a utilizar la palabra. David Gerber (2007, 73), en su tesis doctoral "*La práctica paramétrica*", agradece en los créditos a Maurice Ruiter por usar el término en un documento de 1988 con título "*Diseño Paramétrico*"[3]. 1988 fue también el año de fundación de la *Parametric Technology Corporation* (por el matemático Samuel Geisberg en 1985) lanzando el primer software de modelado paramétrico, Pro/ENGINEER (Weisberg 2008, 165). Pero Robert Stiles (2006) argumenta que la procedencia real del término ocurrió unas décadas antes, en escritos de 1940 de un arquitecto italiano de nombre Luigi Moretti (Bucci y Mulazzani 2000, 21). Moretti escribió extensivamente sobre la "Arquitectura Paramétrica", en dónde define un estudio de sistemas de arquitectura con el objetivo de definir las relaciones entre las dimensiones dependientes a partir de varios parámetros. Moretti usa el diseño de un estadio como ejemplo (figura 9), explicando como la forma del estadio puede derivar de diecinueve parámetros relacionándolos con ángulos de visión y con los costos más económicos del concreto (Moretti 1971, 207). Versiones del estadio paramétrico diseñado por Moretti fueron presentados como parte de su exhibición *Arquitectura Paramétrica* en la Doceava Trienal de Milán en 1960. En los años siguientes a la exhibición, entre 1960 y 1965, Moretti diseño el Complejo de Puente del Agua, el cual "*se cree que es el primer mayor trabajo de construcción significante con el uso de las computadoras*" (Livingston 2002). El Complejo del Puente del Agua es mejor ahora conocido por el escándalo de interrupciones telefónicas que tuvieron lugar aquí y "apenas es discutible" (Stiles 2006, 15); incluso por muchos arquitectos que utilizan computadoras para crear modelos paramétricos computacionales, argumentan que Moretti fue el pionero.

Moretti no temió a la oscuridad tanto como temía al uso incorrecto de términos matemáticos como el de *paramétrico*. Escribió a su amigo Roisecco que la "inexactitud [respecto a los términos matemáticos], es en verdad, más temido que la anterior ignorancia [cuando no conocían los términos ni los trabajos previos de Moretti]" (Moretti 1971, 206). El término ha tenido una larga historia en las matemáticas y en algunos ejemplos anteriores he podido encontrar que *paramétrico* ha sido utilizado para describir modelos tridimensionales que han llegado casi más de 100 años antes que los escritos de Moretti. Un ejemplo de lo anterior es el documento de James Dana escrito en 1837 sobre el Dibujo de las Figuras de Cristal (otros ejemplos de este periodo incluyen: Leslie 1821; Earnshaw 1839)[3].

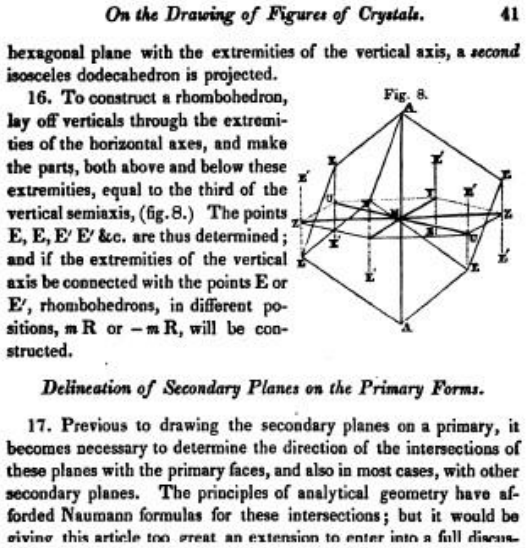
3 A través de la búsqueda en Google NGrams (http://books.google.com/ngrams/) fui capaz de encontrar la palabra "paramétrica" de la colección de libros que Google Books ha escaneado. Mientras James Dana 1837 es uno de los resultados más convincentes, otros ejemplos incluyen a Samuel Earnshaw 1839 que escribió sobre "hiperbólica en superficies paramétricas" deformadas por líneas de fuerza en un documento que dio origen al "Teorema de Earnshaw"; y Sir John Leslie 1821 que demostró la auto-similitud de las curvas catenarias utilizando "círculos paramétricos" en su libro sobre este análisis geométrico particular. Google ha escaneado solamente una colección limitada de libros por lo que puede haber ejemplos incluso anteriores que no fueron devueltos en estas búsquedas. Sin embargo, los escritos de Dana en 1837 son anteriores significativamente a cualquier otro que he encontrado en varios documentos de diseño paramétrico para el primer uso del término en relación con el dibujo.

En este documento James Dana explica los pasos generales para dibujar un rango de cristales y las disposiciones provistas para la variación usando un lenguaje acorde con parámetros, variables y radios. Por ejemplo, el paso 18 (figura 7) del esquema del dibujo, Dana Explica al lector como inscribir un plano paramétrico sobre un prisma:

> Si el plano a ser introducido, en dónde el radio paramétrico es 4P2, el cual es 4: 2: 1, deberíamos de estar en las mismas marcas de 4 partes de e, 2 de e' y 1 de ë. *Dana 1837, 42*

En esta cita Dana se encuentra describiendo la relación entre tres parámetros del plano (4: 2: 1) y la respectiva división de líneas e, e' y ë. El resto del documento de veinte páginas contiene enunciados similares que explican cómo varios parámetros se filtran a través de largas ecuaciones que afectan las variaciones de dibujo de los cristales. Las ecuaciones de Dana sobre los cristales se asemejan a las de varios arquitectos usadas 159 años después para el desarrollo de modelos y edificios *paramétricos*, iniciado por los trabajos que Moretti (1957, 184) ha llamado incidentalmente como el "*esplendor cristalino*".

Figura 7. Las instancias de los dibujos de cristales de James Dana. Arriba: Coordinación del sistema de coordenadas (Dana 1857, 41). Abajo: Impacto de cambio de la relación del borde de la arista (Dana 1857, 43).



On the Drawing of Figures of Crystals. 41

hexagonal plane with the extremities of the vertical axis, a *second* isosceles dodecahedron is projected.

16. To construct a rhombohedron, lay off verticals through the extremities of the horizontal axes, and make the parts, both above and below these extremities, equal to the third of the vertical semiaxis, (fig. 8.) The points E, E, E′ E′ &c. are thus determined; and if the extremities of the vertical axis be connected with the points E or E′, rhombohedrons, in different positions, m R or − m R, will be constructed.

Fig. 8.

Delineation of Secondary Planes on the Primary Forms.

17. Previous to drawing the secondary planes on a primary, it becomes necessary to determine the direction of the intersections of these planes with the primary faces, and also in most cases, with other secondary planes. The principles of analytical geometry have afforded Naumann formulas for these intersections; but it would be giving this article too great an extension to enter into a full discus-

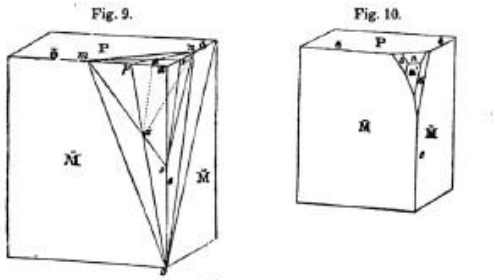


Fig. 9. Fig. 10.

perimeters of the planes *npb* (4P̆2) and *nmo* (P) intersect one another in the points *n* and *a*; consequently the line of intersection, between these two planes must be situated between these points, and therefore the direction of the intersection of P and 4P̆2 is *na*.

The planes *nmb* (2P) and *npb* (4P̆2) intersect in the line *nb*, and therefore the intersection of 2P and 4P̆2 is in the direction of *nb*.

En los escritos de James Dana, el término no tiene un gran significado. Dana no describió sus dibujos como paramétricos, ni tampoco lo reclama, como Schumacher (2009, 15) después lo haría, cuando utilizó ecuaciones paramétricas y computacionales para después "justificar la enunciación de un nuevo estilo en el sentido de un fenómeno de época". Mejor dicho, Dana utiliza el término en su sentido matemático original, una palabra matemática sin más énfasis que otros términos técnicos como *paralelo, intersección y plano*.

Cuando este término fue usado por Dana en 1837, o por los matemáticos actuales, el término paramétrico significa lo que la Enciclopedia Concisa de las Matemáticas llama como "una lista de ecuaciones que expresan una lista y secuencia de cantidades como funciones explícitas de un número de variables independientes, conocidas como parámetros" (Weisstein 2003, 2150)[4]. Esta definición establece dos criterios fundamentales:

1 – Una ecuación paramétrica expresa una "Lista de Cantidades" con un número de parámetros[5].

2 – Las salidas de datos (listas de cantidades) están relacionadas con los parámetros a través de "funciones explícitas"[6]. Este es un punto importante de atención para posteriores definiciones ya que algunos arquitectos contemporáneos han sugerido que las correlaciones constituyen las relaciones principales de los modelos paramétricos computacionales en los procesos de diseño.

Un ejemplo de ecuación paramétrica es la fórmula que define una curva catenaria:

$$x\,(a, t) = t$$
$$y\,(a, t) = a \cosh\,(t/a)$$

Estas dos fórmulas coinciden con el criterio de ecuación paramétrica. Primeramente, expresan una serie de cantidades (en este caso una cantidad *x* y una cantidad *y*) en términos de número de parámetros (*a*, la cual controla el cuerpo de la curva; y *t* que controla la longitud de la curva en el punto dónde ocurre la tangente). En segundo término, las salidas de datos (*x & y*) están relacionadas con los parámetros (*a & t*) a través de funciones explícitas (no hay ambigüedad en las relaciones entre estas variables). El origen del término paramétrico: una lista de cantidades y variables de datos matemáticos, numéricos, físicos y ambientales expresadas como una función explícita de un número de parámetros.

4 Esta definición es consistente con definiciones en otros diccionarios matemáticos y enciclopedias físicas y electrónicas. He seleccionado la definición de la Enciclopedia Concisa de las Matemáticas, así como al editor Eric Weisstein (quién también es el director editor de Wolfram Mathworld), considerado como fuente autorizada y especializada.

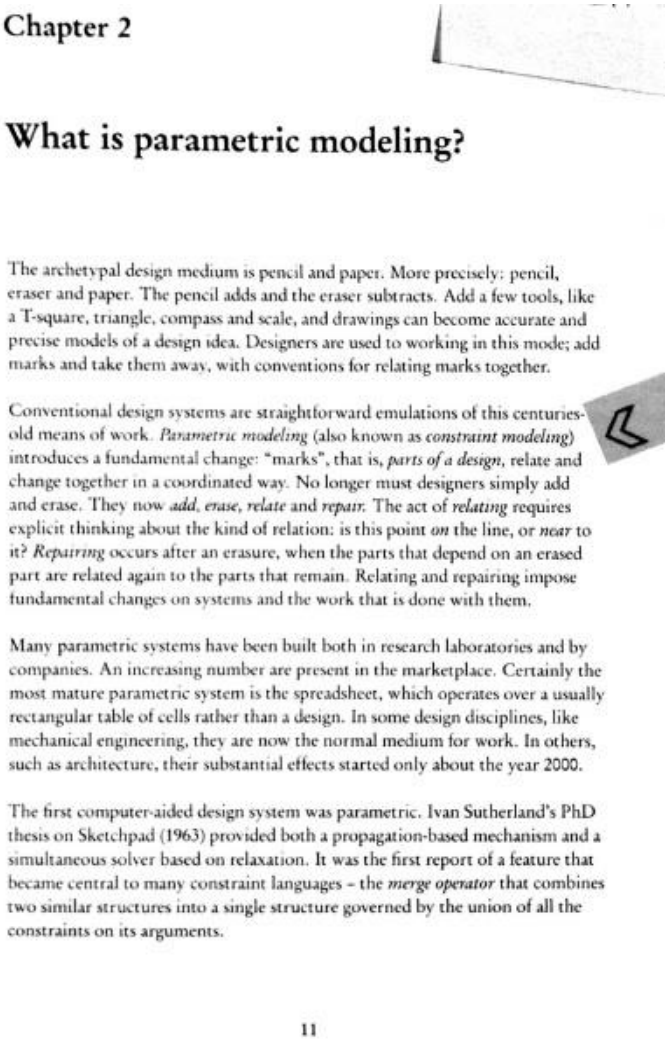
5 Parámetro puede tener un gran número de significados, siempre que sea usado por matemáticos. El gramático James Kilpatrick (1984, 211 – 12) cita una carta recibida por R. E. Shipley: "Sin ninguna justificación aparente, ni siquiera un indicio de la extensión razonable de su uso en las matemáticas, la definición de parámetro ha sido muy discutida, o peor aún, puede tener significados de consideración, como los factores, variables, la influencia, la interacción, la cantidad, la medición, la cantidad, la calidad, la propiedad, causa, efecto, modificación, alteración, computación, entre otros. La palabra ha llegado a ser dotada de 'falta de especificidad de múltiples ambigüedades".

6 Una función explícita es una función cuyo valor de salida se da explícitamente en términos de variables independientes. Por ejemplo, la ecuación x * x + y = 1 * y es la función implícita de un círculo. La función está implícita desde las salidas (x *&* y) y se definen en términos de uno al otro. Para hacer que la función sea explícita, X y Y se han definido en términos de una variable independiente. Así, la función explícita de un círculo se convierte en: x = cos (t), y = sin (t). Por una relación similar, diciendo que 'x es aproximadamente dos veces mayor que t' no es una función explícita ya que no hay ambigüedad en cuanto a la relación exacta entre las variables t y x (la relación no es explícita).

1.4.2 - Arquitectos y el término paramétrico

"¿Qué es el modelado paramétrico computacional?" es el título de cabecera del segundo capítulo del libro de Robert Woodbury *Elementos del Diseño Paramétrico* en 2010. El autor dedica doce páginas a esta cuestión, pero en lugar de responder directamente a la pregunta, desarrolla en la mayor parte de estas páginas la explicación del funcionamiento en una visión particular de la propagación y producción de modelos paramétricos. La respuesta más directa de Woodbury aparece en la primera página del capítulo (figura 8): "el modelado paramétrico introduce un cambio fundamental: marcas", es decir, "las partes del diseño, se relacionan y cambian juntas de una manera coordinada" (Woodbury 2010, 11). Pero Woodbury nunca hace una pausa para explicar cómo se relacionan las marcas en las diferentes relaciones encontradas en un método de plétora en modelado alternativos, en particular BIM. Esto no es para castigar a Woodbury, ya que *Elementos de Diseño Paramétrico* es uno de los libros iniciales sobre el modelado paramétrico computacional, pero esto es poner en relieve la dificultad, incluso de los expertos que han articuladamente tratado de responder a la pregunta básica *¿qué es modelado paramétrico?*

Figura 8. La página número once del libro Elementos del Diseño Paramétrico de Robert Woodbury. En el cual se pregunta *¿Qué es el modelado paramétrico?*, pero nunca se acerca a una respuesta concreta.



Chapter 2

What is parametric modeling?

The archetypal design medium is pencil and paper. More precisely: pencil, eraser and paper. The pencil adds and the eraser subtracts. Add a few tools, like a T-square, triangle, compass and scale, and drawings can become accurate and precise models of a design idea. Designers are used to working in this mode; add marks and take them away, with conventions for relating marks together.

Conventional design systems are straightforward emulations of this centuries-old means of work. *Parametric modeling* (also known as *constraint modeling*) introduces a fundamental change: "marks", that is, *parts of a design*, relate and change together in a coordinated way. No longer must designers simply add and erase. They now *add*, *erase*, *relate* and *repair*. The act of *relating* requires explicit thinking about the kind of relation: is this point *on* the line, or *near* to it? *Repairing* occurs after an erasure, when the parts that depend on an erased part are related again to the parts that remain. Relating and repairing impose fundamental changes on systems and the work that is done with them.

Many parametric systems have been built both in research laboratories and by companies. An increasing number are present in the marketplace. Certainly the most mature parametric system is the spreadsheet, which operates over a usually rectangular table of cells rather than a design. In some design disciplines, like mechanical engineering, they are now the normal medium for work. In others, such as architecture, their substantial effects started only about the year 2000.

The first computer-aided design system was parametric. Ivan Sutherland's PhD thesis on Sketchpad (1963) provided both a propagation-based mechanism and a simultaneous solver based on relaxation. It was the first report of a feature that became central to many constraint languages – the *merge operator* that combines two similar structures into a single structure governed by the union of all the constraints on its arguments.

11

Definiendo que el modelado paramétrico computacional y qué es lo que lo hace único, es un primer paso importante para la identificación de retos idiosincrásicos en modelos computacionales. En las páginas anteriores se describió una serie de definiciones que varios arquitectos han presentado: a partir de la definición histórica del término paramétrico, a través de afirmaciones de que todo diseño es paramétrico, y que el cambio se da en las herramientas, el parametricismo define a lo paramétrico. Al hacerlo así hago caso de las muchas definiciones contemporáneas que tienden a privilegiar lo que los modelos paramétricos representan (en términos de modelo de comportamiento o resultados estilísticos), y de cómo los modelos paramétricos pueden ser (a través de la construcción y el mantenimiento constante de las relaciones logarítmicas gráficas y textuales) y que distinguen al modelado paramétrico computacional de otras formas de representación y aplicados a un proceso de diseño arquitectónico particular.

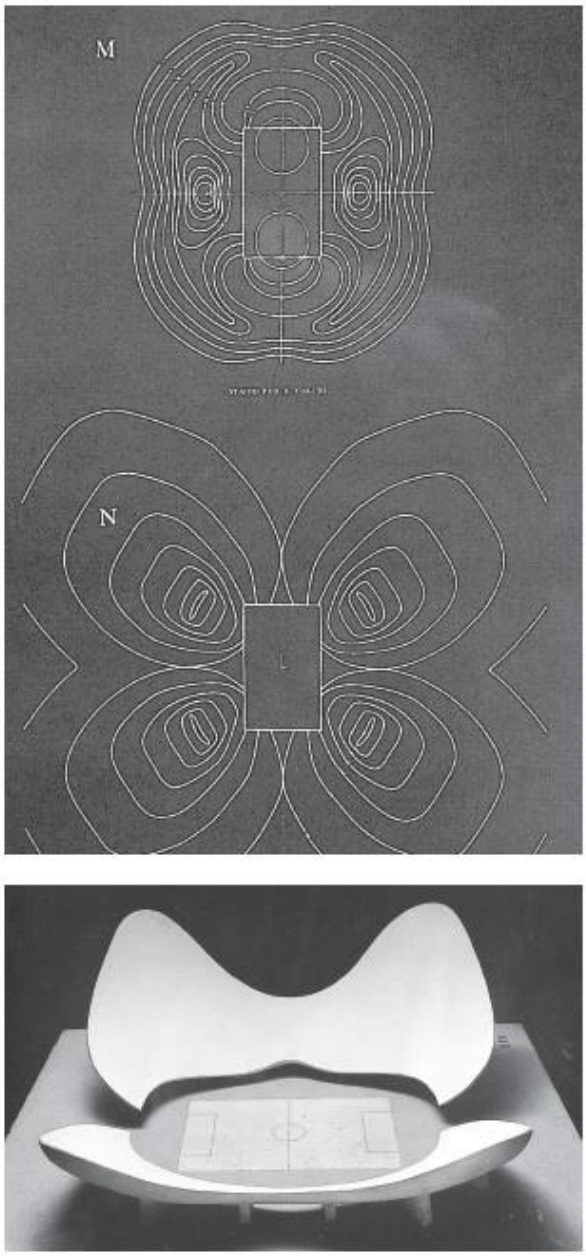


Figura 9. Estadio diseñado por Luigi Moretti en la exhibición *Arquitectura Paramétrica* de 1960 en la Doceava Trienal de Milán. Cada diseño de la cubierta del estadio deriva de un modelo paramétrico que consiste de diecinueve parámetros. Arriba: Los planos del estadio en las versiones *M* y *N* mostrando las curvas “equiatractivas”. (Converso y Bonatti 2006, 243). Abajo: Una maqueta del estadio en su variación formal *N*.

1.4.3 – La producción paramétrica

Si se le preguntara a un arquitecto describir un medio para el diseño en la arquitectura, el cual fomenta la creatividad y la exploración, probablemente no contestaría “un conjunto de ecuaciones que expresan un modelo geométrico como funciones explícitas de una serie de parámetros". Sin embargo, las funciones explícitas y parámetros son el medio de elección de muchos arquitectos que diseñan con modelos paramétricos y computacionales. Entender por qué los arquitectos optan por utilizar modelos paramétricos - un medio aparentemente contradictorio para la creatividad y la exploración - es un paso crucial hacia la comprensión de los desafíos asociados con el modelado paramétrico computacional.

La expresión de las intenciones de diseño con parámetros y funciones explícitas, requiere de un camino diferente de pensamiento al cual muchos de los arquitectos no están acostumbrados. Adicionalmente al pensamiento sobre el proceso de diseño, los arquitectos deberán trabajar con modelos paramétricos, así como también pensar en la secuencia lógica de fórmulas, parámetros y relaciones que explican cómo crear los diseños (Aish 2005, 12; Woodbury 2010, 24-25). Sin embargo, existen siempre dudas sobre el pensamiento paramétrico computacional o el pensamiento algorítmico. Aprender a pensar paramétricamente en los procesos de diseño arquitectónico es una “habilidad muy complicada de ganar al inicio, y con una facilidad no muy relativa”, lo anterior es una afirmación en una entrevista que he leído de Mark Burry (2011, 38). Al mismo tiempo, aprender diseño como un objeto ajeno puede ser muy difícil, la precisión lógica también puede, en algunas ocasiones, ser muy divertida para diseñadores que realmente tratan de imponerse a la dificultad y encontrar una solución a un problema específico en el proceso de diseño arquitectónico, así como a las constantes matemáticas, algorítmicas y lógicas que involucran todo proceso de diseño en arquitectura.

Para los diseñadores y arquitectos que gustan de resolver y producir modelos paramétricos computacionales en relación a sus estados explícitos (y por consiguiente considerar) cada relación geométrica (Aish 2005, 12; M. Burry 2001, 38-39; Kilian 2006, 300-3). Sin embargo, el gran beneficio de aprender a pensar paramétricamente a través de las herramientas computacionales es el costo de los cambios del diseño arquitectónico en todas sus etapas.

1.4.4 – El costo y la dificultad del cambio

En 1976 Boyd Paulson dibujó una gráfica (figura 10) mostrando que el nivel de influencia del diseñador es muy alto en el proyecto de arquitectura y que decrece en las siguientes etapas en el proceso de diseño del proyecto. Paulson (1976, 588) se focaliza en la primera decisión en cara al inicio de cualquier proyecto arquitectónico – en los momentos que se debe o no iniciar un proyecto – ya que tendrá gran influencia en el desarrollo del futuro proceso de diseño arquitectónico en todas sus etapas. Él argumenta que, en todas las subsecuentes decisiones del proyecto, su influencia tendría que disminuir e implementar generalmente más costos en las siguientes etapas del proceso de diseño en el proyecto arquitectónico. En otras palabras: en cuanto el diseño sea más extenso, ellos tendrán mayor dificultad en el cambio. Paulson publicó su observación en unos pocos textos de administración en su oficina de arquitectura (Barrie y Paulson 1991) pero su idea no llegó a ser tan conocida y discutida.

El gráfico de Paulson reapareció en mayo de 2001 en un centro turístico en México. Los líderes de HOK (una de las mayores firmas de arquitectura del mundo) se habían reunido en un complejo de hotel para discutir "estrategias clave para el futuro" (HOK 2012). Durante las discusiones, Patrick MacLeamy presento un gráfico (figura 11) el cual muestra que un diseñador tiene mucha capacidad "para impactar el costo y función de un proyecto y las capacidades "(MacLeamy 2010) al inicio de un proyecto, y que esta capacidad disminuye durante las etapas siguientes del proyecto, mientras que el costo de hacer cambios en el diseño aumenta. El gráfico que MacLeamy presentó era idéntico al de Paulson. MacLeamy reclamó el trabajo como propio (tal vez inconsciente de los esfuerzos de Paulson) y HOK pasó a promover la gráfica de Paulson con el nombre Curva de MacLeamy (HOK 2012) - un nombre que se ha establecido en parte gracias a la influencia en marketing de HOK[7]. Dos años después de que MacLeamy presentó la gráfica a los líderes de HOK, se disculpó como director, una posición que había deseado por más de una década.

Como director de HOK y de la junta internacional de edificación *buildingSMART*[8]. MacLeamy utilizó esta curva como la justificación de todos sus proyectos arquitectónicos. MacLeamy (2010) puntualiza realizar las decisiones más importantes al inicio de cualquier proyecto (exponiendo siempre su éxito posterior) desde que la curva muestra que los cambios son cada vez menos costeables para iniciar el proyecto en comparación con la etapa final del mismo. Paulson llegó a las mismas conclusiones con su gráfica y sugirió que el conocimiento correspondiente a la construcción debería de ser incorporado en las etapas iniciales del proceso de diseño. Los ejemplos más sofisticados de carga frontal están dadas por MacLeamy (2010) que aboga por las dos condiciones siguientes: Envío de Proyecto Integrado (IPD), que se amalgama actualmente contra todas las partes del proyecto para guiar al equipo de diseño a encontrar soluciones viables tempranas en todo los procesos del proyecto; y BIM, que proporciona una base de datos de proyecto central para mejorar la comunicación entre los miembros del equipo y al mismo tiempo ayudar a las simulaciones en las fases iniciales y posteriores, así como a la documentación del proyecto. Estas ideas han sido ampliamente difundidas y la concepción de MacLeamy de carga ha informado al estudio de la arquitectura contemporánea en todo, desde el Instituto Americano de Arquitectos (AIA) las directrices para IPD y los manuales de instrucciones para Autodesk Revit.

7 Estoy muy agradecido con Ehsan Baharlou por llamar mi atención sobre el trabajo de Paulson en un comentario que me realizo en mi blog dedicado al diseño y a la visualización arquitectónica 3D. Lo mejor que puedo decir, sin esta investigación previa solo habría comentado la curva MacLeamy.

8 BuildingSMART es un consorcio influyente en la industria CAD y usuarios que desarrollan estándares abiertos para la descripción de los edificios. Son quizás los más conocidos para el desarrollo de la norma de la CFI, que facilita la interoperabilidad entre el software BIM.

Figura 10. La curva de Paulson (1976, 588). En el texto que acompaña a eta gráfica, el autor hace énfasis en los beneficios de realizar y tomar decisiones iniciales en el proceso de diseño cuando el nivel de influencia es alto.

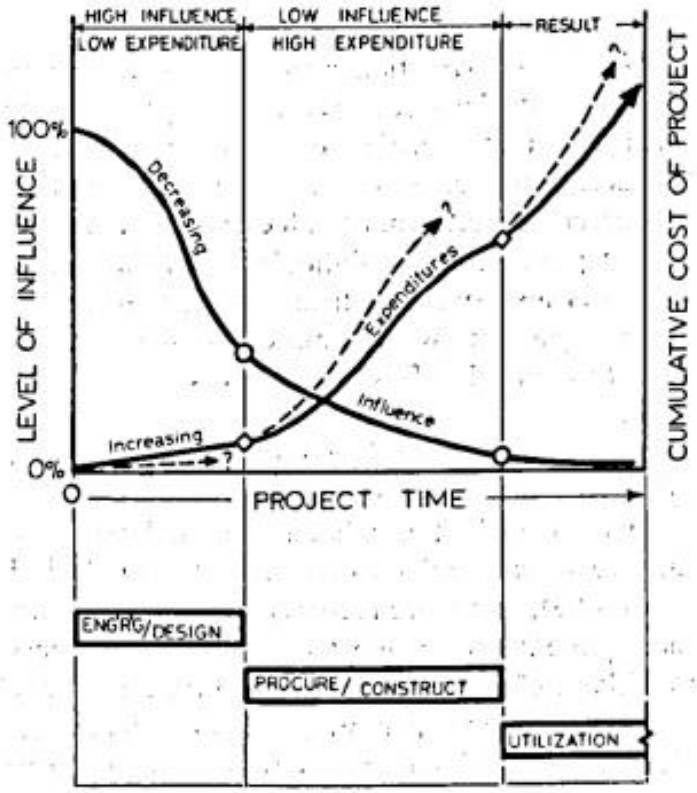
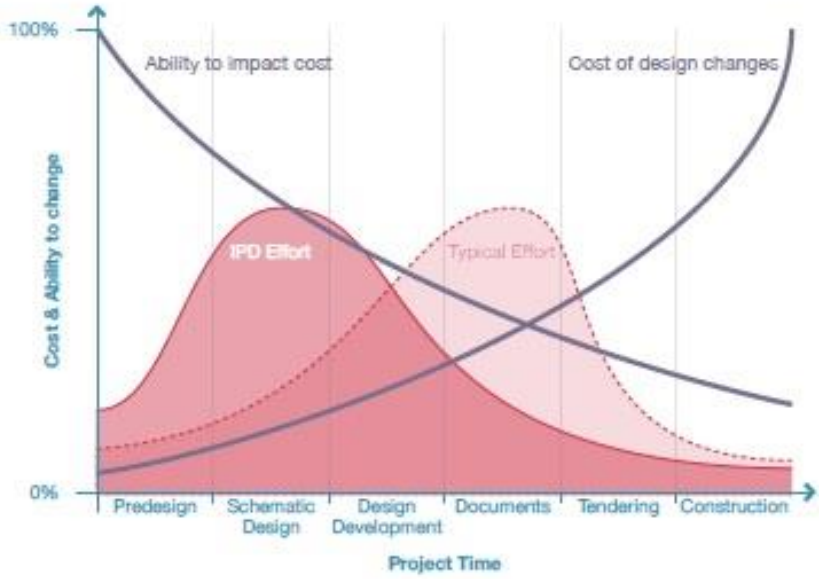


Figura 11. La curva de MacLeamy (2001). Sobresalta la importancia de hablar sobre el éxito del diseño arquitectónico típico y lo relaciona con la etapa inicial del proyecto. En teoría, esto significa que los diseñadores estarán trabajando cuando sus decisiones tengan un mayor impacto e incrementen la complejidad del mismo, así como los costos asociados.



La introducción del modelado paramétrico fue motivada por el deseo de reducir el costo del cambio. Esta motivación es discutida por Samuel Geisberg, el fundador de *Parametric Technology Corporation*, durante una entrevista con la Semana de la Industria en 1993:

“El objetivo es crear un sistema que sea lo suficientemente flexible para permitir al ingeniero o diseñador producir fácilmente una variedad de diseños. El costo de hacer cambios en el diseño, debería estar lo más cerca posible a cero. Además, el software CAD - CAM tradicional de la época, realmente está restringido a cambios de bajo costo final en la misma fachada durante el proceso de diseño en la ingeniería¨.

Geisberg citado en: Teresko 1993, p. 28

Los comentarios de Geisberg sugieren que en lugar de buscar un costo de cambio en la curva de MacLeamy, concluyen que los esfuerzos de diseño deben ser desplazados hacia un objetivo en común “misma fachada del proceso de diseño que el proceso de ingeniería," una mejor conclusión puede ser cambiar el costo de cambio en la curva para que el "costo de hacer cambios de diseño sea tan cerca de cero como sea posible”. "En teoría, un modelo paramétrico ayuda a reducir el costo del cambio que proporcionan los parámetros del modelo y las funciones explícitas requieren menos esfuerzo para cambiar de métodos de modelado alternativos”. Geisberg llama a lo anterior flexibilidad (Teresko 1993, 28), la cual puede ser entendida como una medida del costo de los cambios de diseño y, por consecuencia, un componente en la capacidad del diseñador para proponer variaciones.

La flexibilidad define el aspecto central del modelado paramétrico. Al mantener un modelo flexible que el diseñador puede variar y darse el lujo de hacer cambios, lo cual es importante dada la inevitabilidad del cambio en un proyecto de arquitectura convencional. Mientras que algunos cambios se pueden anticipar y quizá algunos de primera instancia, muchos cambios provienen de fuerzas externas a la esfera del diseñador. Por ejemplo, el cliente puede cambiar algo en breve, los políticos pueden cambiar la legislación y las fuerzas del mercado pueden cambiar el precio de los materiales. Otros cambios se deben a que el diseño es un proceso de generación de conocimiento. A menudo, es sólo a través de la iteración, la exploración y la reflexión del problema, y en menos importancia, la respuesta del objeto - diseño. Frente a estos cambios inevitables, la flexibilidad de los parámetros de un modelo paramétrico computacional y funciones explícitas se convierten en un medio fascinante de diseño; uno que muchos arquitectos emplean para ayudar a mejorar la capacidad del diseñador para producir e innovar en una gama alta de variaciones.

1.5 – La ingeniería del software: un cuerpo de conocimiento

Hay razones para sospechar que el cuerpo de conocimientos sobre la ingeniería del software también puede aplicarse a los procesos de diseño utilizando modelos paramétricos computacionales. Frederick Brooks (2010) hace una conexión similar en su libro *The Design of Design*, donde narra el diseño de su casa y relaciona esto a sus experiencias de gestionar el diseño arquitectónico en el sistema BIM/360 (2010, 257-346). Brooks (2010, 21) dice que el cambio es inevitable tanto para programadores y arquitectos, ya que ambos normalmente comienzan con "un vago e incompleto objetivo específico, o el objetivo principal" aclarado a través de la interacción de producir y cambiar prototipos. Estas dificultades se agravan en las dos prácticas, tanto por el hecho de que el costo del cambio en general se eleva exponencialmente a medida que progresa un proyecto, y por el hecho de que no están determinados los resultados que deben de ser expresados en las instrucciones lógicas precisas para la computadora. Aunque este problema es relativamente nuevo para los arquitectos que producen ingeniería de software en modelos paramétricos, el mismo problema ha sido un desafío durante décadas. La evidencia sugiere que los ingenieros de software han adquirido un conocimiento que durante este tiempo les permite un cierto control sobre el costo del cambio. Este conocimiento potencialmente podría hacer lo mismo en la arquitectura.

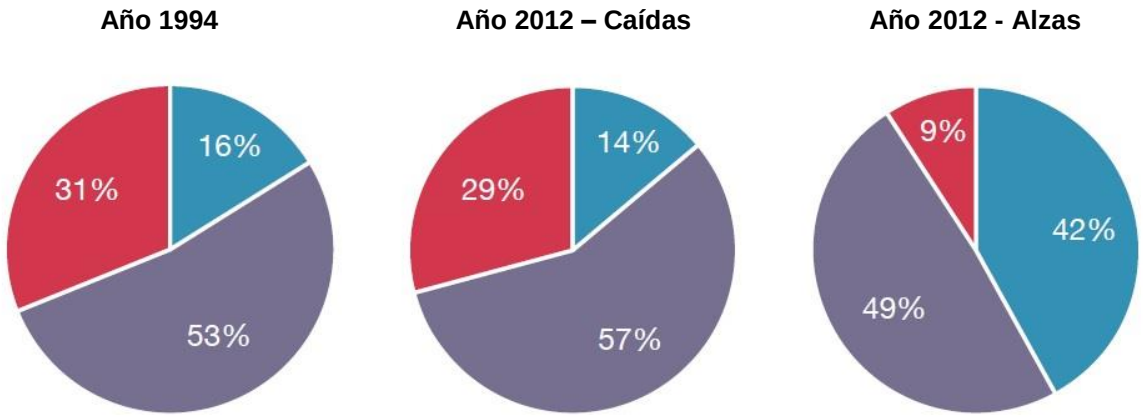


Figura 12. Porcentajes de éxito y fracaso de proyectos de software de acuerdo con el Grupo Standish (1994; 2012).

- Proyectos exitosos, entregados a tiempo y forma, sin incremento de precio y en tiempo de realización y entrega.
- Proyectos cambiados, desde su concepción original. Entregados tarde y con modificaciones.
- Proyectos fallidos, los cuáles fueron abandonados.

Hay dos advertencias principales en la aplicación de la ingeniería de software al modelado computacional y paramétrico. Un aspecto importante es que los ingenieros de software a menudo no son particularmente exitosos en lo que hacen. En promedio, el 49% de los proyectos de software que han utilizado un proceso de desarrollo ágil se encontrará con retos significativos, mientras que el 9% fracasará rotundamente. Sólo el 42% de los proyectos de software se entregan a tiempo, dentro del presupuesto, y con las características especificadas (figura 12). Si bien una tasa de éxito del 42% puede parecer bajo, el Grupo Standish (2012, 25) dice que esto representa el "remedio universal para el fracaso del proyecto de desarrollo de software", principalmente porque los ingenieros de software han tenido históricamente una tasa de éxito de sólo el 16% (figura 12). Por lo tanto, incluso los ingenieros de software siguiendo las mejores prácticas, todavía incluso se encuentran con más problemas que con éxitos.

Otra advertencia importante es que la producción de software es similar, pero no idéntica, a la producción en la arquitectura. En términos generales, algunos puntos comunes de diferencia incluyen los siguientes:

- **El usuario:** Los ingenieros tienden a hacer que el software sea utilizado por otras personas, mientras que los arquitectos en general, producen modelos paramétricos computacionales, ya sea para sí mismos o para sus colegas.

- **El producto:** Los ingenieros de software hacen software, pero en última instancia, los arquitectos hacen arquitectura en lugar de modelos paramétricos. Mientras que el software puede ser evaluado en sí mismo, un modelo paramétrico se valora normalmente para la arquitectura que produce.

- **Tamaño del equipo:** los equipos de ingeniería de software van desde individuos solitarios que construyen una sola aplicación, a miles de desarrolladores que crean un sistema operativo o programa específico. En comparación, los modelos paramétricos se hacen generalmente por equipos pequeños de personas.

- **Curso de la vida del proyecto:** los proyectos de ingeniería de software pueden durar desde unos pocos minutos a unas décadas, mientras que el código en un modelo paramétrico es poco probable que persistan más allá de un par de años (o quizás incluso meses).

Habrá numerosas excepciones a estas generalizaciones. El punto, sin embargo, es que mientras que los arquitectos e ingenieros de software comparten desafíos similares, no todos los aspectos de la ingeniería de software son igualmente relevantes para las circunstancias idiosincrásicas de modelado computacional o paramétrico. En esta sección se mostrará una gráfica SWEBOK[9] e hipótesis acerca de qué partes son más pertinentes para la práctica de modelado computacional y paramétrico en arquitectura.

9 Una gráfica y lineamientos SWEBOK son un tipo de representación esquemática y textual, la cual agrupa relaciones por contenidos y características similares y las relaciona con sus partes. Son muy utilizadas como sistemas de clasificación en la ingeniería del software.

1.5.1 – Clasificación de conocimiento computacional general

Ha habido un número de intentos para clasificar los conocimientos relativos a Ingeniería de Software. En 1997, el Instituto de Ingenieros Electrónicos y de Electricidad (IEEE) formaron un comité encargado para la creación del primer “Conjunto amplio de conocimientos para la ingeniería de software". Esta era una empresa controversial. La Asociación para Maquinaria Informática (ACM) temió que este cuerpo de conocimientos "probablemente proporcionaría la base para un examen de concesión de licencias de software, para ingenieros profesionales" (ACM 2000). El ACM, como muchos otros, retiró su apoyo al proyecto. La resolución del comité IEEE tardó de cuatro años a siete años de deliberaciones. Mientras tanto, Thomas Hilburn hizo a un lado el comité IEEE para producir su propia, y el primer, Cuerpo de Conocimiento de Ingeniería del Software Versión 1.0 (SWEBOK.1999; figura 13). Este documento resumió el conocimiento esperado de un programador quien había pasado tres años en la industria, y fue lanzado en conjunto con Donald Bagert (1999) con las directrices correspondientes para Educación Ingeniería del Software Versión 1.0 (1999). Finalmente, en 2004, un par de documentos similares fueron publicados por el comité IEEE: Alain Abran y James Moore (2004) *Guía para el Cuerpo de Conocimiento de Ingeniería del Software* (SWEBOK. 2004) junto con Jorge Díaz Herrera y Thomas Hilburn de (2004) *Ingeniería de Software* en 2004: *Lineamientos Curriculares para Pregrado en Programas de Grado en Ingeniería del Software* (2004).

La relación entre los diversos SWEBOK se muestra en la figura 13. Mientras que las taxonomías son diferentes, todos ellos utilizan el método de cascada como una plantilla para clasificar el proceso de ingeniería del software. Esto no es una aprobación del método de cascada desde la división del conocimiento que no tiene por qué prescribir su despliegue. Por ejemplo, los proyectos mediante una ágil metodología se aplican necesariamente a los conocimientos de la planificación, la codificación y experimentación, aunque no en la misma forma lineal como los proyectos que utilizan el método de cascada. Con cada clasificación se emplean los métodos de cascada a las diferentes etapas de los procesos, las principales diferencias entre los distintos SWEBOK son las distinciones del conocimiento en las distintas etapas de la gráfica. El SWEBOK segrega claramente estas zonas, con etapas en cascada o listas, confinadas a la categoría de ingeniería, que está separada de los fundamentos de computación por categoría y la categoría de gestión de software. Si bien la gestión de software vuelve a aparecer en todos los demás SWEBOK, los fundamentos de categorías computacionales son únicos para la gráfica y cubren áreas de conocimiento - como el hardware y la programación informática por idiomas - que son potencialmente aplicables a modelado paramétrico computacional en la arquitectura.

Por esta razón, he seleccionado el método SWEBOK para usarlo en la siguiente página como hipótesis sobre la que se aplican también, a los arquitectos en partes de la creación de modelos paramétricos y computacionales en el proceso de diseño arquitectónico. Sin embargo, dada la relativa homogeneidad de los diversos SWEBOK, me gustaría esperar resultados similares para utilizar cualquiera de las diferentes clasificaciones por SWEBOK.

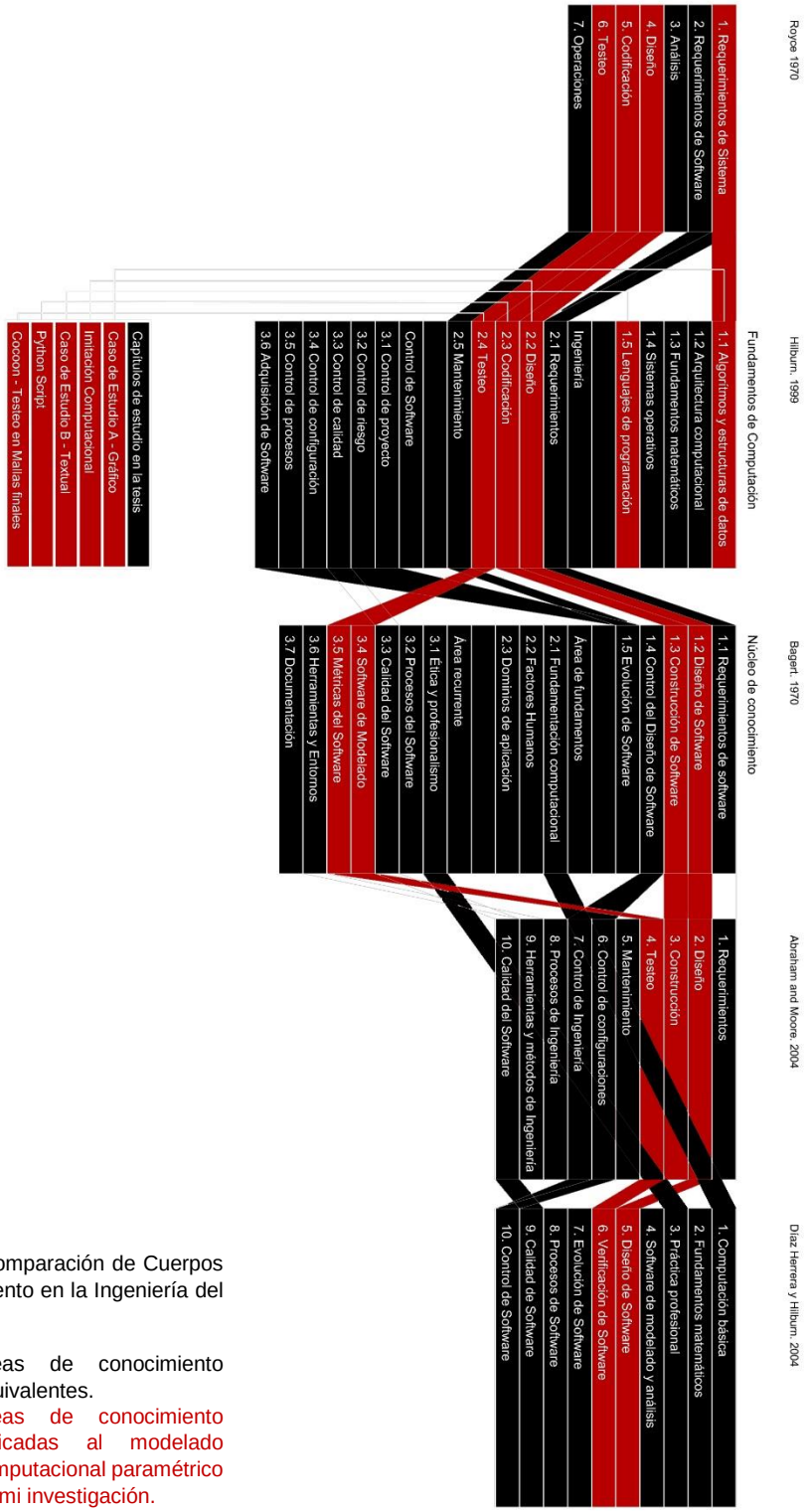


Figura 13. Comparación de Cuerpos de Conocimiento en la Ingeniería del Software.

- Áreas de conocimiento equivalentes.
- Áreas de conocimiento aplicadas al modelado computacional paramétrico en mi investigación.

1.5.2 – Fundamentos computacionales

Los *fundamentos de computación* [Categoría 1] en la gráfica de SWEBOK cubren las teorías y conceptos fundamentales de la ingeniería de software. Muchas partes de esta categoría son tan esenciales para la informática que ya necesariamente contribuyen al modelado paramétrico computacional en el diseño arquitectónico. Por ejemplo, la *arquitectura de computadoras* [Categoría 1.2] se refiere a la estructura subyacente de una computadora, que incluye la forma en que los transistores están dispuestos a permitir los cálculos más intensivos, y cómo las redes de intercambio de datos permiten la colaboración a distancia. Derivando los beneficios de este conocimiento, los cuáles no requieren de la intervención del ingeniero de software o modelador paramétrico, ya que está encapsulado dentro de un hardware de computadora. Lo mismo es cierto tanto en la *función matemática* [Categorías 1, 3], que proporciona la lógica formal para la programación y de funcionamiento en los *sistemas operativos* [Categorías 1, 4], que brindan el marco que respalda al software.

Mientras que la *arquitectura de computadoras* [Categorías 1, 2], *los fundamentos matemáticos* [Categoría 1.3], y *sistemas operativos* [Categorías 1, 4] han hecho grandes contribuciones a la ingeniería del software, estas contribuciones vienen - en muchos aspectos – independientes de las acciones de los ingenieros de software. Por proximidad, los diseñadores ya se encuentran beneficiados en estas áreas de los *fundamentos de la informática y lo computacional* [Categoría 1], siempre al comprar nuevo hardware o invertir en nuevos sistemas operativos y programas.

Los *algoritmos y estructuras de datos* [Categorías 1,1] no están integrados en el hardware y su deber es formar activamente una tarea en particular a la vez. Muchas investigaciones han indagado en la *adaptación algorítmica* [Categoría 1.1.2] y *estructuras de dato*s [Categoría 1.1.1] para el modelado paramétrico computacional. Ejemplos de *algoritmos existentes* [Categoría 1.1.2] que se utilizan en el modelado paramétrico, incluyen algoritmos para propagar los cambios a través de modelos paramétricos (Woodbury 2010, 15-16), la racionalización algorítmica para la simplificación de superficies complejas (Wallner y Pottmann 2011), algoritmos para la simulación de las propiedades físicas (Piker 2011), y muchos algoritmos propietarios incluidos en el software comercial y en los núcleos de geometrías complejas. Un trabajo similar se ha hecho sobre los *datos de estructuras* [Categoría 1.1.1] para desarrollar formatos de archivos especializados para cosas como compartir modelos BIM, que describen la geometría B-Rep [Boundary Representation], y guardar modelos paramétricos. Si bien hay margen para avanzar en estos *algoritmos existentes* [Categoría 1.1.2] y estructuras de datos [Categoría 1.1.1], con cualquier mejora, es probable que sean refinamientos de lo ya existente. Dada la madurez de la investigación en esta área, veo pocas oportunidades para hacer frente a la flexibilidad de los modelos paramétricos computacionales a través de hacer nuevas contribuciones a algoritmos ya definidos y estructuras de datos [Categoría 1.1].

Al igual que con los *algoritmos y estructuras de datos* [Categoría 1.1], ya existen muchos *lenguajes de programación* [Categorías 1, 5] para los arquitectos que crean modelos paramétricos computacionales. Cada *lenguaje de programación* tiene un estilo único para expresar conceptos, denominados *paradigmas de lenguajes de programación* [Categoría 1. 5. 2] (figura 14). El paradigma influye en cómo se resuelven los problemas en un idioma de programación determinado.

Figura 14. Los lenguajes de programación que los arquitectos utilizan, en categoría hecha por Appleby y VandeKoppel (1997, XIV) en la taxonomía de los paradigmas de los lenguajes de programación.

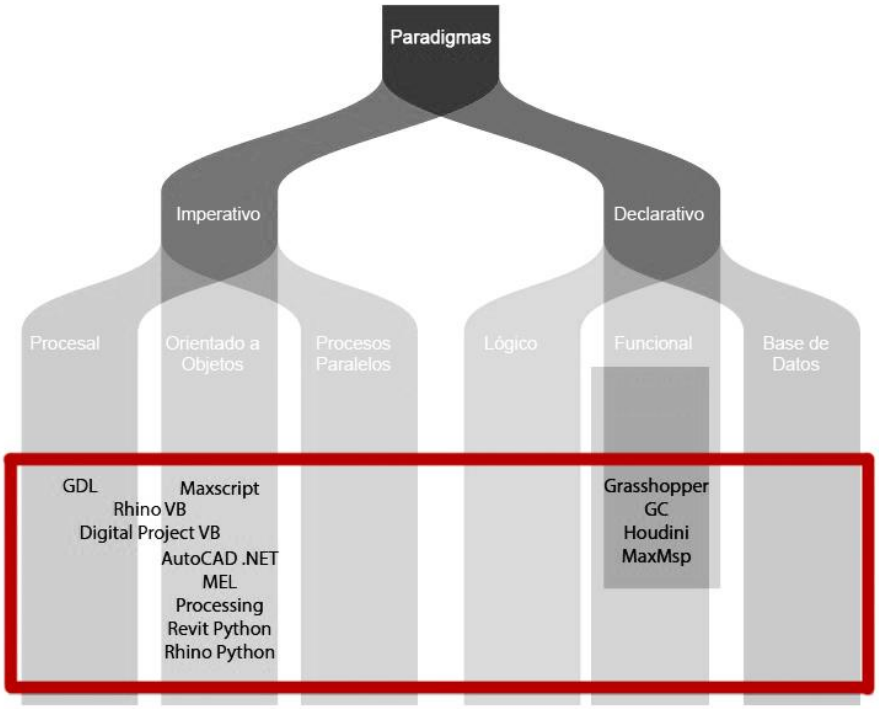
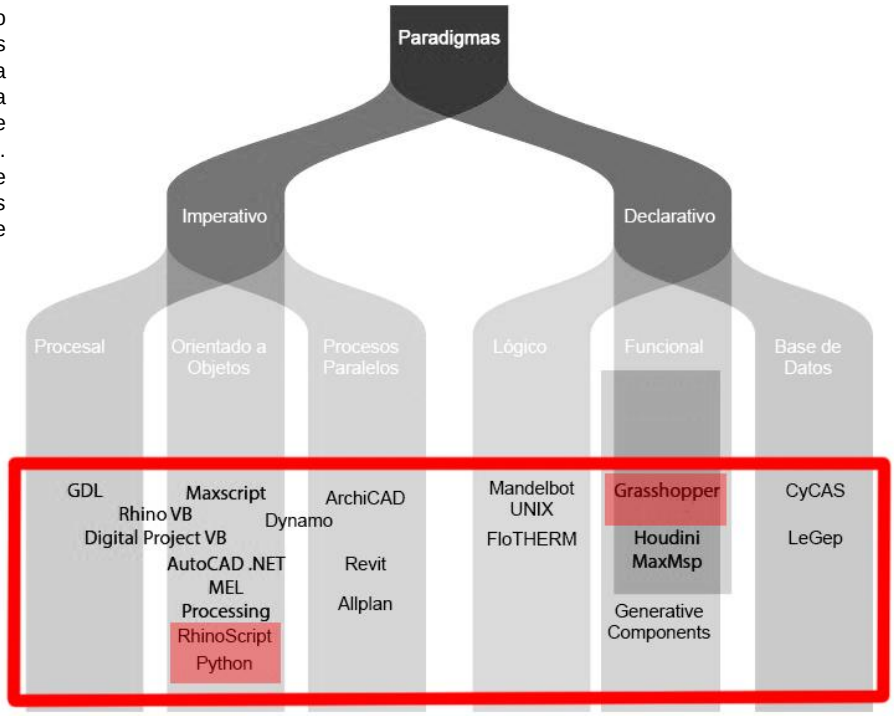


Figura 15. Los lenguajes de programación, así como algunos programas que los arquitectos utilizan en la actualidad (Categoría hecha en base a la taxonomía de Appleby y VandeKoppel). En cuadros en rojo, lo que van a ser empleados en los capítulos 4 y 5 en casos de estudio.



Por ejemplo, Appleby y VandeKoppel (1997, 7) muestran cómo el Departamento de Estado de la Defensa de los Estados Unidos de América, abordó problemas como *"imposibles de mantener"* y el software de *"frágil"* mediante la creación de una nueva *programación multi-paradigma idiomática*. Appleby y VandeKoppel (1997, XIV) dividen la programación en paradigmas - como muchos otros – que consisten en paradigmas imperativos y paradigmas declarativos (figura 14). Se explicarán estas denominaciones más adelante en el capítulo 4, pero por ahora basta con decir que no es una amplia taxonomía de posibles paradigmas de programación. Actualmente los arquitectos sólo tienen acceso a dos bandas estrechas de paradigmas de programación (ver distribución en la figura 14): la principal en programación textual CAD, lenguas que son predominantemente imperativas con un sesgo hacia lo procesal de programación; mientras que, los principales lenguajes de programación visual CAD residen la mayoría en un inciso de muy estrecha programación declarativa, conocida como la programación de flujo de datos. Mientras que las dos bandas de paradigmas son ocupadas por lenguajes CAD de programación y están muy investigados, son en última instancia limitados.

Para los arquitectos esto significa tener un rango limitado de medios disponibles para expresar ideas mediante programación computacional. Esto presenta una oportunidad para ampliar la práctica de modelado paramétrico computacional para la investigación de nuevos paradigmas en lenguajes de programación con participación como ingenieros de software.

1.5.3 – Flexibilidad cuantitativa

La medición de la flexibilidad en un modelo paramétrico computacional es una propuesta un poco difícil de analizar y realizar. No existe acuerdo sobre la definición de flexibilidad, ni es que exista alguna forma existente para medirla. Además, como he descrito al principio, la flexibilidad se relaciona a menudo con las circunstancias de un proyecto específico, por lo que es difícil de observar con claridad lo que está sucediendo. La flexibilidad sigue siendo en gran medida un enigma.

En esta sección esbozo un marco para la observación de la flexibilidad en los modelos paramétricos computacionales gráficos. Empiezo por proponer un método de investigación, el cual se basa en triangulación entre los diferentes casos de estudio, para mitigar algunos de los retos circunstanciales de la observación en el aspecto de la flexibilidad en el proceso de diseño arquitectónico. En los apartados siguientes se utilizarán conceptos encapsulados en las secciones [Categorías 2, 4] de pruebas en las gráficas SWEBOK de 1999. En los conceptos retomados de la ingeniería de software, se esboza un conjunto de instrumentos de investigación cuantitativos y cualitativos para medir diversos tipos de flexibilidad en un modelo paramétrico computacional. En conjunto, la investigación de instrumentos y métodos, servirán de base para la observación de la flexibilidad en los estudios de casos presentados durante los capítulos 4 y 5.

En un intento por comprender la calidad del software, los ingenieros de software han inventado un buen número de métodos cuantitativos, con el fin de evaluar varios aspectos de los códigos hechos por ellos. Existen alrededor de veintitrés métodos únicos de medición en calidad del software, los cuáles han categorizado Lincke y Welf en su *Compendio de Calidad de Software Estándar y sus Métricas* del año 2007. Además de más 100 reglas ISO/IEC que miden los estándares de la *Calidad del Producto de Software* (ISO 2000). Mientras que muchas de esas métricas son aplicables solo a circunstancias específicas[10], algunas pocas son usadas casi universalmente por los ingenieros de software. En los siguientes párrafos tomaré seis conceptos claves de medición cuantitativa y explicaré como se pueden aplicar al modelado paramétrico computacional.

Tiempo de construcción

La construcción de la medición del tiempo determina el intervalo de tiempo para la formulación, prueba y construcción de un modelo específico de software. Claramente puede haber beneficios en tiempos de construcción cortos, particularmente en modelos que son constantemente reconstruidos durante los procesos de un proyecto. Diferentes usuarios están de acuerdo y simpatizan con las compañías, en tener diferentes tiempos de construcción, y ayudan desde la familiarización con el ambiente inicial de programación y determinan que tan rápido se puede construir un modelo de software satisfactorio por las dos partes. En general, la construcción de tiempo para un modelo paramétrico computacional es frecuentemente muy extensa, en comparación con otros métodos de modelado, ya que los procesos de crear parámetros y definir funciones explícitas, requiere de manera implícita de algún conocimiento previo relacionado, el cual, es con frecuencia recuperado a través de los cambios cortos en la modificación de los tiempos.

10 Los estándares ISO/IEC tienen métricas para cada aspecto en la medición de calidad del software, desde cómo ayudar a que el sistema pueda ser usado, hasta que tiempo el usuario final puede acceder a un código de programa desde un dispositivo externo.

Modificación del tiempo

Es la modificación de los tiempos del diseño y el proyecto, que mide el tiempo que tardan las salidas del modelo desde una instancia inicial o media a otra, sea intermedia o final. Una modificación pequeña del tiempo, permite a los diseñadores hacer cambios rápidamente, la cual es una de las principales razones para usar un proceso paramétrico en la generación de modelos de diseño. Los cambios anteriores pueden involucrar la modificación de los valores en los parámetros del modelo, además de relacionar los procesos más arduos en el procesamiento de modificaciones en las funciones explícitas de los modelos. Cuando los diseñadores tratan de *"anticiparse a la flexibilidad",* normalmente hablan sobre reducir las subsecuentes modificaciones del tiempo en el arreglo del modelo que ocurre a través de las manipulaciones de los parámetros anteriores o iniciales que frecuentemente son manipulaciones o procesos lentos de funciones explícitas. Un punto importante es la que la modificación en el tiempo es altamente dependiente de la organización presente en el modelo, y particularmente cómo resulta partir de la acumulación residual de cambios durante todos los procesos. Además, que como con el tiempo de construcción, la familiarización del usuario con un modelo y el ambiente de modelado, tienen una gran etapa de "poca atención" sobre la modificación en el tiempo de ejecución de un proceso determinado.

Latencia

Es definida como el periodo de tiempo que el usuario se encuentra esperando – después de realizar un cambio - a observar un cambio en la última salida de algún parámetro, componente o representación tridimensional después del procesamiento de la información por parte de la computadora y por los procesos de cálculo necesarios para generar los resultados finales. Frecuentemente estos cálculos resultan en una latencia imperceptible, pero en modelos más complejos, la latencia puede llevar unos minutos e inclusive horas.

La latencia es importante porque los diseñadores algunas veces comenten errores al observar cambios en los modelos finales, particularmente si hay una pausa entre el cambio realizado y el cambio que se observa (ver capítulo 2 y 3; Nasirova 2011; Erhan, Woodbury y Salmasi 2009). Para apreciar que un modelo es interactivo, la investigación sugiere que la latencia debería de ser idealmente de menos de 10 segundos (Miller 1968, 271). En muchos casos es imposible cubrir todas las demandas computacionales que se relacionan con el diseño arquitectónico en cálculos geométricos; las limitaciones en el hardware de una computadora y los filtros subyacentes de los algoritmos en los ambientes del modelado paramétrico computacional.

Dimensionalidad

La dimensionalidad es un recuento de los parámetros de un modelo. O, dicho de otra manera, el número de dimensiones en el espacio de búsqueda del modelo. En el cual se busca equilibrar la dimensionalidad de un modelo, ya que, por un lado, los parámetros pueden ayudar a mejorar los tiempos de modificación (a mayor dimensionalidad, mejor) y, sin embargo, por otro lado, también muchos parámetros hacen que encontrar y modificar cualquier característica individual sea difícil de modificar (una menor dimensionalidad es mejor). Por lo tanto, un modelo paramétrico ideal sería abarcar todas las variaciones que el diseñador quiere explorar dentro de la menor dimensionalidad posible.

Tamaño

Los ingenieros del software comúnmente realizan la evaluación y medición de un programa en base a las líneas del código (en inglés LOC). Lo anterior es en cierto punto, una medición imprecisa, ya que los programas de software pueden ser re-escritos para ocupar más o menos líneas de código. Esto llevó a Steven McConnell (2006, 150) a argumentar lo siguiente "el procedimiento LOC es un método terrible para medir los tamaños y calidad del software, excepto para aquellos que poseen muchos errores". Mientras existe un grado de imprecisión, el método LOC es usado frecuentemente como instrumento para el rápido entendimiento del tamaño relativo del software. Ordinariamente, un pequeño LOC es mejor que un sistema de correlaciones LOC de gran tamaño, que involucra consecuentemente una complejidad enorme de código y errores de procesamiento; en resumen, más líneas de código provén más posibilidades para modelos o procesos finales erróneos.

En la investigación utilizaré las líneas físicas de medición de código – el número de líneas de código que se escriben actualmente en el entorno de programación -. En lenguajes de programación visual un nodo puede ser considerado, en conjunto, el equivalente a una línea de código en cualquier lenguaje. Por lo tanto, se medirá el tamaño de programas de generación algorítmica geométrica y de datos visuales por medio del número de nodos en el primer caso de estudio y por el conjunto de párrafos, familias y clases en el segundo caso de estudio. Lo anterior permitirá hacer comparaciones entre varios lenguajes de programación visual y textual; sin embargo, dadas las diferencias entre las líneas de programación textual y de nodos visuales, las comparaciones no podrán ser realizadas entre los tamaños de los códigos textuales y visuales.

Complejidad Ciclomática

La complejidad ciclomática es una métrica del núcleo del software para la medida de la estructura del código. En términos técnicos más específicos, es el número de rutas o caminos independientes a través de una Gráfica Acíclica Dirigida. (En inglés *DAG*). Se puede observar lo anterior en la figura 16 y 17. La complejidad ciclomática es calculada típicamente utilizando la fórmula matemática de Thomas McCabe (1976, 314):

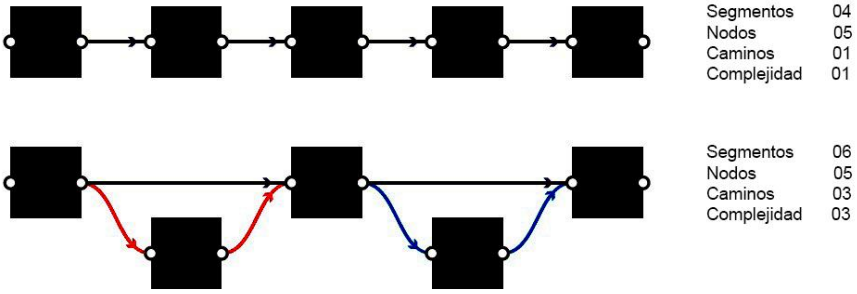
$$CC\ (G) = e - n + 2p$$

Dónde:

- *G*: la gráfica.
- *e*: número de segmentos. Se cuentan segmentos paralelos entre nodos idénticos (segmentos duplicados) como un segmento simple.
- *n*: número de nodos. No se cuentan los nodos no funcionales como comentarios en cajas de texto y representaciones de datos a través de componentes de árbol.
- *p*: número de gráficas independientes (partes).

Figura 16. Una gráfica acíclica dirigida y compuesta de una sola ruta o camino, el cual posee una complejidad ciclomática de uno.

Figura 17. Una gráfica con el mismo número de nodos, pero con tres distintos caminos (coloreados). Por consiguiente, posee una complejidad ciclomática de tres.



La fórmula de McCabe anterior asume que las gráficas *DAG* tienen solo un nodo de entrada y un nodo de salida, el cual no es común en modelos paramétricos computacionales. En una apreciación de modificaciones comunes a la fórmula de McCabe, los ingenieros Henderson-Seller y Tegarden (1994, 263) muestran que los "segmentos ficticios adicionales" pueden ser introducidos para tratar múltiples entradas y salidas. Debido a lo anterior, la complejidad ciclomática se expresa en la siguiente fórmula:

$$CC\ (G) = (e + (i\text{-}1) + (u\text{-}1)) - n + 2$$

Asumiendo que p = 1, se simplifica en:

$$CC\ (G) = e + i + u - n$$

Dónde:

- *i:* número de entradas (dimensionalidad).
- *u*: número de salidas.

La complejidad ciclomática indica que tanto trabajo se requiere para poder entender una parte del código de un programa o proceso de modelado. Por instancia, la figura 16 puede ser entendida por la lectura secuencial a lo largo de los caminos simples de los nodos. Pero el entendimiento es más complicado en la figura 17, ya que requiere de una lectura simultánea a través de tres diferentes caminos que se relacionan al inicio y al final respectivamente. Mientras puede ser posible la comprensión de la interacción de los tres caminos, frecuentemente esto puede ser más difícil por la agregación de componentes, y la complejidad del sistema se incrementa.

Como resultado de la investigación hecha por McCabe, recomienda la reestructuración de cualquier tipo de código con el método de complejidad ciclomática que con otros métodos que no menciono en esta investigación. Este límite se reafirmó por muchos estudios que incluyen el respaldo del Instituto Nacional de Estándares y Tecnología de los Estados Unidos de América, quiénes escribieron "el límite de 10 propuesto por McCabe tiene evidencia suficiente que lo soporta" (Watson y McCabe 1996, sección 2.5).

Aplicación de los indicadores cuantitativos

Los indicadores cuantitativos se prestan para los análisis estadísticos. Si una colección de ejemplos de códigos es medida cada uno cuantitativamente, los resultados pueden ser agregados juntos y ser analizados para ayudar a identificar las tendencias generales en la población mundial. Este tipo de análisis no parece tener y ser incorporado en estudios previos en el modelado paramétrico computacional. Como resultado, la comprensión actual del modelado paramétrico computacional se encuentra mayoritariamente confinado para el relato de los trabajos que se desarrollaron en primera instancia con modelos paramétricos específicos. Lo anterior deja una brecha significante en la comprensión del modelado paramétrico computacional y muchas preguntas básicas como - ¿Cuál es el tamaño promedio de un modelo paramétrico o qué tan complicado es el modelo típico? - que siguen todavía sin respuesta. En las siguientes páginas me focalizaré en responder algunas de esas preguntas básicas y establecer la base para refinar las métricas cuantitativas que he mencionado con anterioridad.

El ensamblaje de una colección representativa de modelos paramétricos es un aspecto difícil desde que los investigadores generalmente solo tienen acceso a modelos creados por ellos mismos o por sus compañeros de la misma disciplina, en esta ocasión, la arquitectónica; una razón probable es que ningún estudio ha analizado cuantitativamente un grupo de modelos paramétricos computacionales. Recientemente, el incremento de sitios web, ha permitido a comunidades de diseñadores de todo el mundo compartir sus modelos o teorías particulares en público, así, grandes colecciones de modelos paramétricos y computacionales se encuentran actualmente disponibles. En una búsqueda en el foro de *Grasshopper* (www.grasshopper3d.com), que tomó lugar entre el 08 de mayo de 2011 y el 22 de agosto de 2014, más de 575 diseñadores compartieron más de 3 041 modelos. Todos los anteriores, habían sido creados en el ambiente de modelado algorítmico gráfico Grasshopper y existía la tendencia de varios diseñadores y programadores a tener problemas con el modelo a lo hora de tratar de resolver problemas particulares. En la actualidad estas cifras se encuentran rebasadas. Mientras estas colecciones no son estrictamente representativas del modelado paramétrico general, es un avance importante que da pie a estudios analíticos abiertos, mecanismos nuevos, de manera que ahora hay un grupo de diseñadores y programadores organizados para producir modelos en conjunto, en un entorno popular de modelado paramétrico gráfico.

Método

Con el fin de analizar los modelos publicados en el foro de *Grasshopper* antes mencionado, se descargaron a través de internet alrededor de 1 500 definiciones que dan lugar a modelos paramétricos computacionales. La definición y modelo más antiguo que tengo es del año 2011 y fue creado con la versión de *Grasshopper 0.7.11*, y la más reciente es del 15 de noviembre de 2015, creada con la versión 0.9.0076. Todos los modelos y definiciones, a excepción de algunos, fueron creados y colocados en la red de internet por sus respectivos propietarios en formato .ghx (figuras 18, 19 y 20). Apliqué ingeniería inversa a estos formatos y escribí un script que extrae las relaciones paramétricas de cada archivo o definición y los analizan en una *DAG.* En los ejemplos, no se consideraron los componentes y parámetros representados a través de “clústers”, que es un método de compactación de nodos y segmentos, con valores de entrada y salida referidos a los procesos finales de una definición dada. Las gráficas de los modelos y definiciones paramétricas computacionales fueron evaluadas con cada una de las métricas cuantitativas discutidas previamente.

Figura 18. Distribución de tamaños de 2002 modelos o definiciones paramétricas.

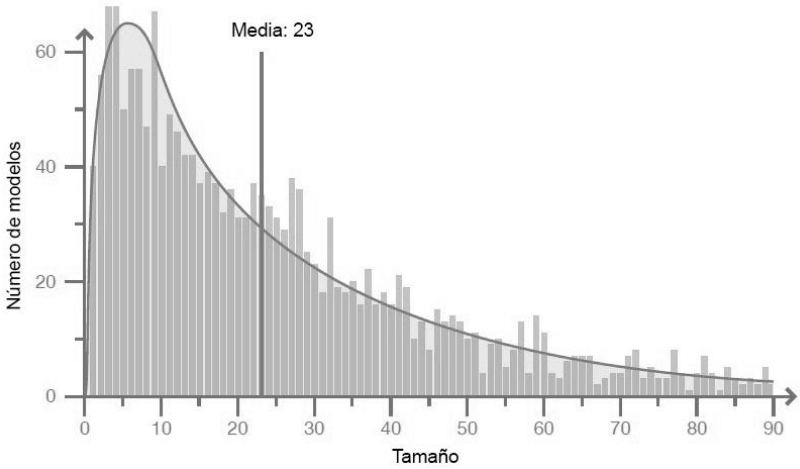


Figura 19. Distribución de 2002 tamaños de modelos o definiciones paramétricas.

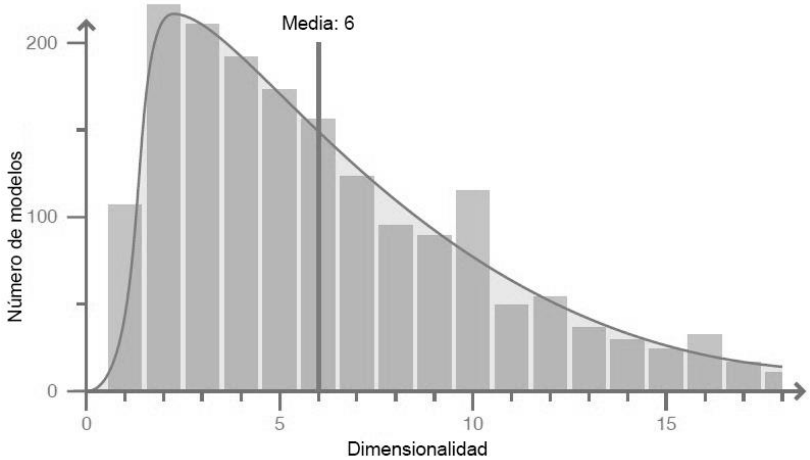
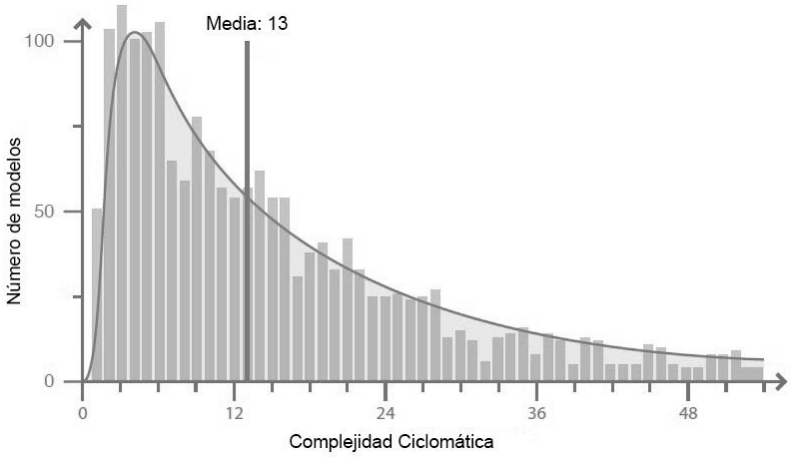


Figura 20. Distribución de 2002 complejidades ciclomáticas de modelos o definiciones paramétricas.



Tamaño

El tamaño de las definiciones y modelos de *Grasshopper* del ejemplo anterior, varían en número de orden de magnitud; la definición más pequeña contiene solo dos nodos y, por otra parte, la más grande contiene 2 007 nodos (figura 21). La distribución de tamaños de definiciones tiende a ser positiva (figura 18) con la media de las definiciones en 23 nodos. Se podría concluir que lo anterior es debido a que las definiciones de *Grasshopper* pertenecen como parte de otras definiciones de mayor tamaño. Por lo anterior, la media podría tender a ser un poco más elevada en la práctica. Incluso con una media un poco más alta las definiciones de *Grasshopper* (incluyendo aquellas que contienen más de mil nodos y no poseen clústers) son más compactas comparadas con aquellas que podemos encontrar en cualquier código de algún software en cualquier disciplina.

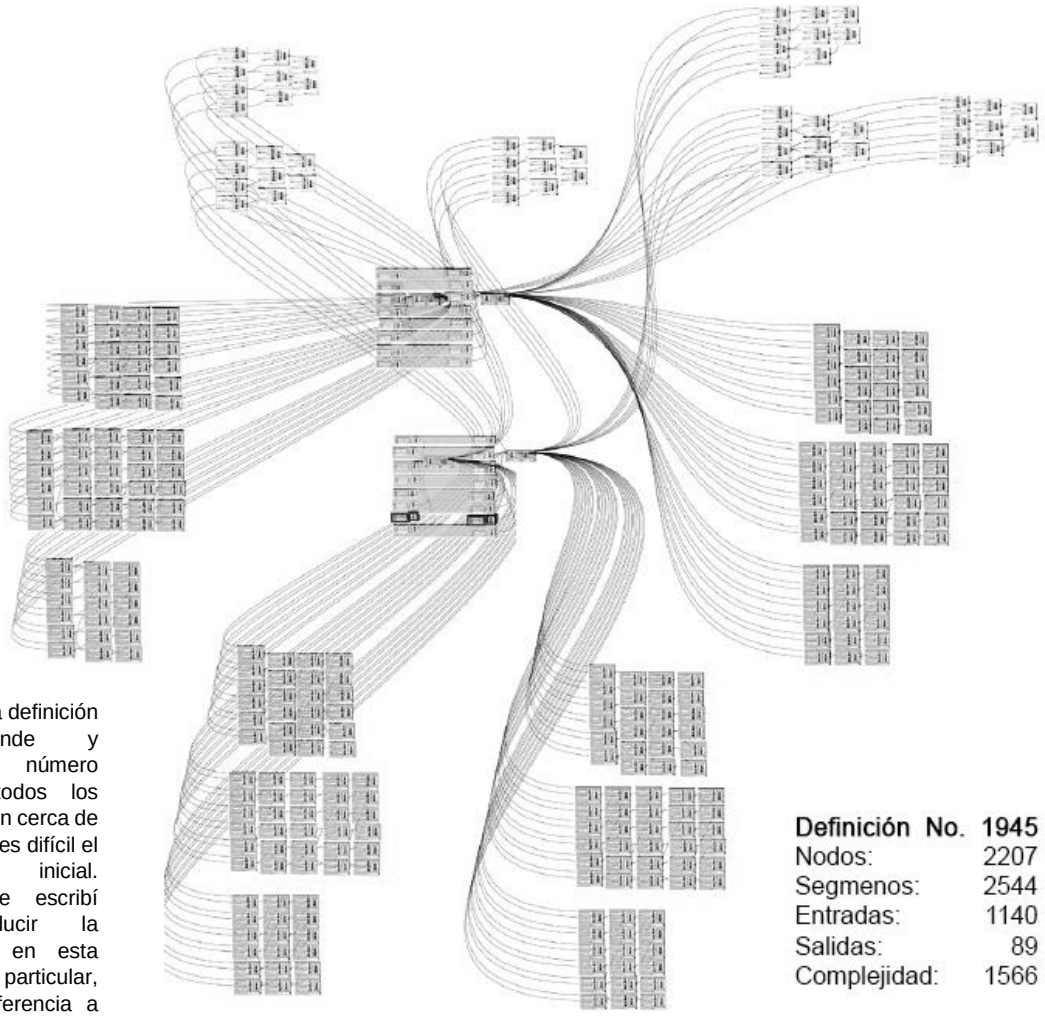


Figura 21. La definición más grande y complicada número 1945 en todos los ejemplos. Con cerca de mil entradas es difícil el análisis inicial. Anteriormente escribí cómo reducir la complejidad en esta definición particular, haciendo referencia a la duplicación y agrupación en veinte factores críticos de procesamiento, conocidos como “clústers”.

Dado el número de nodos y relaciones en la definición anterior, se pueden observar las funciones comunes en ella. El nodo más utilizado fue "*Number Slider*" el cual nos permite introducir valores numéricos. Los dos paneles más empleados fueron "*Panel*" que permite escribir y procesar datos numéricos y textuales de manera unitaria; y "*Scribble*" que permite explicar a partir de una gráfica *DAG* con la adición de texto. Los dos ítems más comunes en la definición para el control de datos fueron "*List Item*" y "*Series*". Seguidos por los componentes que nos permiten el manejo y control de datos "*Domain*", "*Flip*", entre otros. Los nodos más populares de entrada y control de geometría fueron "*Move*", seguido por *"Point XYZ"*. De hecho, solo seis de veinte de los más populares nodos se relacionan con las operaciones geométricas. Lo anterior demuestra que el modelado paramétrico computacional, al menos dentro del entorno del software Grasshopper, se relaciona más con la entrada de datos, manipulación y control de datos, y la organización de gráficas con el modelado y modificación en tiempo real de geometría.

A continuación, realizaré una comparación de definiciones de *Grasshopper* con diferentes complejidades ciclomáticas. Las seis definiciones (figuras 22 – 27) son similares en tamaño y bastante representativas de otras con complejidades equivalentes:

Figura 22.

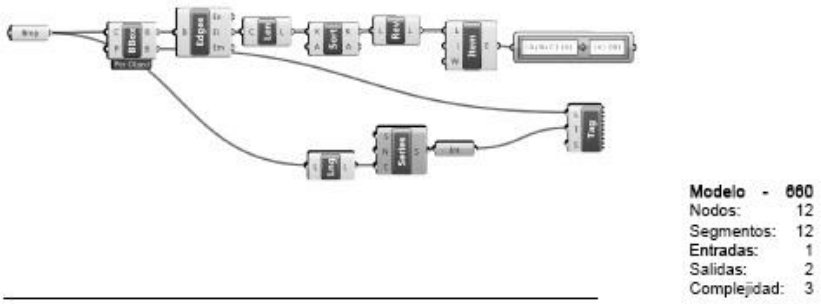


Figura 23.

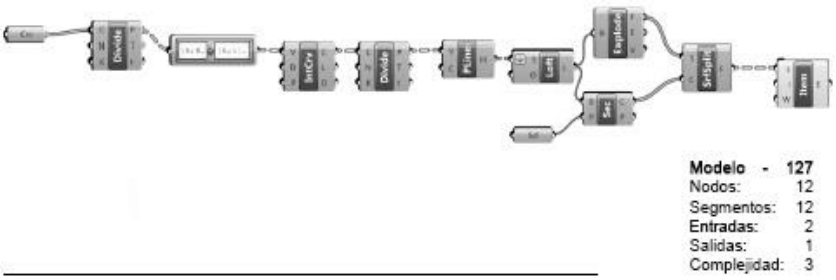
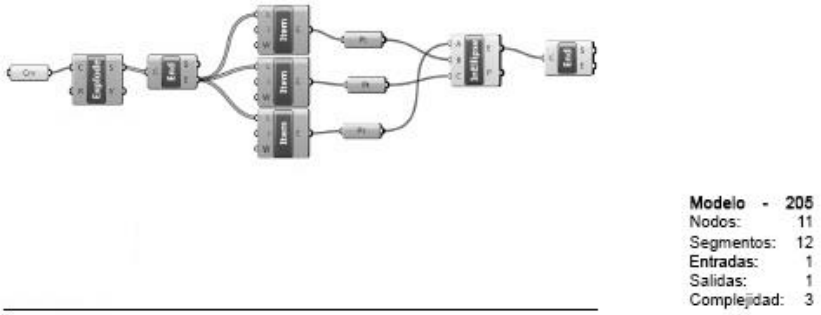


Figura 24.



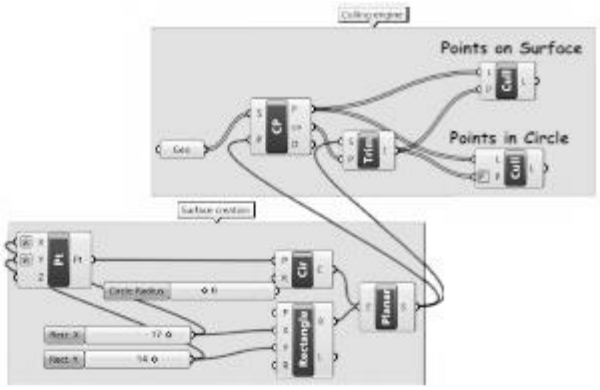


Figura 25.

Modelo - 1915
Nodos: 12
Segmentos: 16
Entradas: 4
Salidas: 2
Complejidad: 10

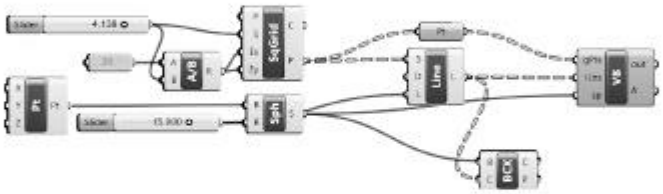


Figura 26.

Modelo - 314
Nodos: 11
Segmentos: 15
Entradas: 4
Salidas: 2
Complejidad: 10

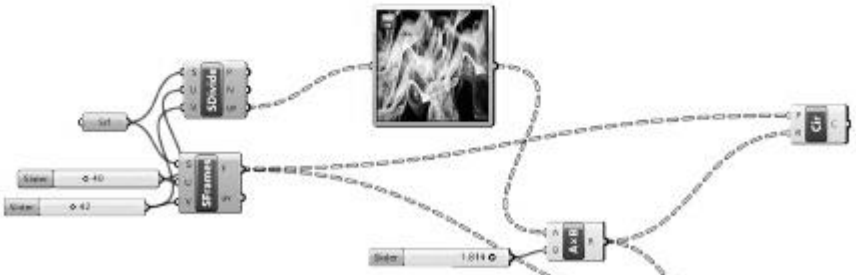


Figura 27.

Modelo - 186
Nodos: 10
Segmentos: 13
Entradas: 4
Salidas: 2
Complejidad: 9

Resultados - Dimensionalidad

La mayoría de modelos y definiciones tienen una dimensionalidad similar, 71% tienen entre una y once entradas, con un promedio de seis entradas en conjunto. Cuando se examinan las definiciones con alta dimensionalidad, es muy complicado entender lo que puede realizar cualquier entrada, y siempre más difícil, comprender lo que significan en la definición (en ocasiones, el único camino para saberlo es manipulación y error). Puedo concluir comparativamente que las definiciones de baja dimensionalidad, podrían ser consecuencia de que los arquitectos han previsto pocos parámetros de entrada al inicio. Por lo tanto, mientras los parámetros son un componente clave en el modelado paramétrico computacional, las representaciones de geometría a partir de componentes son importantes para la conclusión final en representación arquitectónica. La mayoría de los diseñadores utilizan parámetros con moderación en sus definiciones iniciales.

Resultados – Complejidad Ciclomática

Existe una variación alta en las complejidades ciclomáticas de las definiciones ejemplo. La complejidad promedio es de trece (figura 20), pero este rango se extiende desde definiciones simples con una a más altas complejidades, como 1 566 (figura 21). Dentro de esta variación, el 60% de los ejemplos analizados son más grandes que el nivel de complejidad de diez, propuesto por McCabe (1976, 314). Las diferencias entre las definiciones simples a las más complejas se observan aparentemente en las figuras 25, 26 y 27. En las figuras anteriores, las definiciones se ordenaron de acuerdo a las relaciones en cadena de comandos, mientras que las que poseen más complejidad, en relaciones entrelazadas lineales influyen de manera significativa en los nodos y en el resultado final del modelo. Esto indica que la complejidad ciclomática es efectiva y adecuada para la clasificación de niveles de dificultad en una definición paramétrica computacional.

Aprendiendo de los datos cuantitativos

Este primer estudio es un paso importante para la comprensión de las propiedades y variaciones, así como de los niveles de complejidad en modelos y definiciones paramétricas típicas, empleando como entorno de trabajo la interfaz de programación visual *Grasshopper.* Las 1 500 definiciones que he comparado (excluyendo las de 3 a 5 relaciones de nodos lineales), muestran que los modelos paramétricos pueden ser pequeños y fáciles de entender o grandes y complejos (excluyendo sistemas de “clústers”). Las definiciones promedio contienen alrededor de veintitrés nodos, con solo dos de alrededor de más de mil, con lo cual, son cifras muy modestas comparadas con los promedios que se dan en la ingeniería del software. Los tamaños de las definiciones se relacionan proporcionalmente con las complejidades de los modelos geométricos tridimensionales, que tienden a ser grandes en tamaño de procesamiento y almacenamiento de datos. Por otra parte, se puede intuir que la mayoría de las definiciones y modelos están hechos de parámetros y geometría, este estudio muestra que los componentes de organización y el control de datos numéricos y de gráficas, son los elementos más comunes en definiciones creadas en *Grasshopper*. Los parámetros tienden a ser empleados con moderación, ya que la mayoría de las definiciones estudiadas tienen entre uno y once parámetros. Otra observación importe es que la cuantificación de nodos es una gran aproximación para la medida en complejidad y dimensionalidad en las definiciones; además de que la complejidad ciclomática nos permite hacer la diferencia entre modelos simples y complejos en un modelo a definición paramétrica computacional involucrada en los procesos del diseño.

1.5.4 – Flexibilidad Cualitativa

Muchos aspectos de la flexibilidad cualitativa en los modelos arquitectónicos paramétricos eluden a una medida cuantitativa, ya que de alguna manera es útil conocer el tamaño y la complejidad de un modelo o definición paramétrica, pero, de cualquier forma, estas medidas solo nos dan una perspectiva muy incompleta. Meyer Bertrand (1997, 3) argumenta lo siguiente es su libro *Object – Oriented Software Construction*, "la calidad del software se describe de mejor manera como una combinación de diversos factores, no solo los cuantitativos" (Meyer, Capítulo 1). Explica los siguientes diez factores a tomar en consideración referidos a la "calidad del software":

01. **Corrección:** que se refiere a los comandos en todos los productos de software para llevar a cabo de manera exacta todas sus tareas predeterminadas.
02. **Robustez:** referida a la capacidad del software en reaccionar de manera apropiada y correcta a condiciones anormales, como errores en sistemas operativos.
03. **Ampliación:** se refiere a la capacidad del software o programa de adaptarse a las versiones nuevas o modificaciones sobre sus especificaciones iniciales.
04. **Reutilización:** se refiere a la capacidad de los elementos del software para servir en la construcción de nuevo software.
05. **Compatibilidad:** se refiere a la fácil combinación de elementos de software en otros de similares características.
06. **Eficiencia:** se refiere a la capacidad de adaptación de un software específico a las pocas demandas que se pueden exigir en un sistema antiguo de hardware.
07. **Portabilidad:** se refiere a la fácil transferencia de productos de software a entornos diferentes de hardware o dispositivos.
08. **Facilidad de uso:** la facilidad con que las personas de diferentes orígenes y capacidades pueden aprender a utilizar los productos de software y aplicarlos para resolver problemas generales o particulares.
09. **Funcionalidad:** se refiere al alcance de posibilidades que son dadas a un sistema a partir del software.
10. **Ser atemporal:** se refiere a la capacidad de los programadores en lanzar un software con una gran anticipación y con mejoras significativas en procesamiento y adaptación al hardware. Además del grado de innovación, reingeniería interna (compactación de código o definición) y la vigencia en los mercados futuros.

Otros autores han escrito listas similares en cuanto a la calidad el software se refiere. La lista que realizó Meyer mantiene un prestigio significativo en la industria de la ingeniería del software, ya que es citada en los libros más utilizados y reconocidos en ciencias de la computación[11]. Existen relaciones muy notables en la lista de Meyer con otros estándares, como la ISO / IEC para la Calidad del Producto de Software (ISO 2000). Con las características de la lista de Meyer, la portabilidad, facilidad de uso y funcionalidad son las categorías más parecidas a aquellas de las normas ISO antes mencionadas.

11 El programador con pseudónimo *CiteSeer* ha mencionado en el año 2012 que el libro *Software de Construcción Orientada hacia Objetos* es el sexto más citado y vendido en su base de datos de alrededor de dos millones de trabajos de computación y libros relacionados a este tema.

En este documento se toman los factores citados por Meyer con el fin de identificar la calidad del software que utilizaré para llevar a cabo los casos de estudio en los capítulos 4 y 5, en la evaluación estructurada de mediciones cualitativas en los modelos y definiciones paramétricas gráficas y textuales. En particular, haré una referencia a los conceptos que hace referencia Meyer, en cuanto a [1] corrección, [3] ampliación, [4] reutilización, [6] eficiencia, [8] facilidad de uso y [9] funcionalidad; los cuáles explicaré a continuación de manera muy breve.

Corrección

Concierne a las correcciones automáticas que el software realiza a partir de las complejidades en los códigos erróneos de procesamiento. En algunos, esta circunstancia es obvia si se crea una definición paramétrica para dibujar un cubo, el modelo solo será correcto si se procesa adecuadamente en base a componentes y parámetros acordes con la geometría de un cubo de la realidad. En la mayoría de las circunstancias, está correlación no es tan trivial ya que puede ser muy difícil de determinar, por ejemplo, iniciamos el modelado del cubo con relaciones de puntos en coordenadas *XYZ* y luego se hace una relación de puntos, siguiendo un orden "lógico" para conformar una red que representa al cubo en 2 dimensiones, pero no en 3 dimensiones; lo anterior se asegura que suceda a través de una serie de parámetros con valores de entrada iniciales. Los ingenieros de software han desarrollado una serie de métodos para determinar si esta corrección es eficaz – las pruebas unitarias pueden ser un ejemplo notable -. Aunque en la realidad, sospecho que los arquitectos y diseñadores se podrían beneficiar de la adopción completa o parcial de estas prácticas, se trata de un área de investigación que queda fuera del alcance de esta tesis. Para el resto de este documento, utilizaré un modelo y definiciones paramétricas que se encuentren libres de correcciones automáticas (con excepción de los procesamientos lógicos de la computadora para llevar a cabo las soluciones derivadas a partir de los logaritmos), y queden libres de errores, y no crear esferas, cuando deberían de ser cubos.

Ampliación

Este término es esencialmente sinónimo de flexibilidad, que se refiere a la facilidad de adaptarse a los cambios. Meyer (1997, 7) dice que la ampliación se relaciona con el tamaño, desde que "la dificultad de cambio en los programas pequeños no constituye algún problema"; en cambio, cuando el software se hace más grande, se hace más difícil de adaptar al cambio. Esta noción corresponde a lo que otros autores han escrito sobre la crisis del software (ver capítulo 1. 4) y corresponde a las relaciones que existen entre el tamaño del software y la complejidad ciclomática que empíricamente muestra Meyer. El autor continúa argumentando que las posibilidades de ampliación pueden mejorar, siempre que se garantice que el código del software tiene una "*arquitectura simple*", que puede ser lograda a partir de la estructuración del código en "*módulos autónomos*". En esta tesis exploraré esta ampliación antes descrita, y prestaré especial atención al primer caso de estudio.

Reutilización

Esto se refiere a la facilidad con que el código puede ser compartido, ya sea en parte o en su totalidad. La reutilización se ha convertido en una preocupación de los ingenieros de software, así como la reutilización de modelos y definiciones paramétricas es también una preocupación de muchos arquitectos y diseñadores.

Eficiencia

Este aspecto describe la cantidad y tiempo de carga de un programa en un lugar del hardware determinado. Es de gran importancia para los arquitectos, ya que la eficiencia de un programa ayuda a determinar la latencia que, a la vez, afecta al cambiar los tiempos de espera de los procesamientos. En casos extremos, la eficiencia de un modelo o definición puede determinar la viabilidad, ya que ciertos cálculos geométricos necesitan mucha exigencia de la parte del hardware y algunas veces resultan inviables de llevar a cabo, por tiempos o carencia del equipo necesario. Sin embargo, algunos ingenieros de software, no se preocupan de estas velocidades, ya que son un medio para determinar que los modelos son correctos y advierten que "las optimizaciones externas pueden hacer que softwares tan especializados, al final se conviertan en aptos para cambios y posterior reutilización" (Meyer 1997, 9). Debido a lo anterior, la eficiencia puede ser relevante, pero necesita de un equilibrio con otros atributos, como la corrección, reutilización y adaptación al hardware menos potente.

Facilidad de uso

Para los arquitectos, la facilidad de uso se aplica para un entorno de trabajo, para el uso y la modificación de un modelo o definición. La facilidad de uso en el entorno de modelado se refiere a la interfaz de trabajo y el flujo de trabajo; un arquitecto o diseñador que se encuentre familiarizado con un entorno de modelado específico, afectará la velocidad de construcción de modelos y definiciones gráficas y textuales, y los cambios que involucran todos los procesos de diseño arquitectónico. Si el entorno de trabajo resulta fácil de usar, de alguna manera la manipulación y control de los modelos paramétricos también lo serán. Anteriormente mencioné la importancia de la dimensionalidad y la complejidad para entender y cambiar algunos aspectos de los modelos que muestran los resultados de procesamientos finales; Meyer hace mucho énfasis en este punto, ya que "una estructura que se encuentre bien planteada y analizada, tenderá a ser más fácil de aprender y utilizar en modificaciones cruciales que una desordenada".

Funcionalidad

Meyer lo denota como "la extensión de posibilidades de procesamiento de uno o varios sistemas" (1997, 12), algo similar a la facilidad de uso es la funcionalidad, que se aplica a los dos entornos, de trabajo y de modelo. Las áreas más importantes en un ambiente de modelado incluyen los tipos de geometría permitidas, los tipos de relaciones y los métodos de expresar estas relaciones en su conjunto. Las operaciones del modelado paramétrico computacional que son más fáciles de implementar en un entorno inicial o familiar pueden ser más difíciles (o imposibles) en otro, debido a las variaciones de funcionalidad específica de los programas. De la misma manera, los cambios que son fácilmente admisibles en un entorno pueden ser un reto en otro con resultados no esperados o correctos. Por lo tanto, la funcionalidad de un entorno de trabajo para modelado paramétrico ayuda a determinar la funcionalidad de un modelo o definición paramétrica, sea esta textual o gráfica. Esta consideración frecuentemente se debe de tomar como algo importante, ya que un cambio en el entorno de modelado o trabajo de una definición a la mitad del proceso de diseño, significa normalmente comenzar de nuevo.

Aprendiendo de los datos cualitativos

Todos los datos cualitativos anteriormente descritos y explicados requieren de consideración y un juicio temprano antes de su aplicación. Algunos arquitectos han establecido protocolos de investigación; por ejemplo, existen maneras de llevar a cabo estudios con una muy buena investigación, con el fin de analizar la facilidad de uso de un software. Además de otras evaluaciones cualitativas que se pueden deducir lógicamente a través de comparaciones, como la funcionalidad de un entorno de trabajo para modelado paramétrico, mediante la comparación de sus características con las de otros entornos. Pero en parte, los anteriores conceptos, como la ampliación, recaen en la opinión y análisis de expertos en estas materias, que no son temas a tratar a profundidad en esta tesis.

Ninguna de estas mediciones es definitiva, incluso las cuantitativas se encuentran de alguna manera distorsionadas, por lo que, en cierta manera, son imposibles de medir. Sin embargo, las mediciones cualitativas proporcionan un vocabulario de atributos al iniciar con la captura de las cualidades de un modelo o definición paramétrica computacional, empleada en cada una de las partes de un proceso de diseño arquitectónico.

1.6 – Conclusiones

Las dificultades para la definición del *modelado paramétrico computacional* son muy particulares y en ocasiones, imposibles de definir. El término de *parametría* se originó en las matemáticas, dónde al menos desde la década de 1830, los matemáticos y científicos han utilizado el término en relación a diferentes representaciones geométricas. Sin embargo, como los arquitectos han adoptado el término como representación del proceso de diseño a partir del modelado computacional paramétrico, la definición se hace cada vez más complicada. En la actualidad, cuando los arquitectos hacen uso de este término, podría aplicarse a todos los procesos del diseño o solo a los diseños que cambian, o a las herramientas computacionales, o “estilo - forma”. El desacuerdo existe incluso a nivel individual, con bastantes autores que tratan de ofrecer una definición adecuada en el proceso o en el resultado de su trabajo.

El carácter que tienen en común estas definiciones, es el contenido relacionado que se focaliza en lo que pueden o se puede realizar con modelos o definiciones paramétricas computacionales. Hasta cierto punto lo anterior tiene sentido, ya que después de todo, ¿Qué hace tan interesante el empleo de los modelos paramétricos por parte de los arquitectos en la actualidad?, para los arquitectos, estos modelos y procesos supuestamente mejoran la capacidad del diseñador y extienden las posibilidades de hacer cambios, mejorando y haciendo más eficiente el proceso de diseño arquitectónico. En teoría, un arquitecto puede modificar los parámetros de un modelo resultantes de una definición y apreciar el procesamiento en tiempo real, así como resultado final del cambio efectuado. Como tales, los modelos paramétricos han llegado a ser entendidos en términos de sus resultados geométricos como métodos para producción de herramientas específicas a problemáticas de diseño particulares, o en la creación de representaciones de diseño que cambian sus relaciones iniciales y finales a través de parámetros.

Se podría deducir y hacer hincapié en que las métricas cuantitativas y cualitativas del software entran a menudo en conflicto. Cada una nos ofrece una perspectiva, y las mejoras en un aspecto pueden tener consecuencias negativas en otro. Por ejemplo, hacer un modelo paramétrico más eficiente, puede resultar que sea menos ampliable, y de la misma

manera, hacer un modelo o definición paramétrica más reutilizable puede perjudicar a la latencia. Además, estas medidas no se manifiestan necesariamente en las mejoras en la flexibilidad, ya que esta es producto en parte del azar y las circunstancias particulares de cada proceso o problemática de diseño arquitectónico. Un modelo aparentemente flexible (que lleva a cabo un procesamiento correcto, es fácil de utilizar y tiene una baja complejidad ciclomática) puede dificultarse con los procesos de diseño posteriores y propiciar errores al dar solución a algún problema geométrico o de representación tridimensional de espacios que han sido la problemática inicial del modelo paramétrico; mientras que un modelo aparentemente inflexible puede realizar la misma tarea o cambios particulares sin esfuerzo. Esta incertidumbre hace que una sola medida de flexibilidad – en el mejor de los casos – sea una estimación aproximada de resultados futuros y finales.

En este capítulo he discutido una serie de indicadores que se pueden aplicar al modelado paramétrico computacional, definidos como métricas cuantitativas para medir los tiempos de procesamiento, el tamaño o dimensionalidad, la complejidad, entre otras; así como métricas cualitativas que fomentan en el pensamiento inicial de todo modelo cualidades como la corrección automática, la funcionalidad y la reutilización de una parte del modelo o definición como unidad funcional básica de otros sistemas más complejos. Mediante la recopilación de estos indicadores a lo largo de este capítulo, he articulado un vocabulario inicial a emplear en las descripciones de modelos y definiciones paramétricas; un vocabulario que va más allá de los binarios que se refieren al éxito [1] o fracaso [0]. El uso de partes de este vocabulario que he analizado, por primera vez, a partir de un gran número de ejemplos expuestos, me ha dado la oportunidad de entender ideas que conllevan complejidades particulares, composición de datos y componentes, así como los tamaños de los modelos finales y las definiciones paramétricas que les han dado origen. Esto no solo demuestra la viabilidad en las cualidades de medición cuantitativa como complejidad ciclomática, sino que también demuestra el por qué los cuantificadores cuantitativos no son suficientes para observar y resolver ciertos casos de estudio particulares relacionados con alguna parte del proceso de diseño arquitectónico.

Puedo decir que se trata de la construcción y el mantenimiento de las relaciones explícitas dentro de un modelo y definición gráfica o textual paramétrica que distingue el modelado paramétrico computacional de otras formas de representación del diseño, no solo de la arquitectura. Como tal, me refiero a un modelo en común que muchos matemáticos relacionarían con un conjunto de ecuaciones que expresan un modelo geométrico en relación de sus funciones explícitas a partir de una serie de parámetros.

Además del conjunto de métricas, en este capítulo he identificado tres casos de estudio en relación con los aspectos de las gráficas SWEBOK y diagramas en estados de procesamiento computacional referido a complejidades ciclomáticas. Los casos de estudios a desarrollar en los siguientes capítulos no serán necesariamente representativos de desafíos que los arquitectos normalmente encuentran, sino una oportunidad para aprender de las problemáticas que involucran estos desafíos y su posible analogía con problemáticas arquitectónicas particulares.

Capítulo 2 - Programación Lógica Gráfica

2.1 - Los paradigmas en la programación

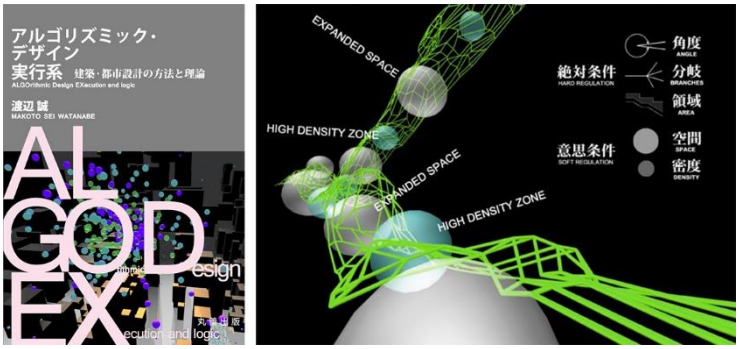


Figura 28. ALGODEX (2012) del arquitecto japonés Makoto Sei Watanabe, en dónde se analiza la relación que existe entre lo computacional, la biología y los procesos de producción de algoritmos textuales y gráficos aplicados a paradigmas en el diseño arquitectónico.

El diagrama formal de generación computacional en la parte inferior de la estación Iidabashi en Tokio, Japón diseñada por el arquitecto japonés Makoto Sei Watanabe, así como una imagen de referencia del libro ALGODEX (reedición de 2012), enuncian una serie de problemáticas y conceptos aplicados desde la perspectiva de estudios biológicos basados en el crecimiento controlado y en la alta densidad de población como origen, es una metáfora de flujos en representación tridimensional a partir de un medio de programación gráfico. El autor discute una serie de paradigmas empleando una herramienta lógica gráfica de programación, así como un intérprete formal computacional, sin embrago, se discute muy poco acerca de los diagramas desde el punto de vista de producción sistemática. Un paradigma de programación en este contexto, es una lista de principios no revelados al inicio que dan forma al estilo del idioma de programación. Estos estilos o lenguajes no son los mejores ni peores, solo diferentes a los demás. Esta diferencia es la que considero como un aspecto de diferenciación a otros lenguajes anteriormente descritos para emplear en los procesos de producción de definiciones lógicas paramétricas gráficas a desarrollar.

Los paradigmas en la programación se encuentran divididos de manera simple por Van Roy y Haridi (2004), así como por Appleby y VandeKoppel (1997) en dos tipos de paradigmas, los imperativos y declarativos (figura 15). Los lenguajes imperativos describen una serie de secuencia de acciones que realizará la computadora por medio de partes ya programadas, denominados componentes. En contraste los lenguajes declarativos definen los resultados que se quieren alcanzar, mediante el acto de introducir instrucciones definidas por un sistema lógico textual. Estos dos tipos de lenguaje pueden ser clasificados en subcategorías más pragmáticas más específicas, como se muestra en la figura 16. La mayor parte de los lenguajes de programación están basados al menos en una de esas subcategorías, y muchos se encuentran fuera y presentan una dificultad de clasificación a la hora de resolver paradigmas más complejos.

Como he mencionado anteriormente, los lenguajes preferidos y más comunes que utilizan los arquitectos y diseñadores tienden a ocupar una estrecha gama de posibles paradigmas.

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

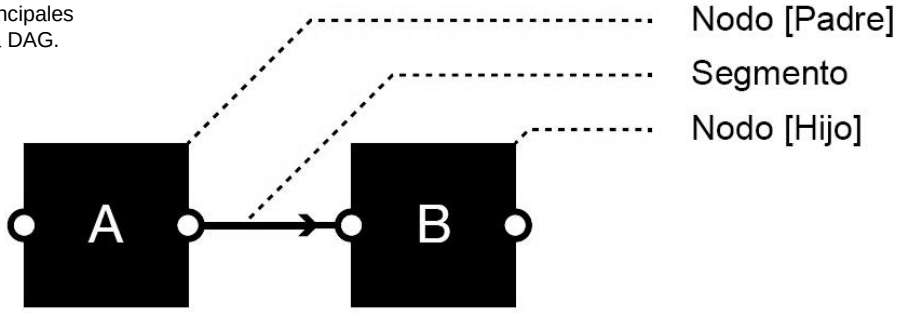
La mayoría de lenguajes de programación imperativos[12] son predominantemente CAD. Esto no es de sorprender, teniendo en cuenta que los cinco lenguajes de programación más populares en el mundo son predominantemente imperativos[13] (TIOBE 2014). En contraste, los lenguajes de programación visual tienden a ser declarativos. Lo más importante de este tipo de lenguajes de programación CAD es que se clasifican en una subsección muy estrecha en la programación declarativa conocida como programación en flujo de datos[14].

2.1.1 – Cambios en los flujos de datos

En la introducción del libro *Data Flow Computing* de John Sharp (1992, 3), el autor define un programa de flujo de datos como "uno en el que el orden de las operaciones no es determinado por el programador, pero que es derivado de las interdependencias de los datos"; en otras palabras, un lenguaje de flujo de datos describe las conexiones entre las operaciones de cálculo, que es diferente al enfoque imperativo que procesa operaciones e inclusión en el orden que deben aparecer. Cuando es ejecutado un programa de flujo de datos, la computadora infiere el orden preciso de las conexiones entre las operaciones establecidas al inicio del proceso. El ejemplo por excelencia de un programa de flujo de datos es una hoja de cálculo, el usuario especifica cómo se deberán de conectar las células y cómo las células deben procesar los datos, esto permite a la computadora decidir el orden preciso por el cual las células se actualizan. El mismo principio se aplica a cierto software de modelado paramétrico computacional, como Digital Project. Los usuarios de este software especifican una red de conexiones entre las operaciones geométricas, mientras la computadora gestiona la secuencia exacta y la ejecución de estas operaciones.

Muchos lenguajes de programación visuales operan en un principio similar al anterior, desde que las conexiones entre operaciones pueden ser representadas a través de un diagrama de flujo o estado conocido como la Gráfica Acíclica Dirigida (*DAG*). Los dos componentes principales de este tipo de gráfica son los nodos y los segmentos dirigidos (figura 29). En un programa visual el nodo es la representación de operaciones y el segmento dirigido representa una conexión (un flujo de datos entre dos operaciones), las cuáles son empleadas por arquitectos en entornos de programación visual y gráficas (Grasshopper, GenerativeComponents, Houdini, Dynamo, entre otros) y representan programas de flujo de datos.

Figura 29. Los elementos principales de una gráfica DAG.



12 Se incluyen: 3ds max, MaxScript, ArchiCAD, GDL, Digital Project, Visual Basic, Maya, Revit, Rhinoceros, SketchUp, Ruby, entre otros.
13 A año 2014 los cinco lenguajes más utilizados y populares alrededor del mundo, en referencia a TIOBE 2014 son: C, C++, Java, Python y C #.
14 En estos incluyo Grasshopper, GenerativeComponents [GC], Houdini y MaxMsp.

Un componente muy importante en la programación de flujo de datos gráfico, es que los flujos de datos tienen una dirección que es determinada por el programador o arquitecto. Al definir la dirección e inicio del flujo de datos, la fuente se denomina “padre” y el receptor de los datos se denomina “hijo” (figura 29). La dirección tiene implicaciones importantes ya que el usuario final no solo está especificando la conexión entre operaciones, sino que está dando una jerarquía a cada una de estas conexiones. Como consecuencia, si el usuario invierte el flujo de los datos – los hijos se convierten en padres y los padres en hijos – y se corre el riesgo de alterar la estructura del programa o proceso llevado a cabo por la computadora. Por ejemplo, el programa de flujo de datos en la figura 30 genera una línea entre dos puntos “padres”. Si estos datos son invertidos en este modelo (un punto se convierte en un hijo de la línea) por lo tanto, el cambio requiere de la eliminación de los nodos y la adición de unos nuevos nodos (figura 30). A pesar de que estos cambios no introducen una nueva geometría, modifican la jerarquía, tanto que empezar de nuevo sería tan fácil como cambiar un componente (figura 30) para que coincida con el modelo de la geometría nueva (figura 31). Estos cambios en la jerarquía serían anulados si el usuario no necesitara especificar las direcciones de las conexiones y en su lugar solo necesitará especificar dos operaciones que ya se encuentran conectadas. En este caso, considero que la programación lógica gráfica puede ayudar a eliminar la necesidad de especificar la dirección entre operaciones, casi de la misma forma en que la programación de flujo de datos elimina la necesidad de especificar un orden en las operaciones.

Figura 30. La relación entre una gráfica DAF y la geometría que genera. En la gráfica la línea es un hijo de dos puntos, de acuerdo con la geometría de la línea, esta depende completamente de la localización de los puntos. Al mover cualquiera de los puntos, la línea cambiará de posición y longitud.

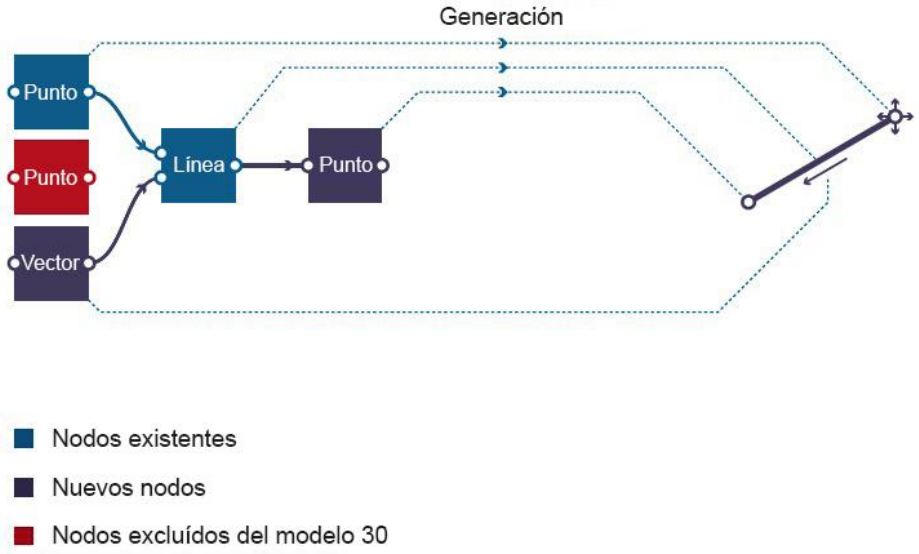


Figura 31. Las modificaciones en la gráfica DAG de la figura 30. La geometría final es la misma, pero las conexiones han cambiado: uno de los puntos es ahora un hijo de una línea. En el modelo geométrico, el punto hijo no puede ser desplazado libremente, ya que su localización depende de la posición de la línea, además de un punto y un vector, que son los padres.

2.2 – Programación lógica gráfica - desambiguación

La programación lógica gráfica, como la programación de flujo de datos, se inscribe en la rama declarativa en paradigmas de programación (figura 15). Definido por Sterling y Shapiro (1994, 9) en la primera parte como "un programa de lógica es el conjunto de axiomas o reglas, que definen las relaciones entre los objetos. Estas relaciones no especifican un flujo de datos, sino más bien expresan, en la lógica de primer orden, declaraciones de hechos". Por ejemplo, un axioma podría ser: **un perro es un animal**. La segunda parte de un programa de lógica es un intérprete que lee los axiomas y los deduce lógicamente como consecuencias[15]. Utilizando el axioma anterior, el intérprete podría preguntar: **¿Qué es un animal?**, con lo que podría deducir: **un perro**.

En los inicios de la programación lógica, a finales de los años 1960 y principios de 1970, muchos programadores formalizaron el término de programación lógica enfocada al razonamiento humano con el fin de empujar e incitar a la humanidad más allá de sus limitaciones cognitivas (limitaciones que aparentemente surgieron y fueron provocadas por los eventos en la crisis del software). Robert Kowalski en 1998, recuerda a sus contemporáneos en esta lógica que emplea en los periodos de programación para ambiciosos proyectos y haciendo una implicación en el "lenguaje natural de preguntas de respuesta" (p. 38) y "una demostración automática de teoremas" (p. 39). Los avances iniciales fueron prometedores con los investigadores que descubrieron varias maneras de obtener computadoras para imitar a la razón y responder a una serie de preguntas, incluso si las preguntas eran limitadas a la dominación de problemas menores. El objetivo era si un equipo de computadoras era capaz de responder a esta simple pregunta: ¿Cuántas pirámides están en la caja de color azul?, y por consiguiente es posible conseguir hacer responder a las computadoras una pregunta más complicada, como: ¿qué significa "cuántas pirámides están en la caja de color azul" como significante? pero como lo he discutido en el anterior capítulo, los sistemas relacionados con los problemas pequeños – ya que los programas computacionales o modelos paramétricos – no siempre abordarán las problemáticas más grandes. El interés inicial en la programación lógica gráfica se incorporó con éxito en problemas pequeños, bien definidos, pero no se pudo conseguir un éxito igual en problemáticas más grandes y complejas.

Al día de hoy, el lenguaje de programación lógica más común ocupa solo el lugar trigésimo sexto en la clasificación de TIOBE al año 2014, con base en el índice de lenguajes de programación populares. Sin embargo, la programación lógica todavía encuentra aplicaciones en el razonamiento de ingenieros expertos, acerca de las relaciones con otros softwares; por ejemplo, la computadora ganadora de IBM Jeopardy, estuvo programada en Prolog, un lenguaje de programación lógico.

15 De forma superficial, puede parecer que muchos ejemplos en programación lógica son empleados en software CAD. Por ejemplo, los dibujos de Sutherland en 1963, se llevan a cabo por la solución de restricciones geométricas que permite al usuario final definir axiomas entre la geometría. Sin embrago, no lo hace en los dibujos o resultados finales, el éxito en la definición de programación lógica es que los axiomas son interpretados usando algoritmos numéricos y un controlador que escala de forma temprana la forma inicial, en lugar de una lógica de deducción. La deducción lógica que establece las bases para la programación lógica, no se inventó hasta años después.

En el sector de la arquitectura, la programación lógica ha seguido un camino parecido, aunque años atrás de lo que sucedía en la ingeniería del software. Investigadores en el ámbito arquitectónico tomaron el trabajo realizado en programación lógica espacial y la aplicaron de manera entusiasta a un problema popular de los tiempos: el diseño de una habitación (Keller 2006).

Ninguno de los trabajos a los que hago referencia (o que he encontrado), discuten el uso de la programación lógica en proyectos de arquitectura reales – a pesar de que muchos de ellos - como el de Peter Swinson en 1982, que estaba haciendo un juicio en confianza absoluta en que la programación lógica sería la "herramienta para el arquitecto del futuro".

Al igual que los ingenieros de software que llegaron antes que ellos, estos investigadores en el ámbito de la arquitectura han probado la programación lógica con problemas particulares en cada caso, se podría suponer que los éxitos de estos pequeños sistemas eran representativos para posteriores escalas más grandes y complejas. El interés inicial en la programación lógica disminuyó (como sucedió con la ingeniería de software unos años antes), debido al poco éxito obtenido y los pequeños y bien definidos problemas, no alcanzaron el éxito en la generalización de problemas de mayor escala. Sin embargo, todavía hay ejemplos recientes en el empleo de la programación lógica por algunos investigadores (aún en pequeños problemas idealizados), incluyendo la herramienta PolyFormes de Dominique Martin y Philippe Martin en el año de 1999 y MultiCAD[16] de Makris del año 2006. Sin embargo, en contraste con el paradigma de programación imperativa generalizada y empleada por arquitectos, la programación lógica nunca se convirtió en la herramienta informática del futuro. De hecho, aún no he podido encontrar algún ejemplo de programación lógica que se utilice en un proyecto de arquitectura real.

16 Para la historia completa ver el libro *"Constraint Solving for Computer Aided Design"* de Fudos, Ioannis.

2.2.1 - Programación lógica gráfica y relaciones paramétricas

Como parte de mi investigación desarrollaré un modelo o definición simple basada en un lenguaje de programación lógica visual. Un arquitecto usando este lenguaje podría definir conexiones entre las operaciones resultantes de los componentes, para luego ser empleadas como intérpretes que deriven en un flujo de datos que sin que el usuario especifique una cierta jerarquía inicial (como se podría realizar en un lenguaje de flujo de datos). He elegido desarrollar mi propio intérprete de programación lógica basado en una gráfica DAG para emplearla como implementación de una programación lógica existente. Esto me permitirá enlazar al intérprete directamente con el motor paramétrico del software – *plugin Grasshopper* que se incorpora la versión 5.0 SR 6 de *Rhinoceros*. Este ejemplo lo desarrollaré de manera práctica en el primer caso de estudio.

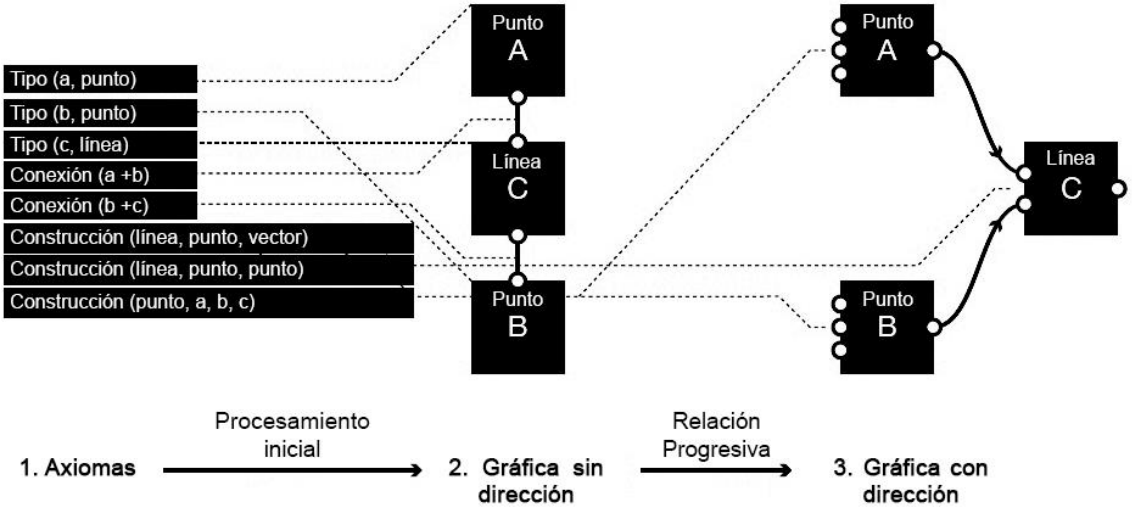


Figura 32. Las principales etapas que implican la deducción de un modelo paramétrico para satisfacer un conjunto de axiomas textuales. Las líneas discontinuas muestran cómo los axiomas generan partes particulares de los gráficos en las diversas etapas.

Este intérprete "personal" utiliza un proceso inicial de tres etapas que derivará automáticamente en un modelo paramétrico a partir de una serie de axiomas a desarrollar posteriormente. Estas etapas se encuentran ilustradas en la anterior imagen (figura 32). La primera etapa se refiere a la lectura e interpretación de axiomas referidos a la ciencia biológica en sistemas de crecimiento y comportamientos en ramificaciones de organismos marinos específicos a desarrollar y su incorporación como proceso de producción en la disciplina arquitectónica en un caso de estudio particular. En la segunda etapa, los axiomas son analizados dentro de un sistema de relaciones, procesamiento de datos y parámetros numéricos (empleando reglas que explicaré brevemente). En la etapa final, el intérprete gráfico utilizará los procesamientos finales computaciones en base a la definición lógica, para producir la gráfica *DAG* y que representará a los axiomas iniciales en un producto tridimensional como representación de dicho proceso. La definición de los axiomas base a este lenguaje se desarrollarán en el primer caso de estudio.

2.3 – El diagrama algorítmico - gráfico en el proceso de diseño

Para explicar de manera práctica la utilidad y el proceso para emplear axiomas en el proceso inicial del diseño, trataré de ejemplificar estos pasos mencionados anteriormente con la incógnita de construcción de una línea, basado en axiomas de inicio (figura 32).

Se podrían permitir tres tipos de axiomas en el lenguaje anteriormente descrito:

Tipo de Axioma 1: Geométrico

Un axioma geométrico define una entidad única geométrica y la geometría tipo. Por ejemplo, una línea con el nombre *"C"* es definida por el axioma:

tipo (C, línea)

Tipo de Axioma 2: Conexión

Un axioma de conexión establece una conexión y relación de datos entre dos entidades geométricas. Por ejemplo, cuando se conecta un punto *"C"* con el punto *"B"*, el axioma podría ser:

conectar (B, C)

Este tipo de axiomas no definen la dirección de la conexión, solo establecen un estado de relación entre dos entidades geométricas definidas.

Tipo de Axioma 3: Construcción

Un axioma de construcción describe una combinación de pares de datos que definen un tipo de geometría particular. Por ejemplo, una línea puede ser definida por un par de puntos, los cuáles se expresan con el siguiente axioma:

construir (línea, punto, punto)

Cualquier tipo de geometría particular puede tener múltiples axiomas de construcción. Otro ejemplo puede ser una línea que está definida por un punto y un vector con el axioma:

construir (línea, punto, vector)

Utilizando las relaciones anteriores, el intérprete selecciona la construcción más apropiada para una situación específica. En la figura 32, el algoritmo relacionado ha deducido que el nodo *C* (línea), debe ser definido por dos pares de puntos. Para satisfacer esta relación, el intérprete organiza la gráfica sin dirección en jerarquías, dónde la línea *C* se convierte en el resultado o *niño* de los puntos (*A & B*). En la siguiente página explicaré las tres reglas utilizadas para deducir los axiomas de construcción más apropiados.

Inferencia A: Construcción asimétrica

La lista de axiomas de construcción del tipo de geometría que pueden procesarse como los *padres* para cualquier lista de nodos tipo. Si dos nodos se encuentran conectados, y el primer nodo tiene un axioma de construcción permitiendo al segundo nodo establecer una relación directa, y el segundo nodo no tiene axiomas de construcción que permitan al primero ser su *compañero*, entonces el primer nodo necesariamente tiene que ser el *compañero* del segundo nodo.

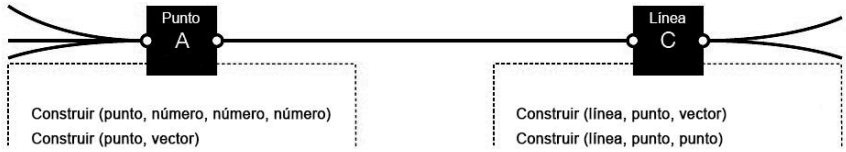


Figura 33. *A* debe de ser el padre de *C*, ya que ningún axioma de construcción para *A* requiere una línea como padre, mientras que los dos axiomas de *C* requieren un punto relacionado como padre.

Inferencia B: Eliminación del constructor

Si un nodo tiene múltiples axiomas de construcción, algunos axiomas deben de ser eliminados si el nodo no está conectado con la combinación derecha de tipos de nodo para cumplir un axioma de construcción particular. La eliminación del axioma de construcción lo convertirá en algo parecido a un constructor asimétrico.

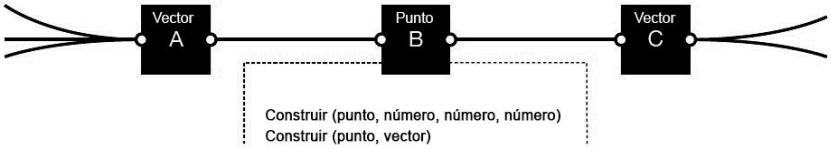


Figura 34. El primer axioma de construcción requiere a *B* para tener tres padres que son números. Mientras *B* no esté conectado con números, no será capaz de cumplir con el axioma y será eliminado. El segundo axioma todavía es posible, lo que significa que uno de los vectores deberá ser padre de *B*.

Inferencia C: Definición del constructor

Si un nodo tiene todas las relaciones requeridas para cumplir la función de un axioma de construcción, entonces no necesitará más relaciones y todos los nodos restantes se conectarán a través de conexiones sin dirección y serán los resultados o *hijos*. Una vez que esta regla vuelve relevante a la direccionalidad, normalmente las listas de datos a través de la gráfica, se convertirán en grandes cantidades numéricas y se vuelven gráficas con dirección.

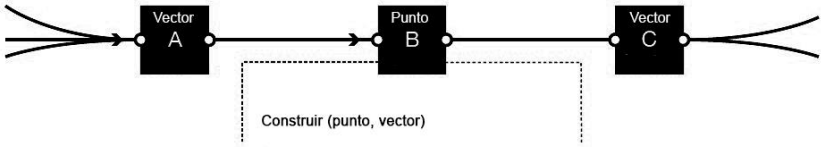


Figura 35. Si *A* es padre de *B*, entonces el axioma de construcción se cumple y no puede haber más padres. De acuerdo a esto, todas las conexiones restantes sin dirección deberán fluir lejos de *B*, y *C* se convertiría en hijo de *B*.

Idealmente estas tres reglas son suficientes para deducir los flujos completos de la gráfica *DAG* para la construcción y representación de una línea. Una vez que los axiomas se conecten y estén definidos en una gráfica con dirección, se producirá una situación dónde los nodos estarán conectados sin posibilidad de ser *padres* de los otros. En este caso, en primera instancia, el intérprete examinará para ver si la adición de nodos numéricos permitirá un progreso para avanzar a una salida de datos. Si lo anterior no sucede, el usuario tendrá que resolver inicialmente los conflictos entre axiomas.

Un arquitecto empleando este sistema de lenguaje de programación solo necesitará definir conexiones entre las entidades geométricas, las cuáles permitirán al intérprete deducir las direcciones de las conexiones y relaciones de datos. La figura 36 muestra como los axiomas se transforman en un modelo paramétrico y cómo los cambios en los axiomas resultan automáticamente en nuevos flujos de datos. Este ejemplo es idéntico al de la figura 32 excepto, en el flujo de datos, que gestionan los cambios en las jerarquías en el modelo paramétrico que son controladas automáticamente por el intérprete; el usuario puede añadir un poco de axiomas y nuevos flujos de datos que serán procesados automáticamente por el intérprete. La siguiente ilustración describe cómo se lleva a cabo lo anteriormente descrito:

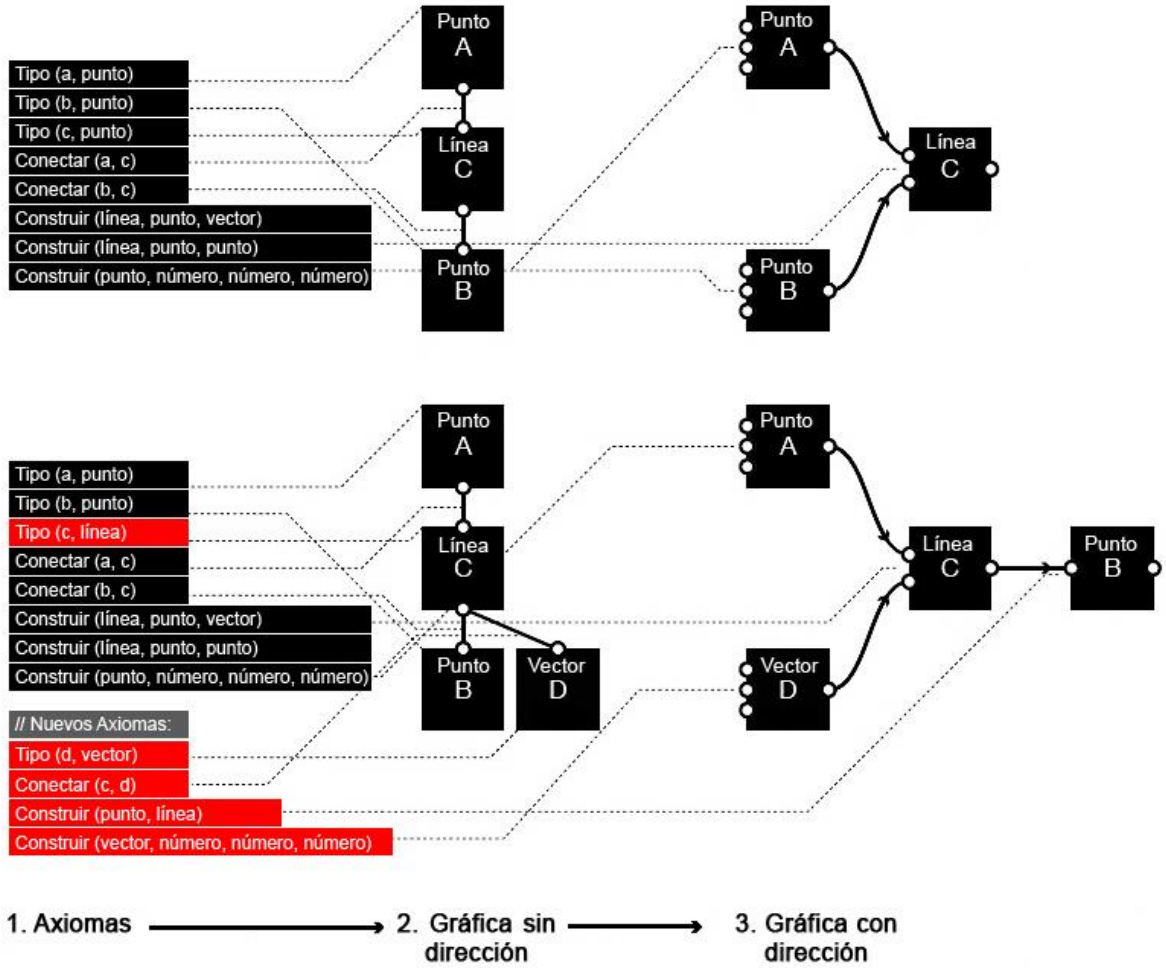


Figura 36. Un ejemplo de cómo el cambio en la lista de axiomas se traslada a variaciones dentro del modelo paramétrico. Arriba: los axiomas y el modelo paramétrico resultante de la figura 32. Abajo: la inclusión de nuevos axiomas crea una diferencia significativa en la gráfica sin dirección, que interpreta estas relaciones en transformaciones dentro de un modelo paramétrico radicalmente diferente cuando lo comparamos en el modelo de arriba.

2.4 – Conclusiones

Van Roy y Haridi (2004, 25), escriben que, al momento de programar los paradigmas, “no es mucho mejor (o peor), solo es diferente”. En la actualidad, todavía la aplicación de modelos paramétricos por parte de los arquitectos, como en el ejemplo de producir una línea en un “X” diseño, han tenido dificultades para emplearlos en paradigmas de programación. Los arquitectos que producen modelos paramétricos computacionales, se encuentran actualmente confinados a cualquiera de los paradigmas de flujos de datos declarativos en lenguajes de programación visual, o paradigmas imperativos procesales en los lenguajes de programación textual. La pequeña investigación que presenté referida a la construcción de una línea, demuestra que la influencia de los paradigmas de programación – o por lo menos – la construcción del modelo paramétrico con el fin de representar una línea en un entorno de modelado, a partir de axiomas que definen su posición, dirección, datos, números y relaciones que son procesadas por la computadora. Desde que los paradigmas de programación no pueden cambiar normalmente por sí solos, sino a partir de la reconstrucción de un modelo, la selección de un paradigma de programación inicial apropiado para el proyecto, es una decisión importante. Por lo tanto, es una decisión que tiene un efecto evidente en muchos aspectos de la flexibilidad en un modelo paramétrico, sin embargo, es una decisión muy complicada de conseguir para los arquitectos que a menudo no pueden elegir algo mayor, menor o diferente en complejidad, referido al flujo de datos o procesamiento. Las discusiones sobre lenguajes de programación son muy interesantes, no solo por el significado metafórico que los paradigmas de programación, sino por las discusiones en torno a lo que la programación significa como modelo paramétrico y que puede privilegiar los diferentes tipos de cambio en producción del diseño.

Quizá la parte más intrigante del ejemplo anterior, sea la disyunción entre las ventajas teóricas de la programación lógica gráfica y las ventajas de la programación en conjunto. Comencé mencionando los retos de la modificación de las jerarquías en los lenguajes de flujos de datos y postulé que el tiempo asociado con estas modificaciones teóricamente se podría reducir si la computadora – en lugar del usuario – organizaba las jerarquías de manera automática en el modelo. En una serie de diagramas ilustré cómo un lenguaje de programación lógico gráfico permitiría a un arquitecto especificar conexiones sin dirección que serían organizadas automáticamente por la lógica del intérprete de programación, reduciendo de este modo, el tiempo del cambio. Sin embargo, en realidad ocurrió algo cercano, pero no lo esperado, ya que cuando se utiliza un modelo paramétrico para generar la geometría final, a partir de todas las operaciones, en realidad la programación lógica alarga el tiempo de la modificación. Esto es una parte importante, ya que el intérprete frecuentemente toma un lapso de tiempo para generar nuevas instancias en el modelo; pero puede tener una importancia significativa al momento en que un arquitecto tiene que esperar un largo periodo de tiempo, con lo que el intérprete en la programación lógica trabajará en ese lapso de tiempo “perdido”. Aunque estas circunstancias son altamente dependientes del resultado geométrico final, la precisión es algo relevante, ya que los resultados finales subrayan el procesamiento correcto, basado en la práctica del modelado paramétrico. Esta disyunción entre la teoría y la práctica, cuando se habla de paradigmas de programación, es quizá la razón por la que la programación lógica no cumple las expectativas teóricas formuladas por los autores citados en este capítulo.

Lo anterior no quiere decir que la programación lógica gráfica carece de fundamento, sino que pierde una ventaja relativa a algunas configuraciones geométricas, ya que en la arquitectura como en la ingeniería del software, el aspecto sobresaliente es el razonamiento en las relaciones. Esta cualidad, parece adecuada para la extracción de las relaciones geométricas preexistentes con el fin de derivar en un modelo paramétrico; lo contrario al proceso de modelado paramétrico típico. En el ejemplo explicado, con un modelo paramétrico no existente, se conduce a la reducción de tiempos de producción inicial de cualquier modelo espacial cartesiano computacional en comparación con el proceso de modelado en flujo de datos. Sin embargo, ambos métodos producen grandes y complejos resultados de datos y geometrías que son difíciles de revisar y comprender al final de las operaciones computacionales.

Capítulo 3 – Programación Lógica Textual

3.1 – Ver, programar, diseñar

Antes de comenzar a escribir el código de un programa, es importante distinguir las diferencias que conllevan escribir un programa en un lenguaje de computadora humano y escribir un ensayo o un correo electrónico. Escribir en un lenguaje humano, permite al autor utilizar la ambigüedad de las palabras y su significado, que conlleva directamente a la construcción de frases simples o complejas. Esas técnicas permiten múltiples interpretaciones de un texto simple y dan a cada autor una voz única. Cada programa de computadora también revela el *"estilo"* del autor, pero en esta parte hay menos espacio para la ambigüedad. Mientras el lector o usuario puede tener un bagaje de interpretaciones que pueden disgregarse por la gramática, en los lenguajes computacionales lo anterior existe a través de un orden de las partes del código, número de elementos mínimos y máximo, entre otros. Si esos detalles se mantienen siempre presentes conforme se lee el código, llegarán a ser tan habituales al programar. Las variaciones sistemáticas de los lenguajes de programación textual, son básicamente importantes de conocer.

Los conceptos iniciales relacionados con los lenguajes de programación textual son la forma visual, el movimiento y la interacción. Los tres elementos anteriores integran un lenguaje de programación básico, en un entorno desarrollado y una metodología de enseñanza dentro de un sistema unificado de "gramática". Algunos lenguajes de programación basados en escribir texto fueron creados para enseñar los fundamentos de la programación computacional en un entorno visual, para que el software sirva como cuaderno de bocetos y pueda ser utilizado como herramienta de producción. Estudiantes, artistas, diseñadores, arquitectos, investigadores, entre otros, lo usan para aprender, para prototipos en todas las fases de desarrollo y producciones finales.

Los sistemas de programación textual en un entorno visual computacional facilitan el aprendizaje y la enseñanza de gráficas computacionales y técnicas de interacción, incluyendo el dibujo vectorial, el procesamiento de imágenes, modelos de color, movimiento del mouse, recepción de instrucciones mediante la escritura de comandos desde el teclado, comunicación a partir de redes, y programación orientada a objetos. Además de permitir la incorporación de librerías externas desarrolladas en el mismo u otros lenguajes compatibles con el fin de llevar a cabo tareas más complejas de manera más eficiente y con un rápido procesamiento.

Los conceptos iniciales del software como medio de procesamiento de datos, variables, emociones humanas, comportamientos humanos, entre otros, se encuentran relacionados con los siguientes aspectos del entorno particular de cada software:

El software es un medio único con cualidades únicas
Los conceptos y emociones que no pueden ser expresados por medios especiales, pueden ser expresados en este medio. El software requiere su propia terminología y discurso, y no deberá de ser evaluado en relaciones primarias con el cine, la fotografía, la danza, la pintura, entre otros. La historia muestra que las tecnologías relacionadas con los lienzos, las cámaras, y la animación han cambiado en práctica y discurso artístico, y mientras no tratemos de involucrar y relacionar cada vez las tecnologías computacionales y artísticas, no podremos y no seremos capaces de sentir nuevas formas de comunicación y expresión humanas.

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

El software mantiene varias posturas en relación a los medios artísticos que se enfocan en la habilidad de producir formas dinámicas, procesamiento de acciones, comportamientos definidos, simulación de sistemas naturales, e incluye otros medios, como la inclusión del sonido, el movimiento, la imagen y el texto.

Cada lenguaje de programación es diferente a otro
Así como cualquier medio, la diferenciación es apropiada dados los tipos de tareas. Cuando una silla es idealizada, el diseñador tendrá la posibilidad de elegir el material como acero, madera, materiales nuevos basados en polímeros, desechos o los requeridos por el usuario final. Este concepto se puede traducir como escribir el código de un software. El animador abstracto Larry Cuba describe su experiencia de esta manera: “cada una de mis animaciones ha sido realizada utilizando diferentes sistemas de lenguajes de programación. Un lenguaje de programación te puede dar la capacidad de expresar algunas ideas, más que limitarte a expresarlas por otros medios”. Existen muchos lenguajes de programación disponibles para cualquier elección de problema, y algunos son más apropiados que otros, dependiendo de los objetivos.

Dibujar es necesario para el desarrollo de ideas
Es necesario dibujar inicialmente en relación al medio de desarrollar y producir ideas enfocadas a la arquitectura y al diseño, ya que el dibujo inicial se puede aproximar al resultado final. Los pintores pueden realizar dibujos complejos y bocetos antes de aproximarse al objetivo. Los arquitectos tradicionalmente han trabajado inicialmente con una hoja de papel y un lápiz, además de modelos tridimensionales de materiales para aproximarse a un entendimiento de las formas en el espacio cartesiano tridimensional. Los músicos frecuentemente trabajan con el piano antes de escribir alguna partitura. Para realizar bocetos iniciales en los medios electrónicos, es importante trabajar con herramientas y medios electrónicos; así como algunos lenguajes de programación son mejores en diferentes tareas, algunos son mejores al usarlos para dibujar que otros, y los artistas que encuentren necesidad de trabajar con software en entornos digitales específicos, trabajarán aquellas ideas antes de escribir un código computacional final.

La programación no es solo para ingenieros
Muchas personas creen y piensan que la programación es solo para personas muy buenas en matemáticas u otras disciplinas técnicas. Una razón que se encuentra presente en este tipo de personalidad es que muchas personas enfocadas en las disciplinas técnicas usualmente crean lenguajes de programación. Es posible producir diferentes tipos de lenguajes de programación y entornos que permitan a las personas no relacionadas con estas disciplinas ser partícipes para el desarrollo de conceptos espaciales y visuales. Existen lenguajes de programación alternativos que pueden ser fáciles de aprender y enseñar a personas no relacionadas con estos entornos de producción textual. Un lenguaje de programación alternativo en los inicios fue LOGO, diseñado en la década de los años de 1960 por Seymour Papert como un lenguaje conceptual para niños. Este software permitía programar en nivel básico a los niños para diferentes representaciones, como una tortuga robótica e imágenes gráficas en la pantalla. Un ejemplo más contemporáneo es el entorno de programación MAX desarrollado por Miller Puckette en 1980. MAX es muy diferente a lenguajes de programación típicos, la programación es creada a partir de la conexión de cajas que representan un código del programa, relacionadas con líneas y texto. Esto ha generado atención en miles de artistas que lo utilizan como herramienta de producción.

La flexibilidad del código

En la ceremonia del premio Turing para el ingeniero del software ganador en la competencia *The Design of Design* de 2010, Frederick Brooks, cita de manera frecuente a Schön y Cross en relación a las posiciones de la programación en las disciplinas del diseño. Brooks argumenta que la programación, así como las formas del diseño, no son un simple salto a una solución premeditada, sino más bien, un viaje complicado que requiere de incertidumbre y que se caracteriza por la iteración. Durante este proceso, los ingenieros de software representan sus ideas con código informático, que contiene principalmente las relaciones expresadas a través de una sintaxis lógica. A medida que estas redes se desarrollan durante el proceso de diseño, pueden llegar a ser frágiles y se rompen de manera inesperada. Estos momentos de inflexibilidad son momentos importantes de algunos modelos paramétricos complejos en dificultad y procesamiento. Durante un periodo en la década de los años de 1960, los científicos que temían que estas roturas frágiles fueran insuperables y el límite de la computación no serían la velocidad de las computadoras, sino el conocimiento de los programadores para crear y mantener el software (Naur y Randell 1968, cap. 7.1).

Los ingenieros de software todavía luchan contra la inflexibilidad. Si bien es difícil obtener datos precisos, algunas encuestas de la industria han sugerido que entre el 14% y el 42% de los proyectos de software han tenido éxito en el año 2012 (The Standish Group 2012; The Standish Group 2009; Eveleens y Verhoef 2010). En una disciplina donde los costos de los materiales y de fabricación son nulos, los autores de las encuestas sostienen que los principales determinantes del éxito de un programador se convierten en los "retos esenciales de la complejidad y el costo del diseño" (Faulk 2010, 439). En consecuencia, la ingeniería del software es clasificada a partir del conocimiento esperado de un programador – con estrategias de gestión, algoritmos, estructuras de datos, problemas de tácticas de descomposición, paradigmas de programación e interfaces de codificación – con el fin de reducir tanto la complejidad como el costo de diseño. Este es al acumulativo de conocimiento que los programadores debaten con el fin de luchar contra la inflexibilidad.

Por el contrario, los arquitectos, como menciona Mark Gage, vicedecano de la escuela de arquitectura de Yale, en Estados Unidos de América, en el año 2011, respecto a "la utilización de la computación en la escuela, que ya se deje de hablar de ello". El vicedecano se burla de los arquitectos que hablan de ello al justificar proyectos computacionales provenientes de los "códigos secretos en libros perdidos parecidos a la biblia encontrados por un sujeto extraño" (2011, 1) y lo anterior se derivó de "un semestre de producción en la más intrincada red paramétrica jamás vista por el hombre" (2011, 1). Mientras que es evidente que existe un elemento de verdad a la caricatura polémica de Gage, que no respalda necesariamente su conclusión de que los arquitectos deben de dejar de hablar de computación. Robert Woodbury ha señalado por otro lado que "los arquitectos modeladores paramétricos tienen una causa en común con los profesionales de la programación" pero "hay poca conexión explícita entre ellos" (2010, 66). Desde este punto de vista, el problema no es relacionado a lo que los arquitectos hablan respecto a la computación, sino que los arquitectos suelen hablar sobre el cálculo en términos bastante extraños en comparación con los intercambios verbales que se dan entre los ingenieros de software. Los cambios de posición son aspectos esenciales, deseables e inevitables, tanto en el diseño del software como en la arquitectura, una cualidad que se relaciona rígidamente y abiertamente en la práctica.

Programación estructurada

En marzo de 1968, Edsger Dijkstra escribió una carta a la Asociación de Computación y Maquinaria, titulada "*Go to Statement Considered Harmful"*. En esos años, esta declaración fue el primer mecanismo de control para una secuencia de ejecución en un programa de computadora (el estatuto GOTO permitía a la ejecución de un programa, saltar hacia adelante o hacia atrás, a partir de una lista de comandos de control). El autor argumentaba que el salto a este tipo de secuencias de control era un avance importante en el trabajo de los programadores de la época "era una invitación a hacer el trabajo más complicado". Basado en el trabajo de Böhm y Jacopini de 1966, Dijkstra propuso reducir la dificultad con una serie de simples comandos estructurados *como if – then – else,* y *while – repeat – until*. Aunque esta estructura de comandos actualmente se encuentra en la mayoría de los lenguajes de programación, no era un avance evidente en 1968. Muchos especialistas en programación tuvieron preocupaciones en cuanto a la "interrupción del arte de programar", y que el código podría llegar a ser más complicado de desarrollar y comprender al emplear las estructuras de programación anteriores. Dijkstra estuvo al principio de acuerdo y advirtió, sin embargo, cuando los científicos estaban reunidos en la conferencia de la OTAN, referida a la crisis del software en 1968, que "el diagrama de flujo final no puede ser más transparente que el origen". No obstante, cuando los expertos reunidos en la conferencia de la OTAN con Dijkstra presente, estas conversaciones se enfocaron a hacer énfasis en la estructura del código (Naur y Randell, 1968, 144-284)[17].

Si bien no hubo un gran número de propuestas de solución para la crisis del software, la estructura anterior es ahora reconocida como un recurso importante para programar, como lo llama Bertrand Meyer "el inequívoco espagueti enredado" (1997, 678) de declaraciones que contribuyeron a ser parte de la crisis. Existen muchos tipos de estructuras como la anterior, la desarrollada por Böhm y Jacopini en 1966 es una prueba original al emplear solo tres símbolos [Π, Ω, y Δ] (figura 37). Los autores anteriores mostraron cómo estas tres estructuras podrían combinarse, sin una instrucción GOTO con el fin de producir programas completos de Turing. La implicación de su comprobación es que cualquier programa sin estructura que emplee el estatuto GOTO puede ser reescrito sin este, y que el programa computacional puede ser descompuesto en una estructura de subprogramas que se encuentran relacionados utilizando [Π, Ω, y Δ]; eliminando con lo anterior, el peligro de los saltos de los estatutos GOTO que se dirigen a localizaciones inesperadas. Sin embargo, esta prueba tardó más en ser conocida en la práctica antes de la carta de Dijkstra[18].

17 En la reunión referida a la estructuración del código, esta no fue identificada como una de las causas de la crisis del software. De hecho, ninguno de los asistentes que postularon actas en las reuniones, que hacen referencia a la programación estructurada o a el estatuto GOTO. Sin embargo, ellos hablan a menudo de la estructura del código y los módulos del código. Dijkstra también presentó un documento que se titula *Complexity Controlled by Hierarchical Ordering of Function and Variability*, dónde describe la agrupación del código en capas que se encuentran restringidas y que pueden comunicarse con capas encima de ellas. Mientras existen principios de estructuración en esta idea, es un tipo diferente de estructuración en relación a la de Böhm y Jacopini discutida en 1966 y a la del autor, discutida en 1968. En esencia, la estructura anterior fue una idea que fue ganando terreno en todo momento en la conferencia de la OTAN, pero que se encontraba en etapas iniciales de desarrollo (Naur y Randell, 1968).

18 Mientras que Böhm y Jacopini mostraron que era teóricamente posible en 1968, escribir programas si el estatuto GOTO, no fue posible en la práctica hasta que los lenguajes de programación pudieron incorporar tres estructuras: secuencia, iteración y selección. Después del desarrollo de estos lenguajes, los programadores que estaban cómodos usando el estatuto GOTO, y lo continuaron utilizando. Después de diecinueve años después del estatuto de Dijkstra en 1968, los especialistas aún consideran que es muy potente al emplearlo en estructuras complejas.

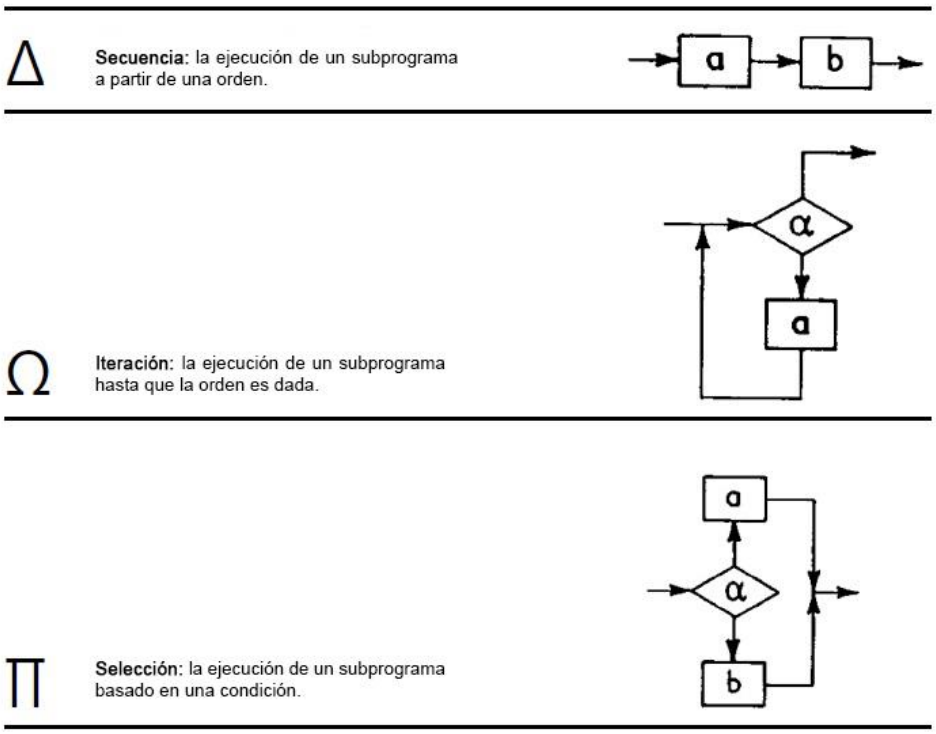


Figura 37. Las tres estructuras que Böhm y Jacopini demostraron que pueden ser llevadas a cabo y podrían ser combinadas para crear un mecanismo de Turing en 1968.

Módulos

Los subprogramas utilizados por Böhm y Jacopini tienen muchos sinónimos en la programación contemporánea: métodos, funciones, procedimientos y módulos. Cada término significa lo mismo en ideas generales con algunas diferencias entre ellos. Escogí utilizar la palabra *módulo* por las connotaciones de estandarización, el reúso, autocontención y el ensamblaje. Un módulo es definido por Wong y Sharp (1992, 43) como "una secuencia de instrucciones de programa acotados por una entrada y un punto de salida" para permitir "una relación particular de un problema con una tarea determinada" (estos pueden ser aplicados a un módulo realizado en *Grasshopper*, figura 38). Si los empleamos de manera correcta, los módulos tienen cinco principios de beneficio, de acuerdo con Bertrand Meyer (1997, 40-46):

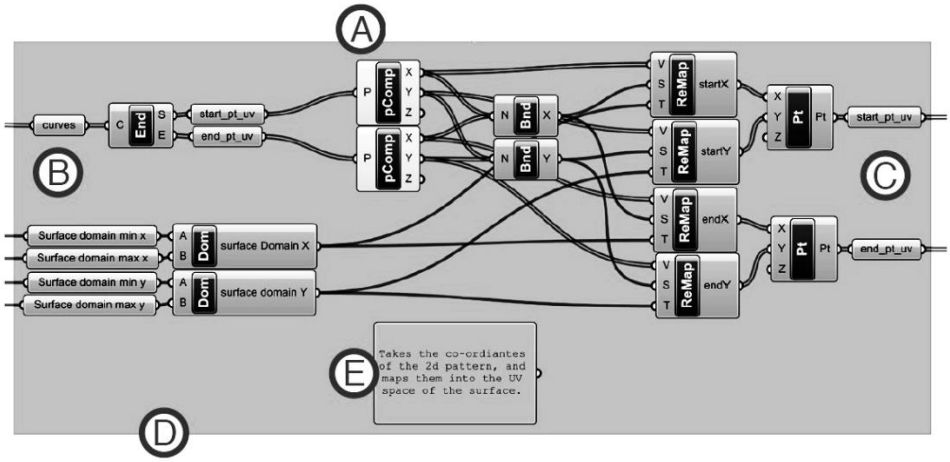


Figura 38. Un módulo típico en *Grasshopper*. Los nodos son módulos relacionados con otros.

A – Nombre del módulo.

B – Entradas de datos.

C – Salida de datos.

D – Las operaciones de los módulos son agrupadas para permitir la entrada y salida de datos a través de unidades.

E – Descripción de la tarea del módulo.

1. **Descomposición:** un problema complicado puede ser la descomposición de una serie a simples sub-problemas, cada uno contenido en un módulo. Los problemas de descomposición como el anterior pueden ser sencillos de resolver en equipos, a partir de que cada equipo puede trabajar en resolver cada uno independientemente.

2. **Composición:** si los módulos son adecuados en forma autónoma, podrán ser combinados para la formación de nuevos programas. Esto permite el manejo de conocimiento que se tiene dentro de la estructura de programación de cada módulo. Con el fin de ser compartido y reutilizado más allá de su entorno de trabajo original.

3. **Compactación:** si un módulo tiene gran contenido de datos, un programador podría comprenderlo sin necesidad de extender todo el código completo del programa. Es decir, un programador podría excluir los detalles al resolver el problema y trabajar en los detalles de cada módulo independiente. Dijkstra discute con preocupación lo anterior, ya que deja menos transparencia y deja menos tiempo para la abstracción de la estructura del programa (1968, 148). En contraparte, Thomas McCabe argumenta que la modulación mejora la difícil comprensión ya que se puede aplicar la complejidad ciclomática, enfatizando que "es una manera en la cual la complejidad del programa puede ser controlada" (1976, 54). Meyer señala que la modulación ayuda a la compresión, por parte del programador, de un código a través de nombres dados a entradas, salidas y los módulos propios (1997, 54).

4. **Continuidad:** un programa tiene continuidad cuando los cambios necesarios pueden ser realizados sin interrumpir los procesamientos de listas de datos. En un programa sin continuidad, el cambio de un módulo afectará a todos los módulos dependientes, provocando una reacción en cadena que beneficiará o perjudicará las posiciones de los módulos y, por consiguiente, las secuencias de procesamiento del programa. La continuidad tiene una importancia grande, en relación a cómo un programa puede ser "descompuesto", en términos de agrupamiento. David Parnas sugiere que los proyectos pueden ser rotos alrededor de las "difíciles decisiones de diseño iniciales que son cambiantes a lo largo del desarrollo del mismo" (1972, 1058), así que cada anticipación a los cambios son contenidos dentro de un módulo para no tener algún impacto significativo en los módulos de desarrollo posteriores.

5. **Protección:** cada módulo puede ser probado y depurado de forma individual para asegurarse que trabaja de manera correcta. Pero si algo resulta erróneo dentro de cualquier módulo, la unidad puede contener el error y detener su propagación en todo el sistema del programa (protegiendo al resto de los módulos de un error).

Los beneficios de la agrupación por módulos son tan relevantes que algunos lenguajes de programación modernos como C# y Java, son imposibles de usar si no hay código contenido en algún tipo de módulo. Java inclusive dejó de aplicar el estatuto GOTO, y algunos idiomas de programación más recientes – como Python y Ruby – nunca han basado su funcionamiento en lo anterior[19]. En su lugar, son utilizados los grupos de construcciones estructuradas de *switch – case* y *try – case* para objetos polimórficos.

19 Los lenguajes de programación *Python* y *Ruby* no basan de manera determinada el uso en el estatuto GOTO, pero se puede añadir mediante la importación de bibliotecas llamadas "Richie Hindle", contenidas de manera predeterminada en *Rhinoceros 5.0* en dónde se incorpora, por ejemplo, *Python*.

Las estructuras originales de tres, descritas anteriormente, como la de Böhm y Jacopini, ofrecen a los programadores componer y descomponer partes de programas, en secciones más pequeñas. Los debates continúan sobre cómo mejorar las estructuras del código, con el fin de aumentar la comprensión y reducir la complejidad, mejorando al mismo tiempo la continuidad y protección. Estos debates llenan secciones enteras en bibliotecas y ocupan un espacio importante en el desarrollo del diseño de software, relacionados con la sección SWEBOK versión uno de Hilburn del año 1999. Sin embargo, a pesar de los beneficios generalizados de la agrupación por módulos, los arquitectos tienden a producir modelos y definiciones paramétricas en lenguajes de programación visuales y recurren a construir modelos no estructurados (figura 39).

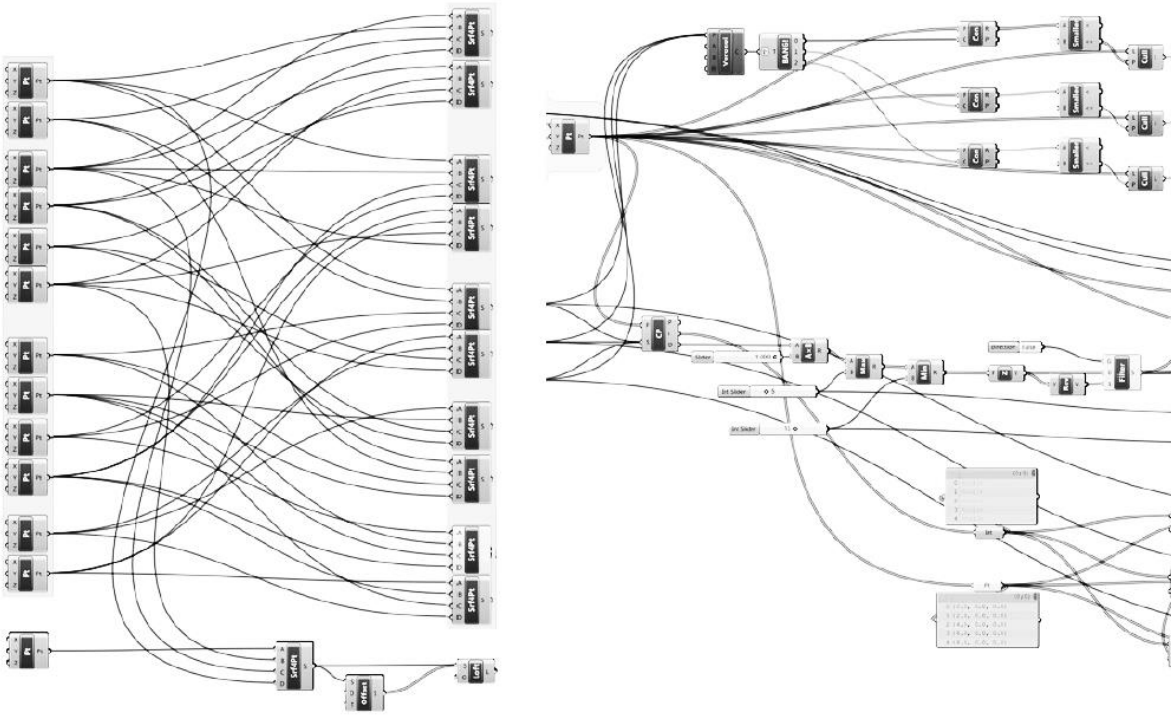


Figura 39. Ejemplos de dos definiciones de *Grasshopper* en “espagueti”, sin estructura. Ninguno de los modelos representados permite conocer, a través de las conexiones, las funciones de cada componente y parámetros, además de ser imposible la deducción simple como método de inspección inicial.

El proceso normativo de programación

En el libro *Inventing on Principle* del autor Bret Victor del año 2012, se describen los principios normativos en el proceso de programación como “edición del código, compilación y ejecución”. Esta secuencia de eventos se conoce comúnmente como el ciclo computacional de *Edición – Compilación – Ejecución*. En el ciclo (figura 40), el programador edita el texto del código [1], presiona el botón para activar en código [2] y luego espera. Posteriormente tiene que esperar el procesamiento de la computadora para la validación del código [3], luego se espera por la compilación del código, con el fin de que la computadora pueda interpretar las instrucciones dadas [4] y finalmente, la computadora ejecuta la lista de instrucciones [5]. Solo con lo anterior el programador podrá ver los resultados que el código ha producido [6]. El autor menciona que los buenos programadores deberán de emplear este medio de producción de manera mental, tratando de simular la ejecución del software de manera anticipada, y no esperar a que la computadora realice todo el procedimiento, con el fin de anticiparse a errores en el texto del código (2012, 18).

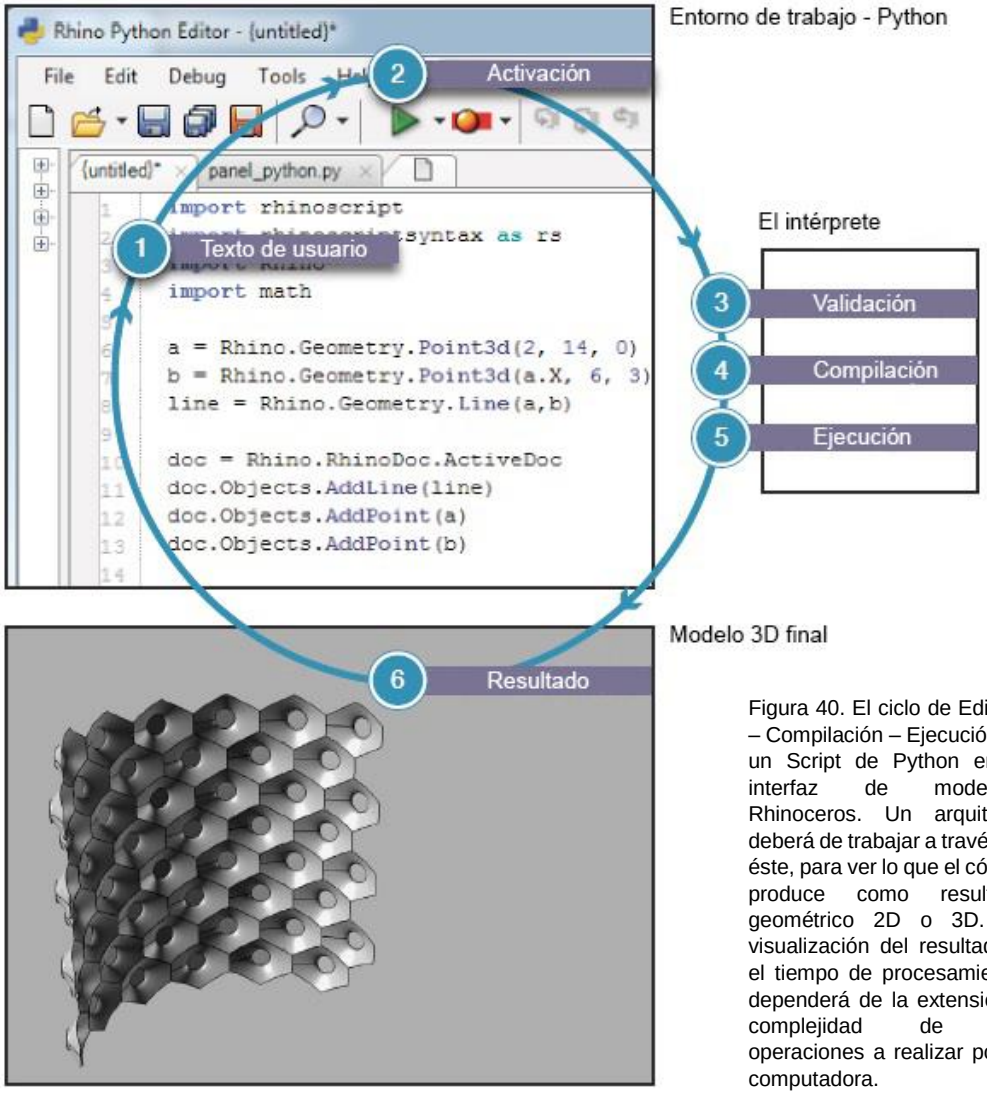


Figura 40. El ciclo de Edición – Compilación – Ejecución de un Script de Python en la interfaz de modelado Rhinoceros. Un arquitecto deberá de trabajar a través de éste, para ver lo que el código produce como resultado geométrico 2D o 3D. La visualización del resultado y el tiempo de procesamiento, dependerá de la extensión y complejidad de las operaciones a realizar por la computadora.

Para los arquitectos, este ciclo de trabajo *Edición – Compilación - Ejecución*, puede ser problemático, ya que en ocasiones los resultados no se aprecian en tiempo real. Ivan Sutherland llamó despectivamente a esto "escritura de cartas a la computadora" (1963, 8). Sin embargo, este procedimiento corto es una simulación mental a través de instrucciones dadas a la computadora, como los programadores frecuentemente lo hacen. Nigel Cross menciona a lo anterior en referencia a "el diseño, que es al inicio muy complicado, es difícil de llevar a cabo por procesos mentales puramente internos" (2011, 11). A través de la anterior afirmación, se puede decir que los diseñadores necesitan retroalimentación continúa, separada de su pensamiento interior; esta opinión es compartida por Sutherland, Victor, Lawson, entre otros. Este punto de vista se ve reforzado por la investigación de la cognición en el diseño, que muestra cualquier tipo de latencia entre las acciones del diseñador y la reacción de la computadora, la cual es problemática debido a los retrasos de tiempos de ejecución en el código, lo que hace difícil a los arquitectos evaluar cambios anticipados en los modelos. Como los arquitectos y diseñadores son incapaces de anticiparse a ver los cambios que realizan, Rick Smith advierte que un cambio en un modelo paramétrico "puede no ser detectado hasta mucho después de la fase de diseño, o peor aún, en la fase de construcción más cara" (2007, 2).

El ciclo de *Edición – Compilación – Ejecución* prevalece, ya que la mayoría de lenguajes de programación "fueron desarrollados para tarjetas perforadas" (Victor 2012, 28), en dónde - "el usuario podría escribir su programa sobre una serie de tarjetas y luego colocarlas dentro del operador de la computadora, y podrías regresar después" – como una suposición de lo que actualmente es la programación. Mientras que a través del funcionamiento de las tarjetas perforadas se puede explicar los orígenes del *"bucle"* de programación, ha habido muchos avances desde los días de las tarjetas perforadas[20]. En particular se han hecho avances significativos en las herramientas que utilizan los programadores para escribir código, conocidos como Entornos de Desarrollo Integrado (por sus siglas en Inglés IDE). Los IDE's modernos a menudo aumentan el ciclo de procesamiento *Edición – Compilación – Ejecución*, por lo que los programadores no tienen que esperar la retroalimentación. Por ejemplo, algunos IDE's identifican errores lógicos simples antes de ejecutar el código, y algunos sugieren y explican instrucciones de programación antes de emplear los comandos mientras los programadores escriben (conocido como función de autocompletar). Otros IDE's permiten la edición básica del código de funcionamiento, lo que permite a los programadores hacer cambios menores en los ciclos a través del *"bucle"* de compilación de gestión en la edición (esto se conoce como depuración interactiva). Estos tipos de IDE's a menudo refuerzan la noción que se tienen de los lenguajes de programación que fueron diseñados para las tarjetas perforadas, mientras que ofrecen formas más interactivas de realizar el ciclo *Edición – Compilación – Ejecución.*

Las evoluciones de los mecanismos interactivos de muchos IDE's no se han incorporado en la actualidad a entornos dónde los arquitectos escriben código. Como los programadores profesionales, los arquitectos usan la mayoría de las veces, lenguajes basados en el ciclo *Edición – Compilación – Ejecución*, por ejemplo, *RhinoScript*, que es un descendiente de una larga serie de dialectos de BASIC, que comenzó antes del año 1964.

20 Un *bucle* se refiere a una sentencia que se realiza repetidas veces a un trozo aislado de código, hasta que la condición asignada a dicho ciclo deje de cumplirse. Generalmente, un bucle es utilizado para hacer una acción repetida sin tener que escribir varias veces el mismo código, lo que ahorra tiempo, deja el código más claro y facilita su modificación en el futuro.

A diferencia de los programadores profesionales, que tienen las ventajas de la vanguardia, en el contexto de la arquitectura la ausencia de una moderna y buena IDE's requiere que los usuarios recuerden las funcionalidades de los comandos y la lectura extensa de código. Por ejemplo, un arquitecto o diseñador empleando un código para una tarea determinada, podría recordar la sintaxis del código escrita y cambiar un parámetro determinado, lo anterior requiere de una memorización de la estructura y las correspondencias geométricas visuales que el código produce al ejecutar. Por lo tanto, los arquitectos están exentos de los entornos de desarrollo modernos empleados por los programadores. Esta falta de interactividad en el proceso de programación es la causa de una latencia pronunciada entre la escritura del código de diseño y la generación de los resultados geométricos a través de computadora, lo que hace muy difícil la evaluación de cambios para los arquitectos. Sin embargo, es de gran importancia la estructuración de los procesos de producción en el diseño arquitectónico, al simplificar con un código representaciones geométricas y de datos complejas que pueden ser modificadas con la simple modificación de un comando o parámetro, lo cual permite un ahorro del tiempo en modificación final.

Programación interactiva textual

Aunque ninguno de los entornos de programación interactivos y textuales existentes es particularmente adecuados para los arquitectos, muchos programas de ambientes no textuales interactivos permiten la creación de geometría. Grasshopper, Houdini y GenerativeComponents superan los problemas de la realización de cálculos geométricos en tiempo real mediante la representación bi – tridimensional, a través de relaciones en Gráficas Acíclicas sin Dirección. Como lo mencioné anteriormente una *DAG* es un tipo de diagrama de flujo, dónde los nodos representan operaciones geométricas y de datos, por otra parte, los segmentos dirigidos representan el flujo de datos de un nodo a otro. Cuando se cambia un nodo, el modelo se actualiza mediante la propagación de los cambios a lo largo de los segmentos dirigidos con el fin de actualizar los nodos afectados. Esto minimiza los cálculos involucrados en la actualización del modelo, ya que los únicos nodos recalculados serán afectados por los cambios. En lugar de recalcular cada operación geométrica, como el ciclo *Edición – Compilación – Ejecución*, la actualización selectiva de una *DAG* ahorra cálculos geométricos innecesarios, comprimiendo mucho el tiempo entre la escritura y la ejecución del código.

Mientras la programación visual permite a los arquitectos trabajar de forma interactiva, no habrá limitaciones cuando se compare con la programación textual. Los entornos de programación visuales no son compatibles con las estructuras de código elegantes que son empleadas en la programación textual. El aprendizaje de un texto en un lenguaje de programación toma más tiempo y esfuerzo que un lenguaje de programación gráfico, pero este esfuerzo se recupera rápidamente cuando la complejidad de los problemas se vuelve muy grande. La programación gráfica es más fácil de aprender debido a la interactividad visual que proporciona el entorno de programación que los usuarios requieren. Mientras que la programación gráfica interactiva tiene más éxito en el dominio de la arquitectura, como modelos de producción geométrica bi y tridimensional, queda la oportunidad de utilizar un lenguaje de programación textual que combine las ventajas estructurales de la programación basada en texto con la interactividad y facilidad de uso de un entorno de programación gráfica visual.

3.2 – Introducción a Python

La programación ofrece a los usuarios la gran posibilidad de automatizar tareas, hacer decisiones, realizar complejos cálculos y manipulaciones geométricas, que pueden ayudar en la actividad diaria del arquitecto o diseñador. Esto puede permitir que miles de cálculos se produzcan basados en condiciones dinámicas, algo que llevaría demasiado tiempo en comparación un proceso mental humano. Como herramienta para la iteración, análisis y producción del diseño, la programación no tiene límites. Por lo tanto, la programación también ofrece un nuevo lenguaje para comunicar al mundo nuestras ideas en cualquier disciplina, desde las ciencias, la ingeniería hasta el arte, utilizando el código como un nuevo medio progresivo de producción. En este apartado haré una pequeña introducción a este lenguaje de programación con el fin de poder comunicar nuestras ideas a través de un código con una gramática poderosa y particular.

Por otra parte, me gustaría aclarar desde esta sección que el funcionamiento de los programas a explicar, así como la terminología de comandos serán en idioma inglés, ya que es una característica estándar en la mayoría de los lenguajes de programación, con el fin de poder ser usados por la mayoría de las personas involucradas en la disciplina computacional alrededor del mundo.

3.2.1 – Macros

El programa de representación geométrica computacional sobre el cual trabajaremos será *Rhinoceros 5.0 SR6*, el cual basa su funcionamiento en una interfaz de línea – comando. Esto significa que el usuario puede controlar todos los procesos utilizando solo el teclado e introduciendo comandos, con ayuda del mouse para los desplazamientos y movimiento de la geometría. Cuando se escribe un comando y se presiona la tecla *enter*, el programa ejecutará la orden. Desde el movimiento del ratón, esta interfaz de usuario está basada en una línea de comandos que es primitiva. Podemos escribir lo siguiente:

Line 0, 0, 0, 0, 0, 0

El anterior comando significa – Línea a partir del punto *x=0, y=0* y *z=0* hasta *x=10, y=0* y *z=0* – que se traduce como “dibujar línea a partir del punto *x* cero hasta *x* diez, cuya longitud final sería de 10 unidades”. También se pude realizar un *click* sobre el botón de la herramienta Line y seleccionar el puerto de vista para definir el inicio y final de la línea. Debido a esta segunda interfaz gráfica, algunas personas optan por esta y dejan la de la línea de comandos. Algunos usuarios, por otra parte, prefieren el uso de la línea de comandos. Los programadores no tienen esta dualidad de elección, ya que prefieren siempre la línea de comandos.

La forma más simple de programación en *Rhinoceros* es utilizando macros, por medio de RhinoScript, la cual es una forma muy sencilla para automatizar procesos. Me limitaré a realizar una pequeña explicación de lo que son los macros, en parte por ser algo sencillo y porque en algún momento podríamos simular el funcionamiento de un macro.

Un macro es una lista pregrabada de los comandos asignados a *Rhinoceros* para ejecutar. El comando para realizar una línea en el anterior ejemplo es muy sencillo de comprender. Si deseamos realizar seis líneas más sería muy cansado escribir o copiar seis veces más el anterior comando, entonces optamos por usar macros. Los macros permiten automatizar

tareas que normalmente nos podría llevar algo de tiempo en la repetición de un mismo proceso, las macros no pueden realizarse automáticamente ni se van a ejecutar sin una orden asignada. Son parecidos a policías de tráfico en ese sentido. Un ejemplo de macro más sofisticado sería:

```
_SelNone
_Polygon _NumSides=6 w 0, 0, 0 w 10, 0, 0
_SelLast
-_Properties _Object _Name RailPolygon _Enter _Enter
_SelNone
_Polygon _NumSides=6 w 10, 0, 0 w 12, 0, 0
_SelLast
_Rotate3D w 0, 0, 0 w 10, 0, 0 90
-_Properties _Object _Name ProfilePolygon _Enter _Enter
_SelNone
-_Sweep1 _SelName RailPolygon _SelName ProfilePolygon _
            Enter _Enter _Closed=Yes Enter
```

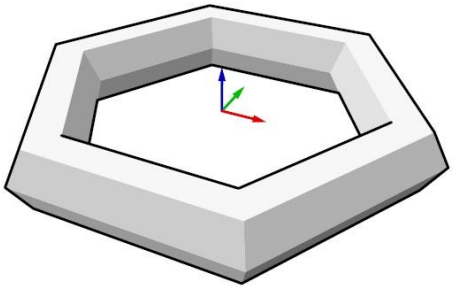


Figura 41. Un Toro en hexágono realizado a partir de una línea de comandos como macro de producción.

El código anterior producirá un Toro hexagonal una y otra vez sin detenerse; podría ser muy útil, pero no es flexible. Se puede escribir todo el texto en la línea de comandos, o se puede compactar este código en un botón que funcione como herramienta gráfica de ejecución de comandos.

Un aspecto importante a tomar en consideración es la utilización de los comandos en las líneas de código y la localización en la interfaz de *Rhinoceros*. El uso de guiones obligará al programa a utilizar los nombres de los comandos en inglés, independientemente de la configuración de idioma en la interfaz de trabajo personalizada. Siempre se deben de utilizar los nombres de comandos en inglés, ya que es la única garantía de que el texto escrito funcionará adecuadamente en todas las etapas de un procesamiento geométrico.

No existen límites de complejidad para un macro, ya que se pueden seguir adicionando comandos sin restricción alguna, pero no se podrá avanzar a más allá de los límites que se encuentran en los comandos asignados de *Rhinoceros* y que son el aspecto esencial de cualquier código basado en macros.

3.2.2 – Scripts

Las limitaciones de las macros han llevado al desarrollo de lenguajes de guiones [Scripting]. Los guiones [Script] son un apartado medio entre las macros, los programas verdaderos [compilaciones] y plugins. A diferencia de las macros pueden realizar operaciones matemáticas, evaluar condiciones variables, responder a su entorno y comunicarse con el usuario. A diferencia de los programas, no necesitan ser compilados antes de ejecutar. El software Rhinoceros implementa el lenguaje de programación estándar *Microsoft Visual Basic Scripting Edition*, así como el lenguaje de programación *Python*. En este apartado realizaré una introducción a este último lenguaje de programación para comprender su utilización dentro del programa Rhinoceros. Los *Scripts*, entonces, son archivos de texto que interpretan todas las líneas a la vez; pero aquí está la parte interesante, a diferencia de las macros, los scripts tienen el control de la ejecución de cada una de las líneas que conforman un código. Este control de flujo permite que la secuencia de ejecución de

comandos pueda omitir ciertas instrucciones o repetir otras. El control de flujo se consigue mediante algo llamado “Evaluación condicional” y debemos familiarizarnos con las normas gramáticas de *Python* antes de que podamos sacar provecho de los flujos de procesamiento.

Las reglas de este lenguaje de programación se refieren generalmente como la sintaxis e indican que es válido o no:

1. “Aquí no hay pastel de piña” **>>válido**
2. “Pastel de piña aquí no hay” **>>inválido**
3. “Aquí, pastel de piña no hay” **>>válido**
4. “Aquí no hay pastel de manzana” **>>inválido**

La lista anterior es una comprobación de validez para las reglas de sintaxis en español. La primera y tercera líneas son propias de español y comunes en su uso, las otras no lo son. Sin embargo, hay ciertos grados de error. Nadie va a entender mal la segunda línea solo porque el orden de las palabras es diferente. La cuarta línea ya es un poco más difícil, ya que cuenta con una palabra que describe otro ingrediente del pastel. Aunque la mayoría de nosotros somos lo suficiente inteligentes para comprender las cuatro líneas, la computadora no.

Python es un lenguaje de programación muy intuitivo para los programadores principiantes o avanzados. Ofrece una sintaxis simple y eficiente, así como funciones de programación de gran alcance, con capacidades de programación orientada a objetos y una gran base de bibliotecas desarrolladas por la comunidad de usuarios y desarrolladores involucrados en este lenguaje de programación computacional. Además, dado que *Rhino Python* está disponible en los sistemas operativos Windows y Mac, los scripts se pueden ejecutar de manera exacta en ambas versiones de *Rhinoceros*.

3.2.3 – Ejecución de Scripts

Existen muchas maneras de ejecutar *scripts* en *Rhinoceros*, cada una tiene sus ventajas y desventajas. Se pueden almacenar los scripts como archivos de texto externos y cargarlos en la interfaz de *Rhinoceros* con el fin de ejecutarlos. También se pueden utilizar *scripts* para editarlos en los entornos de programación de *Rhinoceros* para ejecutar o modificar las secuencias de comandos directamente desde los editores gráficos de este programa. La última opción es insertar secuencias de comandos en la barra de botones, lo que hace que sea muy difícil de editar, pero hace más fácil la distribución y uso en un entorno de trabajo gráfico.

En este documento voy a utilizar el método de construcción y redacción, la cual creo que es la mejor manera de trabajar con guiones simples y complejos. Con el fin de ejecutar un script a través del editor de construcción. Para poder comenzar con lo anterior tenemos que ejecutar el comando **_EditPythonScript** con el fin de activarlo y comenzar a escribir. Tal interfaz y ejemplo se muestra en la figura 42.

Todos los códigos sencillos que explicaré en esta sección se podrán copiar y pegar en esta interfaz con el fin de hacer comprensible la sintaxis y función del código.

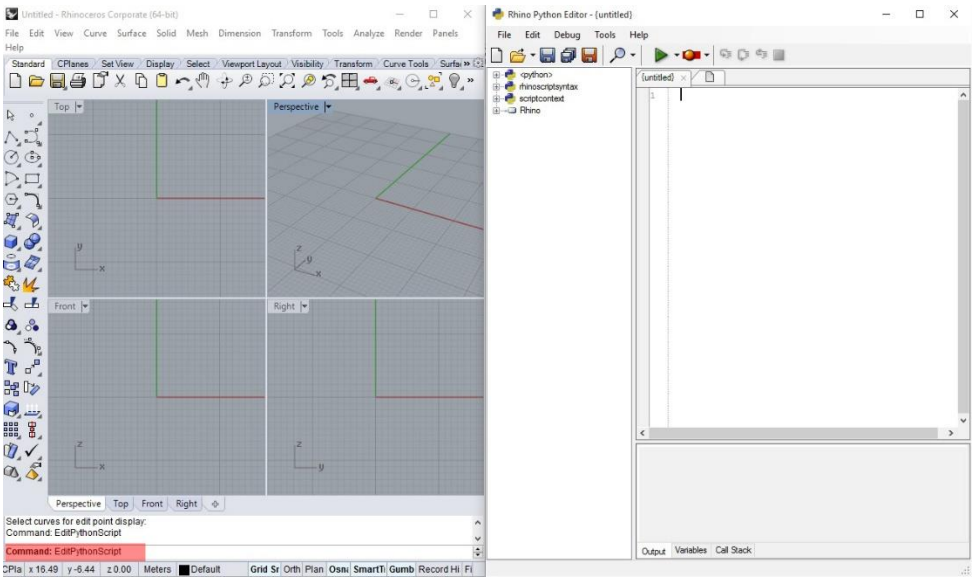


Figura 42. Interfaz de trabajo del editor de Python en Rhinoceros 5.0. La ejecución del comando *EditPythonScript* habilitará este editor con el fin de ejecutar los procesos en conjunto con Rhinoceros 5.0.

3.2.4 – Origen del lenguaje

Al igual que los lenguajes humanos convencionales, los lenguajes de programación computacionales se agrupan en conjuntos de clústers[21]. *Python* es un lenguaje de programación de alto nivel, haciendo énfasis en su diseño y utilización para hacer de fácil comprensión para los humanos. En el otro extremo se encuentran los lenguajes de bajo nivel (a menudo referido como código de máquina), que son lenguajes muy fáciles de comprender. En la parte media de la clasificación se encuentran los lenguajes de programación en *C* o *C*++ lo cuáles ofrecen un grado de abstracción entre la computadora y el código. Como lo mencioné, *Python* es un lenguaje de muy alto nivel, ya que es muy fácil de leer (tal y como el uso del idioma inglés estándar), además de que no necesitamos asignar una memoria específica para gestionar su funcionalidad para la declaración de variables.

Python fue lanzado al público usuario por primera vez en 1991, desde entonces ha crecido enormemente hasta convertirse en un lenguaje de libre acceso con un grupo de desarrolladores y usuarios que hoy en día se cuentan en millones. Fue diseñado por el programador holandés Guido van Rossum[22].

21 Un *Clúster* referido a la agrupación y clasificación de lenguajes de programación se define como el resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran. Los clústeres son usualmente empleados para mejorar el rendimiento y/o la disponibilidad por encima de la que es provista por un solo computador típicamente siendo más económico que computadores individuales de rapidez y disponibilidad comparables. Python es un leguaje de alto rendimiento basado en clústers ya que permite ejecutar tareas que requieren de gran capacidad computacional, grandes cantidades de memoria, o ambos a la vez.

22 Para mayor información al respecto del proyecto Python se puede visitar la página web: https://www.python.org.

Suponiendo que el usuario podría estar leyendo estas páginas sin alguna experiencia anterior al acto de programar, el siguiente ejemplo dará algo de problemas de comprensión si una posterior explicación.

```
somenumber = rs.GetReal("Line length")
line = rs.AddLine ([0, 0, 0], [somenumber, 0, 0])
print "Line curve inserted with id", line
```

Al principio no se tiene ni la menor idea de lo que las anteriores líneas significan, también podría causar confusión por la línea que dice: *"rs.GetReal ()"*, pero en general y en palabras simples es como pedir un pastel de piña en una tienda de pasteles, cuya traducción sería:

- Preguntar a Rhinoceros para asignar un número a algo llamado "somenumber".
- Decir a Rhinoceros que añada una línea desde el punto de origen del plano cartesiano con coordenadas [*x*=0, *y*=0 y *z*=0] hasta una coordenada con un parámetro numérico en "somenumber" en la expresión [somenumber, 0, 0].
- Imprimir el mensaje "*Line curve inserted with ID*" y dibujar la línea en el espacio tridimensional de Rhinoceros.

Al hacer la traducción del código al español, no es tan difícil, al menos no en un nivel inicial. También se podría optar por escribir el código de una manera más compleja, pero esto no es algo a tratar. La sintaxis anterior es similar a la gramática en inglés, al igual que las palabras utilizadas para definir los comandos, por lo que sugiero estar familiarizado en este idioma.

Como lo mencione anteriormente, existen tres cosas que las sintaxis deben de soportar y que el Script anterior también utiliza:

1. **Control de flujos:** dependerá de la salida de datos de la segunda línea, algunas líneas no serán ejecutadas, pero si procesadas.
2. **Datos variables**: somenumber es usado para definir una variable numérica.
3. **Comunicación**: se le pregunta al usuario acerca de remplazar información numérica y textual, con el fin de ser informado de los resultados finales.

Control de flujos

Este control de flujos de datos en Python lo utilizamos para saltar ciertas líneas en el código o para ejecutar otras en más de una ocasión. También podemos utilizar el control de flujo de datos para saltar a diferentes líneas en nuestro *script* y viceversa. Podemos añadir enunciados condicionales en el código que nos permitan proteger ciertas partes, tales como las estructuras *If – Else* y *Else If*, las cuáles vamos a discutir más adelante en la incorporación al lenguaje de programación Python. Un enunciado condicional típico puede ser el siguiente:

"Usted debe tener una altura mínima de 1.50 metros para poder subir a este caballo"

Esta línea de código usa una condición (altura mínima de 1.50 metros) para evaluar si se permite a la persona con esta condición montar o no al caballo. Los enunciados condicionales como éste pueden ser empleados indefinidamente. También podríamos añadir una máxima altura, o una limitante del peso de la persona, o la prohibición por dañar la salud e integridad caballo.

En lugar de saltar líneas de código a partir de condicionales, también podemos repetir las líneas. Podemos hacer lo anterior estableciendo un número fijo de repeticiones:

"Añadir 5 cucharadas de canela al café"

O de nuevo, usar una evaluación condicional:

"Mantener la adición de leche hasta obtener una mezcla moldeable"

La repetición de líneas de código se denomina en inglés "*Looping*". Hay varios tipos disponibles, pero todos ellos trabajan más o menos de la misma manera.

Datos variables

Siempre vamos a querer que nuestro código sea lo más flexible y dinámico posible, para esto tenemos que asegurarnos de manejar todo tipo de situaciones diferentes. Para hacer lo anterior, debemos saber almacenar variables. Por ejemplo, podríamos querer guardar una selección de curvas en nuestro modelo 3D para que podamos eliminarlas en una etapa posterior. O tal vez nuestro *script* necesita agregar otra línea desde el puntero del ratón sobre el origen de la escena en 3D. O comprobar la fecha actual para ver si nuestro software ha caducado. Lo anterior se trata de información que no está disponible a la hora de escribir nuestro código.

Cada vez que necesitamos almacenar más datos o realizar cálculos u operaciones lógicas más complejas, necesitaremos de más variables para que la ejecución y procesamiento de datos puedan llevarse a cabo de manera más rápida. Dado que estas operaciones son dinámicas, no se pueden agregar a una ejecución de comandos antes de la ejecución. Necesitaremos de marcadores de posición.

En el ejemplo anterior, la palabra nombrada "somenumber" es un marcador de posición para un número. El cual comienza por ser sólo un nombre sin ningún dato, pero que se le puede atribuir un valor numérico en la línea:

```
somenumber = rs.GetNumber ("line length")
```

Luego podemos recuperar este valor específico cuando añadimos una curva a la línea de Rhinoceros:

```
curve = rs.AddLine ([0, 0, 0], [somenumber, 0, 0])
```

Todas las otras coordenadas que definen el objeto de línea están codificadas en el script. No hay límite en la frecuencia con la que una variable se puede leer o asignar un nuevo valor, pero nunca puede contener más de un valor ya que no hay sistema recuperación [*backup*] en este editor de programación. Además de los números, también podemos almacenar otros tipos de datos en las variables. Por el momento, voy a hacer una restricción a los más importantes, además de los más especiales para la corrección de errores:

01 – Enteros
02 – Dobles
03 – Booleanos
04 – Cadenas
05 – Variable nula

3.2.5 - La anatomía del *script*

Conocimos ya las características de los *macros* y los *scripts*, pero aún no todo el potencial que tiene el lenguaje de programación Python, así como algunas funciones de Rhinoceros que pueden resumirse en lo siguiente (figura 43):

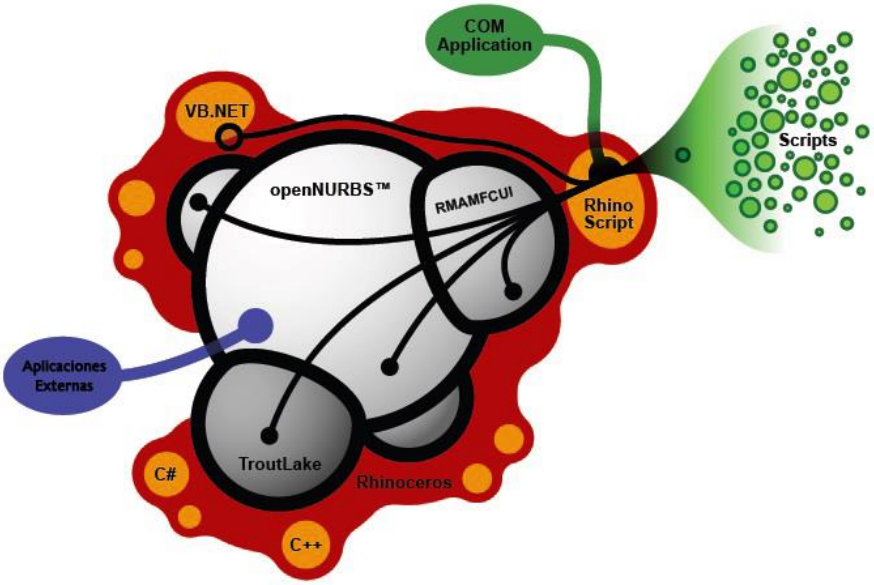


Figura 43. El esquema muestra los complementos extra que pueden ser añadidos a Rhinoceros [verde], a partir de los cuáles el programa se desarrollada [amarillo] y las características adicionales que se le pueden incorporar mediante la adición de librerías [rojo].

La figura anterior es un esquema de las herramientas que pueden ser usadas e incorporadas en Rhinoceros, ya sea para usuarios o desarrolladores. Para explicar de mejor manera este gráfico, se puede decir que las librerías de código son el núcleo de función de Rhinoceros, son esencialmente colecciones de procesos y objetos que son utilizados por los usuarios para hacer más sencilla la interacción del usuario final con el programa. Las librerías más conocidas son las *openNURBS*, las cuáles son de código abierto y pueden ser exportadas a sistemas operativos diferentes, tales como *Linux* y *Unix*[23]. Estas librerías brindan la información necesaria de escritura y lectura para métodos de generación geométrica.

El programa *Rhinoceros* por sí mismo (grupo de color rojo) se encuentra alrededor de todos estos núcleos de librerías, el cual también se puede ampliar mediante la incorporación de *plugins*[24]. El programa puede incorporar más funcionalidades que las iniciales, mediante la incorporación de *plugins* desarrollados por terceras personas, que no tienen influencia en el desarrollo del código inicial del programa; es por esto que los desarrolladores, *Robert McNeel and Associates* no modifican el código central del programa, sino que permiten la incorporación de códigos extra con el fin de dar solución a un problema en particular. En los modelos paramétricos este aspecto es muy común, ya que hay ocasiones en que los arquitectos necesitan herramientas adicionales para dar solución a un problema, por ejemplo, de índole estructural y podrán incorporar una herramienta computacional adicional para dar solución al problema específico.

23 Unix y Linux son dos sistemas operativos portables, multitarea y multiusuario, desarrollados en el inicio de 1969; ambos funcionan a partir de un sistema de sintaxis particular que puede ser implementada por programadores iniciales y usuarios finales, con el fin de modificar el sistema o dar al sistema características adicionales de funcionamiento. Es necesario conocer la sintaxis de códigos u órdenes particulares de estos dos núcleos base de sistemas operativos actuales, tales como iOS, Ubuntu, Fedora, OpenSUSE, entre otros.

24 Un plugin es complemento a una aplicación que incorpora una función nueva y generalmente muy específica.

Lo anterior da la posibilidad a los arquitectos y diseñadores de desarrollar sus propias librerías de código que puedan dar solución a sus problemáticas particulares, con la seguridad de que este componente podrá ser procesado dentro de su modelo paramétrico, ya que se incorporará y procesará en relación a los códigos centrales en el núcleo del programa. Por lo tanto, es importante conocer las posibles sintaxis que pueden desarrollarse dentro del lenguaje de programación *Python*, con el fin de poder incorporarlas de manera externa o en conjunto con el funcionamiento del programa *Rhinoceros*, sin la necesidad de trabajar con un lenguaje más complejo de programación, tal como *C++, C, Java* o cualquier otro.

Los círculos de color naranja representan los complementos o *plugins* que pueden ser escritos en *C++* y todos los lenguajes que soportan el marco de la plataforma de desarrollo y ejecución de aplicaciones *Microsoft DotNET*[25] (*VB.NET, CSharp, Delphi, J#, IronPython, etc.*), los cuáles, como ya mencioné, proporcionan tareas de procesamiento y ejecución más específicas para el programa.

3.2.6 – Fundamentos

Una vez que es ejecutada una secuencia de comandos a través del editor de construcción de Python en Rhinoceros, el intérprete de este editor analizará la sintaxis de los comandos, lo anterior no será una ejecución, sino una interpretación inicial de funcionamiento correcto; durante este proceso el editor es capaz de localizar los errores y señalar las líneas y comandos que han fallado, a esto se le conoce como *PrePass*, y se observará un cuadro de resultados (figura 44):

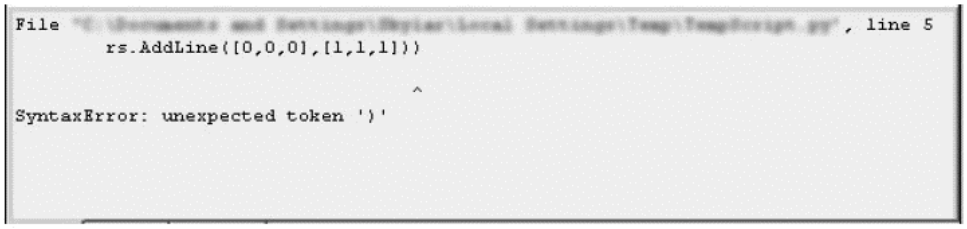


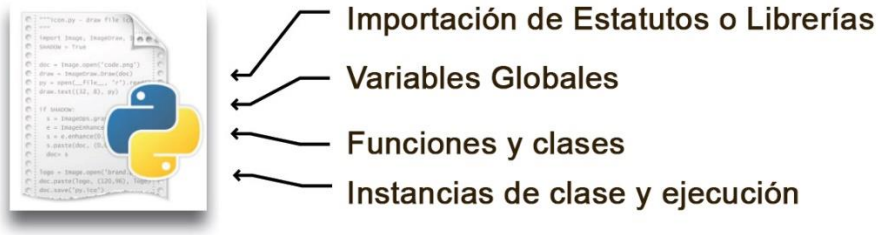
Figura 44. Este cuadro de diálogo en el editor, muestra un error localizado en un comando o sintaxis, antes de que sea ejecutado.

Antes de realizar una ejecución de los comandos, el editor a través de su compilador ha encontrado un error, el cual es un problema con la sintaxis *(SyntaxError: unexpected token*), así que decidió no realizar la ejecución del *script*. Si el *script* se bloquea mientras se está ejecutando, la fuente del mensaje de error no será el compilador de *Python*, sino la interrupción de la ejecución del programa base, o sea *Rhinoceros*. Sin embargo, incluso algún *script* sin errores de sintaxis puede no funcionar como se espera. Para que un *script* de *Python* se ejecute de manera correcta, se deben de cumplir algunas reglas, además de evitar errores "ortográficos" en la escritura de la sintaxis[26]. Cada guion de la sintaxis debe de implementar una estructura que dice al compilador que hacer (figura 45):

25 .NET [*DotNET*] es una plataforma de desarrollo y ejecución de aplicaciones de Microsoft, que hace énfasis en la transparencia de redes, con independencia de la plataforma de hardware y que permite un desarrollo rápido de aplicaciones. Es una respuesta al creciente mercado de los negocios en entornos WEB, el cual ofrece una manera rápida, económica, robusta y segura de desarrollo de aplicaciones para cualquier tipo de dispositivo para empresas.

26 Hago referencia a los errores "ortográficos" en la escritura del código en el lenguaje de programación Python, tales como dos puntos, doble guion, guiones, signos matemáticos, comas, comillas, entre otros.

Figura 45. Reglas a tomar en cuenta antes de iniciar y al desarrollar un *script* en el lenguaje de programación *Python* en *Rhinoceros 5.0*.



La declaración o estatuto en importación de librerías permite al usuario final añadir diferentes módulos de construcción en la sintaxis del código, ya sea en Python o mediante la descarga de códigos desarrollados de manera externa. La importación de módulos permite al usuario métodos de acceso externos al archivo de código y hacer objetos de referencia, funciones u otra información. Existen varios tipos de importación de estatutos o declaraciones*: import X, from X import, from X import a, b, or c, X = __import__ ('X')*, cada uno con sus ventajas y desventajas. Para simplificar el significado de cada uno de los anteriores, mencionaré que importaremos todos los elementos contenidos en la variable *"X"* y que los aplicaremos al código que desarrollaremos.

Las variables globales son variables a las cuáles se puede acceder en cualquier parte de la sintaxis del código, tales como funciones externas, funciones internas y clases contenidas. Se refieren a la limitación o accesibilidad de una variable a través de las diferentes partes del código. A las variables globales, se puede acceder de manera completa, mientras que ciertas variables pueden estar limitadas a ciertas áreas de modificación en su código. Por ejemplo, cualquier variable que se produce dentro de una clase o una función, estará limitada a los procesos contenidos dentro de esa función. Esto significa que no se pueden utilizar estos procesos fuera o como parte de otra función (a menos que pasen como puntos de entrada o salida definidos en el código) [27]. Por ahora, no es necesario tratar a profundidad este aspecto, ya que vamos a suponer que nuestras variables serán globalmente accesibles.

Las funciones son bloques de código que compactan ciertas funcionalidades en paquetes pequeños. Las funciones pueden tener variables de referencia en entrada, las cuáles proporcionan una producción y hacen copias de tareas importantes. Las clases son similares, ya que proporcionan una oportunidad para la producción de códigos en módulos para empaquetar o comprimir ciertas partes del código, mientras que proporcionan, por otra parte, otras herramientas de gran alcance. Las funciones y clases deben de tomarse a consideración antes de ser utilizadas, por esta razón las secciones de funciones y clases anteriores deben de estar a consideración, antes que las instancias de clase y de ejecución; esto significa que antes de llamar a los paquetes de clases y funciones, necesitaremos crearlas para poder hacer referencia a ellas y ejecutarlas.

27 Para mayor información acerca de las funciones y sus características, se puede consultar la sección 4 referida a Operadores y Funciones del manual *Python for Rhinoceros 5.0, Revisión 3. Año 2011.*

3.2.7 – La sintaxis del código *Python*

En relación a la estructura anterior explicada y descrita de manera gráfica, incluyendo los estatutos de importación y de librerías (que siempre son necesarios), las variables globales, las funciones, clases y las instancias. La sintaxis de escritura debe de tomarse en consideración, ya que de ella dependerá la correcta lectura por parte del intérprete de Python, además de una organización adecuada y una correcta ejecución en geometría bi y tridimensional. El lenguaje de programación Python es muy sensible a cualquier error escrito en la sintaxis y puede demorar algún tiempo en corregir esta falta, ya que las estructuras de código suelen ser complejas en la mayoría de los casos de geometría tridimensional representativa. Si el arquitecto o diseñador escribe un comando con una letra mayúscula o minúscula, el intérprete ejecutará una acción diferente por esta diferencia de tamaños en los caracteres de los comandos. Este tipo de características deberán de estar presentes y ser muy consideradas en variables de funciones, ciclos o estatutos de condición dentro de la organización de la sintaxis. En este ejemplo se muestra una sintaxis simple con una estructura correcta para su posterior compilación y ejecución:

```
import rhinoscriptsyntax as rs                                   << Estatuto de importación
# Guión o Script desarrollado por Skylar Tibbits el 04 de Enero de 2012   << Comentarios

strInfo = "Esto es solo un ejemplo"                              << Variable global

Class (Ejemplo):                                                 << Clase primaria

  def simpleFunction (text):                                     << Declaración de función
    print (text)                                                 << Código a ejecutar

simpleFunction (strInfo)                                         << Ejecución de la función
```

Un aspecto relevante de las sintaxis que pueden ser producidas en *Python* es la posibilidad de importar librerías externas al código con una simple orden "*import x as xn*", lo cual nos permitirá incorporar características particulares para las problemáticas en los procesos del diseño arquitectónico. Una de esas particularidades es, por ejemplo, importar una librería con módulos de Grasshopper con el objetivo de resolver aspectos geométricos con algún componente de este lenguaje de programación visual.

El depurador [*Debugger*].

Es una herramienta esencial para cualquier programador, en el lenguaje de programación Python este depurador o intérprete de comandos se encuentra incorporado y trabaja en conjunto con Rhinoceros con el fin de representar los resultados a partir de la compilación y ejecución del código basado en una sintaxis. Una de las funciones principales del depurador es localizar errores en la sintaxis del código, normalmente al ejecutar el código, con lo que responde *"bug in your code"*, que significa que algo está mal en el código a ejecutar[28].

28 El primer error [*bug*] en un código en un lenguaje de programación textual de computadora fue encontrado en 1947, cuando la máquina de cálculo *Aiken Relay* tuvo problemas en la Universidad de Harvard. La investigación de este error mostró que se encontraba "atrapado" en la máquina. Los programadores tuvieron que expandir el código para localizar el error; fue por esto que se incorporó un depurador en la mayoría de los lenguajes de programación textuales con el objetivo de localizar errores antes de la ejecución completa de un código computacional.

El depurador hace el trabajo de los usuarios al escribir código basado en *Python*, más sencillo para la localización de errores, sin embargo, esto puede ser algunas veces extremadamente difícil, especialmente si el código tiene incorporados en su sintaxis varios ciclos, estatutos condicionales, funciones, clases y cientos o miles de líneas. Por otra parte, esta herramienta nos permite colocar un punto de inicio o fin en cualquier línea del código, que puede iniciar o terminar la ejecución en un comando determinado en la cual tenemos interés, además de poder ver los procesos completos de esa línea o comando; adicionalmente, nos permite incorporar pausas a una ejecución de un comando y reiniciar a partir de esta pausa. El depurador representa una interfaz de dibujo y modelado tridimensional con código, que nos permitirá escribir y manipular el comportamiento del lápiz [interfaz del lenguaje de programación] en el papel [representación en las ventanas de visualización] (figura 46).

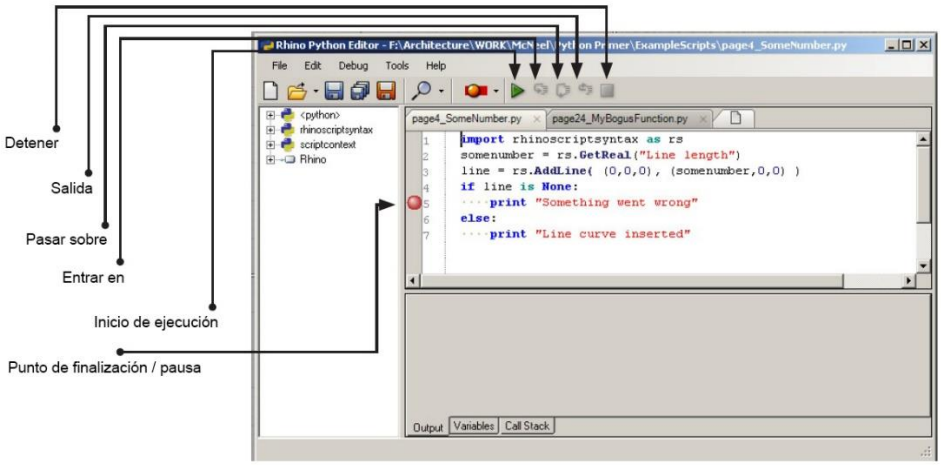


Figura 46. Las principales tareas de ejecución de un código basado en Python, a partir del depurador de compilación y ejecución en Rhinoceros.

Las opciones de salida, pasar sobre y entrar en, se utilizan con el fin de conocer el funcionamiento de un comando o línea y todos sus componentes. Después de que cada línea del código es ejecutada, el depurador mostrará las variables, objetos o nombres de expresiones, así como sus valores y tipos. Conforme las líneas del código sean ejecutadas y procesadas se actualizarán directamente a partir de los cambios realizados con anterioridad. Muchos errores en la ejecución pueden ser solucionados rápidamente y ser ajustados en las líneas de comandos en el código utilizando el depurador de *Python* en *Rhinoceros* (figura 47).

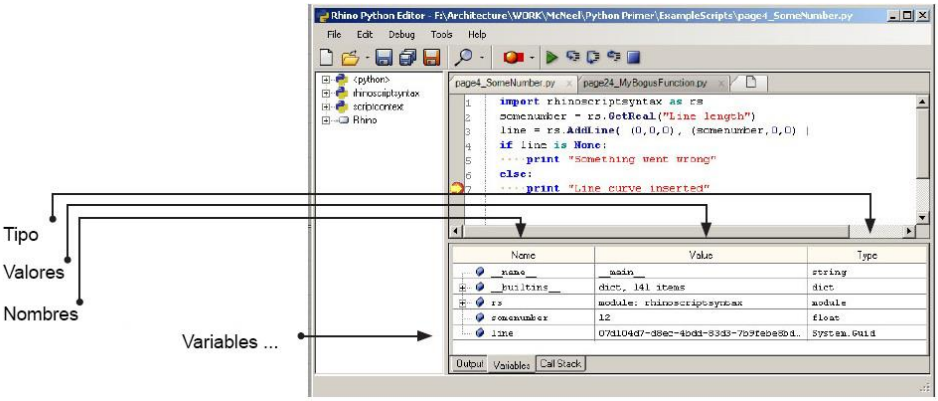


Figura 47. La compilación y ejecución del código se realiza hasta la línea número 07, la cual nos muestra a través del depurador, el tipo de operación o comando, los valores asignados y calculados y, el tipo de función que es.

3.3 – La programación textual en el diseño

Python, al igual que todos los entornos de trabajo para lenguajes de programación textuales posee una sintaxis definida y que se debe cumplir, con el objetivo de representar ideas de los usuarios finales, como arquitectos y diseñadores. Un aspecto importante al utilizar esta herramienta es el grado de profundización y personalización de código que nos permitirá producir ideas variables sobre un mismo concepto. Por otra parte, dada la complejidad en el entendimiento del código, es pertinente acotar las opciones que nos permiten desarrollar nuestras ideas, por eso es relevante una comprensión de la sintaxis particular de este lenguaje de programación, además del entendimiento en el ámbito de cada proceso de diseño.

Al utilizar esta herramienta, como arquitectos y programadores al mismo tiempo, tendremos que definir nuestras ideas iniciales y los caminos para llegar a ellas. Es relevante conocer las reglas del código, su compilación y ejecución que es la representación geométrica sobre la cual podemos expresar las formas y significados de nuestros objetivos iniciales. Por lo anterior, podemos deducir que la geometría bi y tridimensional es la traducción directa del código, que por su parte es la gramática base de inicio, tanto de la computadora como de las ideas humanas en el contexto de la programación.

Con el acto de realizar un dibujo o escribir un texto, el ser humano trata de representar el mundo físico o imaginario. En el dibujo, el ser humano requiere medios físicos para poder llevar a cabo el acto de representar, por ejemplo, un árbol, al cual vamos a llamar “objeto”, y un medio para poder representar, al cual llamaremos “agente”. Los agentes nos permitirán llevar a cabo el acto de representación del objeto, los cuáles podemos clasificar como herramientas físicas, tales como un lápiz, herramientas digitales como un programa ejecutando el acto de dibujar en una computadora, entre otros. Este sistema dual de clasificación para realizar el acto lo definiremos como “clase”, la cual tiene sus propios medios y formas de producción, por una parte, un lápiz necesita ser arrastrado con la mano sobre el papel para ejecutar la acción; la computadora, por su parte necesita una serie de instrucciones gráficas [botones] o textuales [comandos] para poder llevar la acción. La finalidad de estos agentes y sus respectivas clases es representar a un objeto físico, además de que ambos tienen sus propios mecanismos de producción basados en aspectos cognoscitivos de cada individuo en la comprensión particular del mundo exterior, como la memoria, la observación, el pensamiento, entre otros.

3.3.1 – Programación Orientada a Objetos

El proceso anterior puede ser reinterpretado en un lenguaje de programación computacional, en este caso basado en código, que denominaremos “Programación Orientada a Objetos”, que consiste en un paradigma que nos permitirá modelar y representar ideas mediante objetos y clases. Este paradigma consiste en el uso de conceptos de herencia, cohesión, abstracción, acoplamiento, encapsulamiento y polimorfismo; por ello expondré dos ejemplos. Para ello debemos comprender que todo lo que nos rodea son objetos, incluso nosotros mismos podemos ser considerados como objetos en movimiento. Ahora, si todos somos objetos, ¿Qué es una clase en la programación orientada a objetos?, la clase, es por decirlo de alguna manera, aquella plantilla o espacio en dónde es creado este objeto.

Como primer ejemplo mencionaré el acto de pensar en gatos. ¿Cómo podemos reconocer que un gato es en sí un gato y no un perro?, bueno, todo gato tiene ojos, nariz, boca, cuatro patas, mucho pelo, una cola, orejas y tienen garras. Esto sería la plantilla del gato y con lo mencionado anteriormente también podemos imaginarnos uno diferente. ¿Pero fuimos específicos en sus características?, cada uno de nosotros podemos pensar en un gato, pero no será el mismo. Cada uno hemos pensado en una instancia de un gato acorde a las características cognoscitivas que mencionamos. Yo pude haber pensado en un gato gordo de pelaje gris. Mientras que otra persona habrá pensado en un gato virtual de ojos grandes de color naranja. (Figura 48).

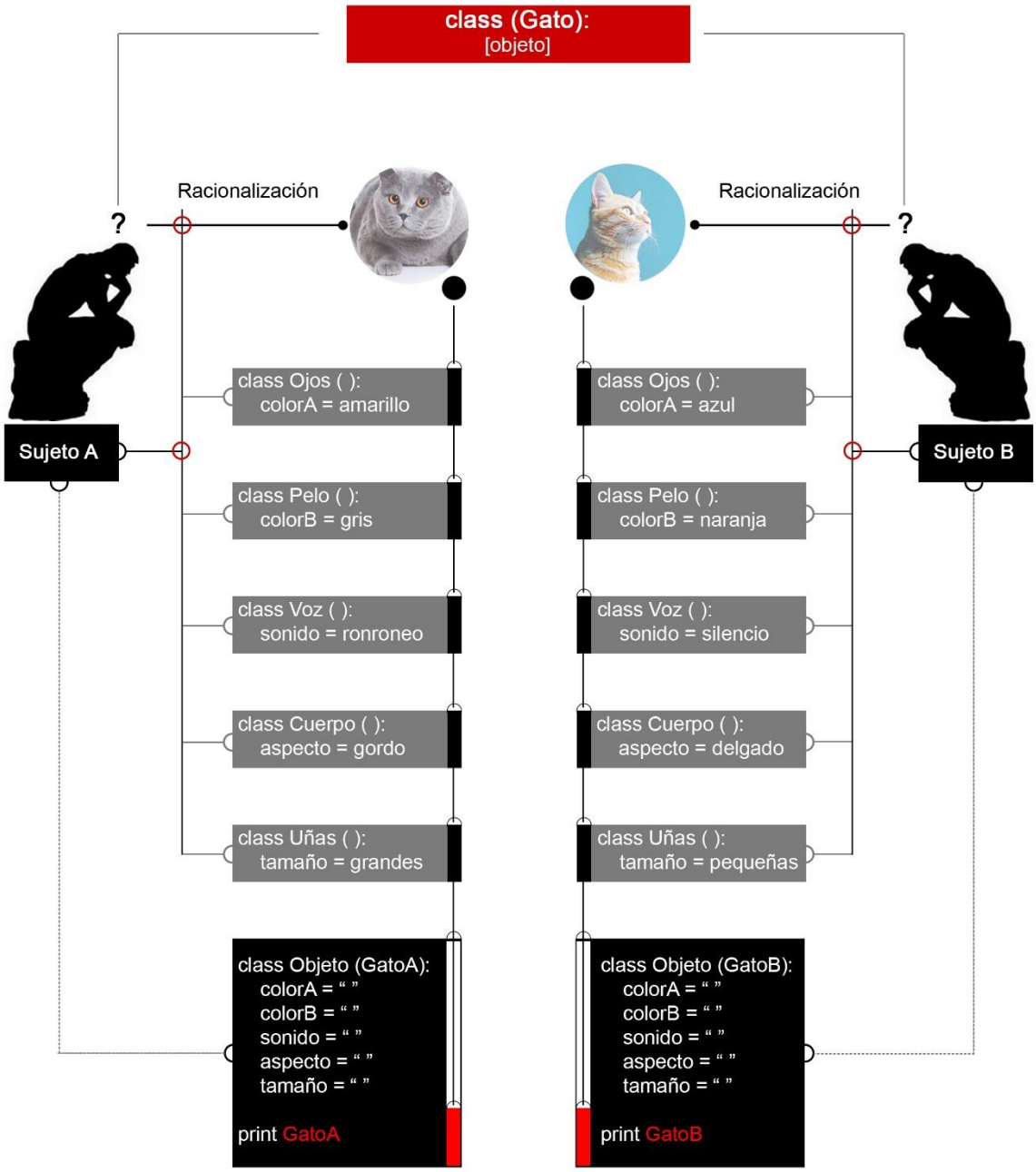


Figura 48. La compilación de un conjunto de clases definidas a partir de cualidades físicas de un objeto [Gato], formuladas por dos sujetos a partir de la racionalización. Este ejemplo muestra la relación y definición de clases para formar un conjunto.

3.3.2 – Posición y fuerzas físicas [Aceleración, Gravedad y Velocidad]

La definición de un objeto y las partes que lo conforman es el primer paso en la comprensión de una sintaxis de código en programación orientada a objetos. Un objeto, además de tener cualidades o características físicas similares o particulares, posee otro aspecto importante, la posición que ocupa en el espacio cartesiano [*x* - *y* - *z*], así como las fuerzas que dan lugar a su posición (gravedad), y otras fuerzas que pueden influir en este estado físico y que pueden alterarlo, tales como la velocidad y la aceleración, que pueden surgir del objeto o aplicarse al objeto. Lo anterior lo podemos describir como un objeto que tiene magnitudes o al que se le aplican magnitudes de fuerza.

Las magnitudes pueden ser expresadas en *Python* a través de funciones vectoriales que afectan a un objeto o a un grupo en específico de ellos, cuando aplicamos una fuerza determinada, estamos aplicando una fuerza vectorial, que tiene una dirección y una magnitud específica. La dirección podrá ser un vector en un eje determinado del plano cartesiano [*x* – *y* – *z*], o el conjunto de ejes; por otra parte, las magnitudes van a ser expresadas con valores numéricos que definirán la intensidad de las fuerzas a aplicar sobre determinado objeto. Así como las clases pueden definir las características físicas de un gato, las funciones vectoriales pueden ser incorporadas a estas clases con el fin de simular la aplicación de una fuerza a un gato. Existe, además, la posibilidad de asignar una fuerza a un objeto con el fin de que este objeto pueda desplazarse y generar geometría en base a su desplazamiento o comportamiento en el entorno de modelado tridimensional. Para poder llevar a cabo lo anterior, emplearemos una variable en *Python* denominada *"self"*, la cual nos permitirá hacer una distinción de características o cualidades de un objeto específico y asignar valores vectoriales de posición, magnitud y fuerzas propias del objeto, así como afectar por medio de estas fuerzas a clases contenidas dentro del mismo código.

Lo anterior puede resultar algo difícil de entender para arquitectos y diseñadores no tan relacionados con estos temas de física en el campo de la dinámica en objetos, pero la comprensión de estos comandos y conceptos es indispensable para la incorporación de códigos basados en lenguajes de programación textual en la producción de modelos paramétricos en la arquitectura. Por una parte, hay que comprender que la adopción de estas reglas es algo difícil de realizar al inicio, ya que involucra mucho razonamiento en un problema particular, sea este de diseño, construcción o estructura; pero por otra parte, una vez adoptado, esta serie de instrucciones será nuestra gramática o sintaxis de código que nos permitirá desarrollar y producir modelos paramétricos más personalizados y que sean capaces de representar nuestras ideas en el campo de la arquitectura y el diseño.

Una de las ventajas que tienen los arquitectos que conocen la gramática del código, en cualquier lenguaje de programación basada en texto, es la gran rapidez de ejecución de modelos paramétricos bi y tridimensionales. Hay que mencionar que la escritura de un código con el objetivo de desarrollar un modelo paramétrico en arquitectura, puede llevar algo de tiempo; pero, por otra parte, una vez desarrollado y corregidos todos los errores en el código, la producción de representaciones en respuesta a una problemática de diseño específica, será tan rápida y eficiente que solo necesitaremos presionar un botón "*play*" para iniciar y producir un modelo en respuesta a una necesidad particular. Además de poder modificar todo el diseño o proporcionar diferentes soluciones con el cambio de un número, variable, función, fórmula matemática o un comando diferente en la sintaxis del código. Hay que reconocer que la adopción de los lenguajes de programación, en este caso *Python*, puede ser muy significativa en el ahorro de tiempo y ejecución de representaciones en el

diseño arquitectónico, y que el esfuerzo inicial en el aprendizaje, será recompensado al final.

En el campo de la disciplina arquitectónica, la incorporación de modelos paramétricos basados en código textual, resulta importante la relación entre nuestras ideas iniciales en el proceso de producción en el ámbito del diseño con las oportunidades que nos brindan los lenguajes de programación como *Python*. Los arquitectos debemos entender que la adopción del código como medio de producción y representación, puede ser muy significativa, y que es, en parte, la descripción física o teórica de un objeto arquitectónico existente; con una diferencia, la de describir un conjunto a partir de partes que aún no existen, es decir, no describiremos a un objeto existente, sino que vamos a generar un objeto a partir de una descripción. Esto es algo difícil de comprender, ya que la representación de los procesos arquitectónicos o del diseño es realizada a menudo con bocetos, maquetas o un modelo tridimensional generado a partir de la modificación ya existente de geometría primitiva[29] en el programa computacional; es muy complicado describir las partes de un objeto que no conocemos y asociar esta descripción como un medio de producción y representación en el diseño arquitectónico.

Por lo anterior, resulta muy relevante la comprensión de la sintaxis de un código basado en el lenguaje de programación *Python* en los procesos de generación de modelos paramétricos computacionales; ya que los medios de representación en arquitectura son en su mayoría gráficos, el entendimiento de una serie de comandos en un conjunto de instrucciones tiene que representar un objeto que pueda ser analizado y observado en relación al código que le ha dado origen. A continuación expondré el segundo ejemplo que posee una complejidad en la sintaxis del código, y en el cual, todas las características anteriormente descritas, se encuentran incorporadas en un objeto que representará a un puente en el espacio cartesiano, basado en generación progresiva, es decir, las partes que lo conforman son generadas a partir de clases, funciones, variables, fórmulas matemáticas, variables de tiempo y en el cual la geometría final es generada a partir de la incorporación de fuerzas, tales como la aceleración, la velocidad y la posición. Dada la importancia que tiene la observación gráfica en la disciplina arquitectónica y el entendimiento de sus procesos de generación, mencionaré que en este ejemplo vamos a destruir antes de construir.

29 Las formas geométricas consideradas primitivas por su básica constitución en las partes que la conforman, se conocen también con el nombre de primitivas geométricas cuyas formas son el Círculo, el Triángulo y el Cuadrado. Las primitivas geométricas en un software 3D pueden ser editadas para conseguir formas geométricas más complejas, agregando nuevos vértices, aristas y polígonos. En cuanto a primitivas tridimensionales existen los cilindros, el tubo, el toro, la esfera y el cubo, entre otros.

3.3.3 - Puente paramétrico en generación progresiva

La incorporación del lenguaje de programación textual *Python*, por parte de los arquitectos y diseñadores nos permite la producción de modelos paramétricos basados en conceptos propios de la herramienta, la cuestión en este aspecto, es ¿Cómo incorporar estos conceptos técnicos propios del programa como un medio de producción formal en el diseño arquitectónico? En relación a esta pregunta, podemos incorporar para fines prácticos de este ejemplo, los términos de programación orientada a objetos como medio de producción de un modelo paramétrico arquitectónico.

En este caso nuestro problema es resolver la unión de dos extremos en el espacio tridimensional computacional [$x + y + z$] con geometría, con el objetivo de conectar estos dos extremos para la conformación de un objeto denominado puente. Las características antes mencionadas son el punto de partida del diseñador, que tienen gran relevancia ya que lo anterior es la incógnita y se planteará a través de una sintaxis de un lenguaje de programación y en una interfaz de programa para gráficos en 3D y *NURBS*[30] como lo es *Rhinoceros*. La terminología descrita en el anterior capítulo nos ayudará a definir los conceptos de generación por medio de la programación orientada a objetos.

Podemos definir un nombre para nuestro puente basado en la interacción que queremos que tenga en el código escrito en *Python* y la ejecución en la interfaz del programa *Rhinoceros*. En este caso, como punto de inicio vamos a definir los siguientes conceptos computacionales y físicos para tener un control del código a lo largo de su producción:

- **Nombre** [Genérico] – Puente paramétrico basado en generación progresiva.
- **Nombre** [Computacional Físico] – Comportamiento en conjunto + Puntos atractores.
- **Tipo de método** – Basado en agentes individuales y de conjunto.
- **Métodos de programación a incorporar** - código basado en *Python* incorporando:

- Alineación – Moverse en la misma dirección que los puntos en conjunto.
- Cohesión – Permanecer cerca de los agentes de conjunto e individuales.
- Separación – Evitar la colisión de los agentes en conjunto e individualmente.

- **Método de atracción** - Control de la forma y geometría final mediante puntos atractores.
- **Objetivo** - A partir del movimiento de puntos atractores, generadores en base a un tiempo definido, se producirán relaciones entre ellos por medio de líneas y se definirá la geometría final del puente a través del componente "*Pipe*" o Tubo en la sintaxis *rs [rhinoscriptsintax].*

Estos son los conceptos basados en la programación orientada a objetos a través de *Python* que vamos a utilizar para la producción de un modelo paramétrico arquitectónico, considerando solo aspectos de programación de comportamiento y geometría (figura 49).

Este proyecto fue desarrollado con la asesoría de dos arquitectos de la Universidad de Stuttgart, Alemania, Prof. Achim Menges y Msc. Arch. Ehsan Baharlou; quiénes brindaron todo el apoyo para la conceptualización, producción y realización. La ejecución en tiempo real del código de *Python* en el entorno de modelado en *Rhinoceros*, así como el código se pueden encontrar en el siguiente enlace: https://www.youtube.com/watch?v=inkj_WgBQ5g

30 Una NURBS se define como una curva racional no uniforme (acrónimo inglés de non-uniform rational B-spline) es un modelo matemático muy utilizado en la computación gráfica para generar y representar curvas y superficies. y por su grado, un conjunto de puntos de control ponderados, y un vector de nodos. Las curvas y superficies NURBS son generalizaciones de curvas B-splines y curvas de Bézier, así como de superficies, siendo su diferencia principal la ponderación de los puntos de control que hacen a las curvas NURBS racionales (las curvas B-splines racionales no uniformes son un caso especial de las curvas B-splines racionales).

Puente

```python
import rhinoscriptsyntax as rs
import Rhino as rc
import time
import math
import scriptcontext as rc
import Rhino.Geometry as rg
from scriptcontext import escape_test
from random import *
```

Importación de librerías en Python, para utilizar los componentes y variables matemáticas, numéricas y geométricas de cada una de ellas, además de incorporar sintaxis de tiempo y generación random de geometría de apoyo.

```python
rectX = 600
rectY = 600
```

Definición y control de los límites en los ejes x + y através de las variables rectX y rectY, con el fin de establecer un límite en la producción de la geometría inicial para el puente.

```python
class Runner:
    def __init__(self, p, v):
        self.p = p
        self.v = v
        self.a = rs.VectorCreate( (0,0,0),(0,0,0) )
        self.ptList = []
```

La clase denominada "Runner" corresponde a la totalidad del sistema de generación de tiempo, geométrico y de simulación física. Con la condición de "self" asignamos fuerzas a cada uno de los objetos generadores.

```python
    def ptRun(self):
        self.v = rs.VectorAdd(self.v, self.a)
        v = rs.VectorLength(self.v)
        if v > 25:
            self.v = rs.VectorScale(rs.VectorUnitize(self.v), 25)
        self.p = rs.VectorAdd(self.p, self.v)
        self.a = rs.VectorCreate( (0,0,0),(0,0,0) )

        self.ptList.append(self.p)
```

Las variables del primer grupo determinarán los límites del puente en el plano bidimensional, además de definir las fuerzas iniciales y generales del sistema. El segundo grupo determinará la separación de los puntos y vectores iniciales.

```python
    def flock(self):
        self.separate(4.0)
        self.cohesion(0.001)
        self.align(0.3)

        self.attractor(0.5)
```

La fuerza de separación definirá los espacios entre cada uno de los puentos iniciales de apoyo para la geometría final, la cohesión y alineación definiran las relaciones de los puntos a través de líneas random.

```python
    def align(self, mag):

        steer = rs.VectorCreate( (0,0,0) , (0,0,0) )
        count = 0

        for i in pts:
            distance = rs.Distance(i.p, self.p)
            if distance > 0 and distance < 50:
                steer = rs.VectorAdd(steer, i.v)
                count += 1

        if count>0:
            steer = rs.VectorScale(steer, 1.0/count)

        steer = rs.VectorScale(steer, mag)
        self.a = rs.VectorAdd(self.a, steer)
```

Este grupo de funciones determinará la posición de los puntos secundarios generadores a partir de vectores basados en fuerzas. Estos vectores serán el camino que seguirán los puntos finales, así como las líneas en relación a la dirección de los vectores en movimiento.

A partir de las fuerza de alineación "self", se establece una relación vectorial, con el fin de dar magnitud y dirección a l sistema generador inicial de puntos y líneas en el espacio de modelado tridimensional.

```python
    def cohesion(self, mag):

        sum = rs.VectorCreate( (0,0,0) , (0,0,0) )
        count = 0

        for i in pts:
            distance = rs.Distance(i.p, self.p)
            if distance > 0 and distance < 70:
                sum = rs.VectorAdd(sum, i.p)
                count += 1

        if count>0:
            sum = rs.VectorScale(sum, 1.0/count)

        steer = rs.VectorSubtract(sum, self.p)
        steer = rs.VectorScale(steer, mag)
        self.a = rs.VectorAdd(self.a, steer)
```

Este grupo de funciones definirá la relación de puntos a través de fuerzas de cohesión, atracción y separación; basado en la generación de puntos random y la unión de líneas a lo largo de sistema de dirección y generación anterior.

```python
    def separate(self, mag):

        steer = rs.VectorCreate( (0,0,0) , (0,0,0) )
        count = 0

        for i in pts:
            distance = rs.Distance(i.p, self.p)
            if distance > 0 and distance < 40:
                diff = rs.VectorSubtract(self.p, i.p)
                diff = rs.VectorUnitize(diff)
                diff = rs.VectorScale(diff, 1.0/distance)

                steer = rs.VectorAdd(steer , diff)
                count += 1

        if count>0:
            steer = rs.VectorScale(steer, 1.0/count)

        steer = rs.VectorScale(steer, mag)
        self.a = rs.VectorAdd(self.a, steer)
```

La fuerza de separación "self" adquiere magnitud y fuerza mediante una adición vectorial, lo que define la relación de puntos a través de líneas de soporte en las generatrices del sistema inicial.

Estos dos grupos definirán la magnitud y dirección de las fuerzas en el sistema de vectores iniciales; así como la longitud y dirección de propagación de puntos finales para la posterior relación random de líneas, tanto en el sentido transversal como longitudinal.

```python
    def attractor(self, mag):

        attrPt = rs.VectorCreate((-500,-400,0) , (0,0,0))
        steer = rs.VectorCreate( (0,0,0) , (0,0,0) )
        diff = rs.VectorSubtract( attrPt, self.p )
        diff = rs.VectorUnitize(diff)

        steer = rs.VectorAdd(steer , diff)
        steer = rs.VectorScale(steer, mag)

        self.a = rs.VectorAdd(self.a, steer)
```

La fuerza de atracción "self" incorpora ahora magnitud y dirección en los puntos de generación, lo que permitirá definir el inicio y final del puente en relación a los puntos que se generarán por el movimiento.

```python
    def drawLines(self):
        for i in pts:
            distance = rs.Distance(i.p, self.p)
            if distance < 50 and distance > 0:
                pt1 = rg.Point3d(i.p[0], i.p[1], i.p[2])
                pt2 = rg.Point3d(self.p[0], self.p[1], self.p[2])
                lns.append(rs.AddLine(pt1, pt2))
```

Estos dos grupos definen los puntos en el espacio tridimensional a partir de las relaciones vectoriales y de magnitud de fuerzas anteriormente descritas; así como las líneas base en el espacio tridimensional.

```python
    def drawPt(self):
        pt = rs.AddPoint(self.p[0], self.p[1], self.p[2])
        return pt
```

```python
def setup():
    global pts
    pts = []
    global lns
    lns = []
    numAG = 50

    for i in range(numAG):
        p = rs.VectorCreate( rs.AddPoint( 100*math.cos(i*2*math.pi/numAG), 100*math.sin(i*2*math.pi/numAG), 0) , rs.AddPoint(0,0,0) )
        v = rs.VectorCreate(   rs.AddPoint( -randint(0,36),-randint(25,50),randint(-3,30)   )  ,  rs.AddPoint(0,0,0   )  )
        run1 = Runner(p, v)
        pts.append(run1)
```

```python
def run():
    pos = []
    vec = []

    for i in pts:
        pos.append(i.drawPt())
        vec.append(i.v)
    for i in pts:
        i.flock()
        i.ptRun()
        if t > 10 and t%3== 1:
            i.drawLines()
```

En base a una relación matemática inicial de coseno en los puntos iniciales y finales, y una relación de seno en la altura, se define la propagación de puntos finales, así como las líneas de generación geométrica final para dar lugar a la forma en los ejes *x + y + z* del sistema.

```python
def drawTime():
    FPS = 30
    last_time = time.time()
    global t
    t = 0
    curves = []
    # ciclo de curvas random...
    while True:
        # dibujar animacion
        t += 1
        # pausa para que la animacion se ejecute a 30 FPS
        new_time = time.time()
        # milisegundos para la ejecucion de la animacipn
        # dividir entre 1000.0 desde el comando time.sleep() utilizar segundos
        sleep_time = ((1000.0 / FPS) - (new_time - last_time)) / 1000.0
        if sleep_time > 0:
            time.sleep(sleep_time)
        last_time = new_time

        run()

        print t

        if t > 108:
            for k in pts:
                curves.append(rs.AddCurve(k.ptList))

            rs.EnableRedraw(False)
            for crv in curves:
                rs.AddPipe(crv, [0,0.3, 1], [6,2,1], cap=2)

            for ln in lns:
                rs.AddPipe(ln, 0, 1, cap=2)
            rs.EnableRedraw(True)
            break

        escape_test()
```

Con el fin de obtener el modelo final del puente, se establece una función que determinará el tiempo de ejecución de la representación del modelo final. Además de "traducir" el tiempo por medio de curvas generadoras.

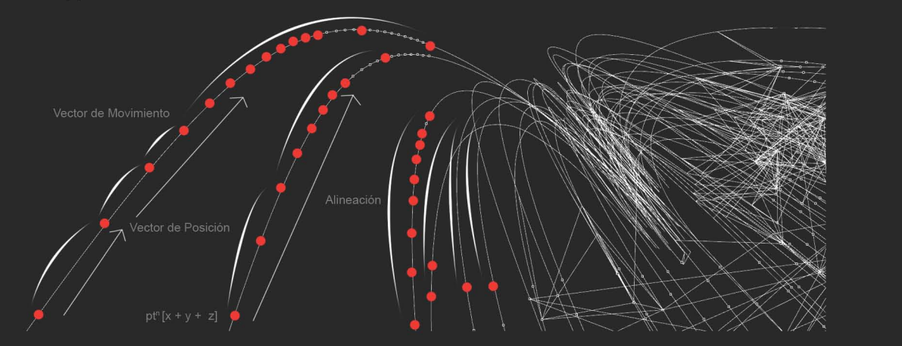
Este grupo de funciones y variables define la geometría final de puntos y líneas en el espacio tridimensional, por medio de auto - eliminación de instancias de referencia, así como el tiempo en 108 FPS [Cuadros por segundo], representando cada paso de generación geométrica.

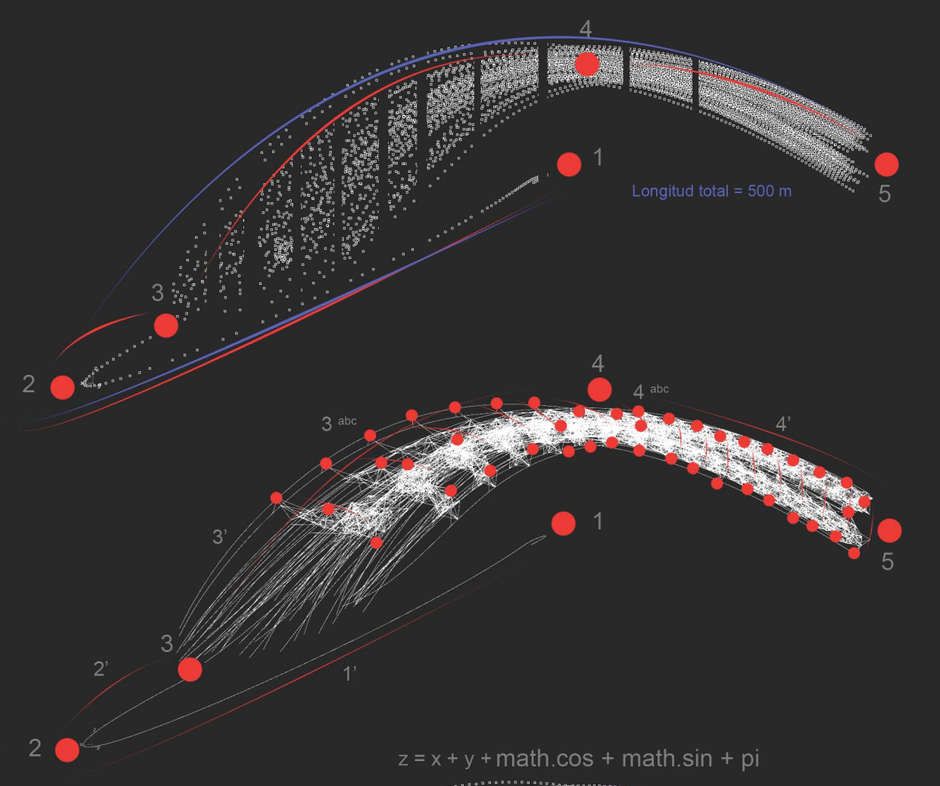
Al definir las curvas finales de generación con ".append", los puntos y las líneas también deben de relacionarse con .append, para evitar clonación de elementos geométricos y poder hacer referencias individuales.

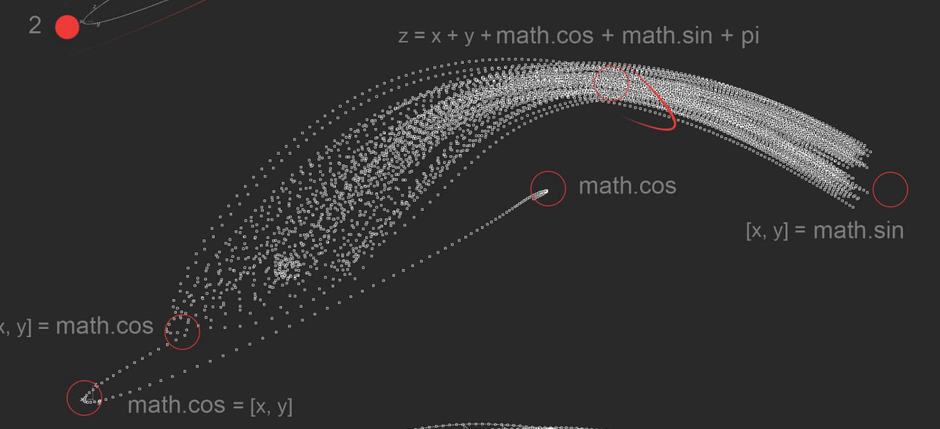
```python
def main():
    setup()
    drawTime()
```

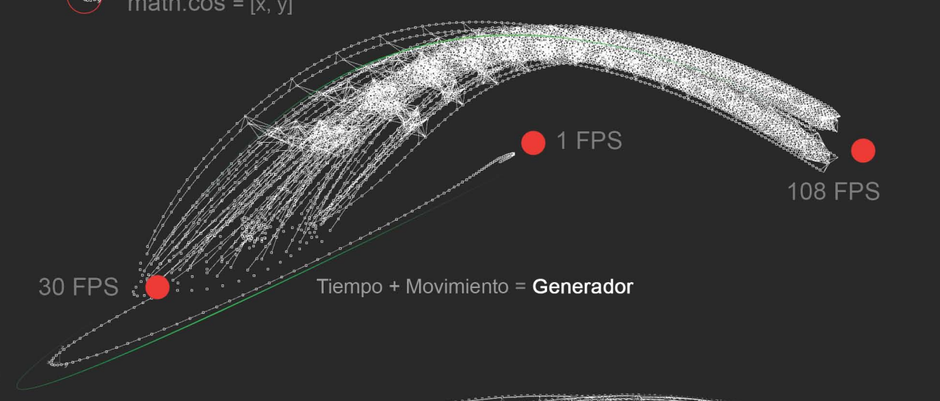
En este grupo se define la clase completa en el sistema y su detención una vez realizada la compilación y la ejecución del código. El ciclo es completado y la geometría final representada por medio del componente "*Pipe*" a través de las referencias de líneas anteriores.

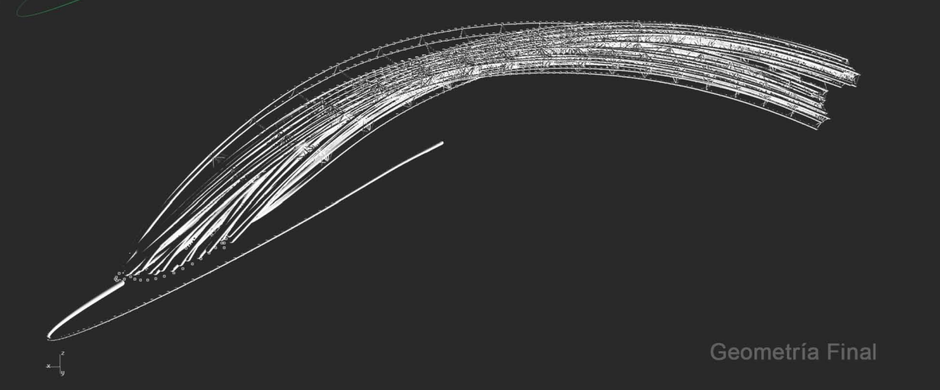
```python
if __name__ == "__main__":
    main()
```

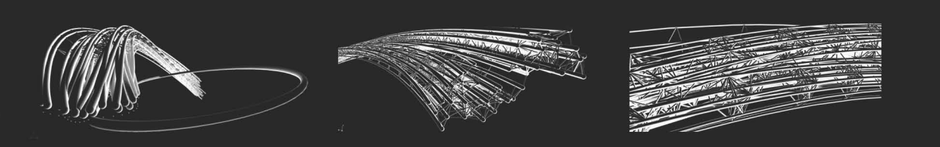












Alan Roo

Figura 49

3.4 – Conclusiones

La escritura de código basado en el lenguaje de programación como *Python* puede demorar mucho tiempo si no se tienen los conceptos de base iniciales en la sintaxis. La definición de conceptos en los lenguajes de programación es muy importante antes de dar solución a un problema general o particular arquitectónico. Los errores en la sintaxis del código toman un aspecto relevante, ya que pueden incrementar los tiempos de producción debido a la corrección en las líneas del código, afortunadamente la escritura se llevó a cabo en computadoras con un formato *UNICODE* similar en los lenguajes alemán, español e inglés, lo que facilitó de alguna manera la generación de la sintaxis y su posterior ejecución.

Existen muchas razones para la implementación del código basado en lenguajes de programación diferentes a *Python*, por una parte, este lenguaje nos permite trabajar en conjunto con una interfaz de modelado tridimensional ya incorporado en el entorno del programa como *Rhinoceros 5.0*. Además de las razones anteriormente descritas y desarrolladas a lo largo de este capítulo, ahora comprendemos que podemos generar una geometría más compleja y organizada en relación a variables y parámetros escritos en un código computacional, esto nos ahorra bastante tiempo en cuanto a modificaciones y correcciones en geometrías duplicadas o erróneas que se producen durante las fases de desarrollo de un modelo paramétrico tridimensional en la disciplina arquitectónica. Podemos tener mayor control de la geometría representativa del sistema final, tendremos un mejor sistema de construcción en cuanto a geometría, un mayor control en la organización de los componentes numéricos y geométricos, además de obtener una mayor retroalimentación por parte de colegas al momento de producir un código para un modelo paramétrico. Por otra parte, el depurador nos brinda ayuda importante al momento de corregir errores en la sintaxis de ejecución, además de proporcionarnos la representación de los errores y la ubicación en las líneas del código, así como una posible solución a tal error.

La cooperación y retroalimentación no siempre es posible por parte de colegas arquitectos o diseñadores que desarrollan modelos paramétricos basados en código. Al elegir un medio de producción geométrico y de datos representativo, los usuarios siempre tienen que realizar una elección, optar por los entornos de trabajo gráficos aceptando los problemas de gran escala a los que se pueden enfrentar, o implementar un medio de producción basado en un lenguaje de programación textual que implicará un tiempo mayor al inicio pero que al final será recompensado por la simple modificación de un comando o un parámetro numérico que afectará al modelo en conjunto o una de sus partes específicas. Muchas personas, incluyendo Ivan Sutherland (1963, 8), Bret Victor (2012) y Nigel Cross (2011, 11), han sugerido que la interactividad en la escritura del texto puede producir una memoria a corto y largo plazo, que propicia el “no olvido” por el simple hecho de escribir y repetir texto en un código con un ciclo de ejecución específico de cada problema. Su intuición es reforzada por medio de estudios cognoscitivos que demuestran que un usuario necesita repetir una sintaxis de un código para aprenderla en sus ciclos y procesos de ejecución específicos, así como de la necesidad de una constante retroalimentación con compañeros o en foros de internet especializados, y que algunos usuarios sufren cuando no han mantenido una disciplina en las actualizaciones, la concatenación lógica, organización y repetición del código, y cuando la retroalimentación es escasa (Erhan, 2009; Nasirova 2011). En otras disciplinas del diseño, los diseñadores tienen un gran rango de acceso en entornos de programación textual, todavía, para arquitectos, la interacción y utilización de un entorno de programación textual con el objetivo de dar solución a problemáticas espaciales se deben desarrollar a fondo, y no como un simple medio técnico de producción geométrica en modelos paramétricos.

En este capítulo he demostrado como el lenguaje de programación *Python* en el entorno de modelado tridimensional *Rhinoceros 5.0*, puede ser un medio de producción interactivo a través de un código computacional, por medio de la incorporación de parámetros de fuerza basados en vectores y librerías geométricas que pueden ser importadas a la sintaxis inicial en el código y que pueden ser una herramienta que ayude a los arquitectos a solucionar problemáticas generales y particulares en cuanto a generación geométrica aplicada al diseño arquitectónico. Así como existen métodos de escritura de código en la programación interactiva – los cuáles son utilizados para propiciar un cálculo intenso para realizar operaciones matemáticas y geométricas – *Python* nos permite la producción, modificación y representación de geometría en tiempo real a la de su compilación y ejecución en el entorno tridimensional de *Rhinoceros 5.0*. Con el objetivo de generar una gráfica *DAG*, *Python* se puede emplear una base en programación orientada a objetos, que es paradigmáticamente algo diferente a otros lenguajes de programación, pero similar en tiempos de construcción, tamaño del código y funcionalidad. De manera diferente a otros entornos de programación textual, *Python* también incorpora un gran número de características que los ingenieros de software han desarrollado para cumplir los ciclos de compilación, ejecución y producción con el fin de hacerlos más interactivos, como la función de auto – completar en el editor, la importación de una gran cantidad de librerías y el depurador de localización de errores en la sintaxis del código, además de la simple e interactiva incorporación del editor de *Python* en el entorno de modelado *Rhinoceros 5.0*.

Al utilizar el lenguaje de programación *Python* como medio de producción de una gráfica *DAG*, esta herramienta reduce la latencia entre la escritura del código y la visualización de los resultados geométricos. En ciertos casos, la reducción de la latencia transformará una tarea en la cual los arquitectos no requieren retroalimentación, en arquitectos que puedan proporcionar o solicitar retroalimentación específica para un problema en particular en un proceso particular de diseño arquitectónico. Como resultado, escribir y modificar la sintaxis en un código de *Python*, puede llegar a ser parecido a la modificación, adición o eliminación de un componente en una gráfica *DAG*. Sin embargo, la incorporación de código como medio de producción en los procesos de diseño, demuestra que los arquitectos y diseñadores pueden reducir de manera significativa la latencia por medio de la interacción entre la extensión del código, la autocorrección a través del depurador y la reutilización del código. En el segundo ejemplo, correspondiente al puente, la reducción en la latencia hace una importante contribución en relación a la identificación y eliminación de cualquier problema de construcción geométrico, así como en la generación a partir de fuerzas basadas en vectores en la representación geométrica final del modelo paramétrico. Esto indica que la calidad de la flexibilidad en un modelo paramétrico – como la latencia en el modelo – puede tener un impacto discernible como medio de producción en el diseño arquitectónico. Estas cualidades pueden ser diseñadas por sí mismas a través de la composición de un modelo paramétrico o a través de la selección de un entorno de un lenguaje de programación.

A través del uso del lenguaje de programación *Python*, conceptos teóricos - técnico computacionales y físicos, se demuestra como el conocimiento básico de la ingeniería del software puede ofrecer muchas alternativas en relación a los diversos lenguajes de programación basadas en un código textual, y que pueden ser sincronizados con atributos particulares en el modelado paramétrico en relación a los procesos del diseño arquitectónico. *Python* es solo una manifestación de un conocimiento específico y cabe mencionar que existen muchas posibilidades, así como diversas opciones en entornos de programación gráficos y textuales, que podrán ser implementados en el futuro por arquitectos como medio de producción conceptual, técnico y análisis ambiental – espacial.

Capítulo 4 – Caso de estudio Lógico – Gráfico

Sistema de crecimiento en ramificación fractal del Coral Acropora

"Un nuevo lenguaje requiere una nueva técnica"
- Philip Glass

"Los materiales son el nuevo software"
- A J Jacobs

4.1 – La crisis de la forma

A lo largo de la gran trayectoria en la historia del hombre, el diseño y la producción de artefactos ha estado caracterizada por la creciente separación de la "*producción formal*" desde sus fundamentos naturales o biológicos en condiciones de material[31]. En contraste con la producción artesanal (Sennett, 2008), en la cual los materiales y la forma están relacionados de manera intrínseca con la tradición del "*hacer*". El diseño y la producción moderna han estado involucrados históricamente muy lejos de esta integración en relación complementaria de la "*producción formal*" considerándolo como un proceso independiente, propio de un conocimiento aislado de propiedades de los materiales. Al menos desde el Renacimiento, con la emergencia de teorías arquitectónicas, la "*producción formal*", ha sido una generación de formas inspiradas por las diferentes teorías, y ha llegado a ser un cuerpo autónomo de conocimiento[32]. Dentro del diseño arquitectónico e industrial, los campos más sensibles a la producción basados en culturas particulares, el aspecto formal ha crecido tanto en la eminencia y presencia temporal en los procesos del diseño al punto de que la condición de la *forma precede a la materialización* ha llegado a ser normativa y virtualmente intuitiva en la cultura contemporánea del diseño. Con excepción de unos pocos casos en el diseño contemporáneo, el pragmatismo y la degradación del aspecto material se ha convertido en un axioma. La materialidad[33] es, dentro de la lógica de la tradición modernista, un agente secundario del aspecto formal.

La Revolución Industrial mantuvo abierta la puerta a la manufactura basada en la máquina, además de la producción en masa. La producción de la forma es en la actualidad concebida y creada a partir de la automatización industrial; los aspectos dominantes de estos procesos asumen a la funcionalidad y a la ontología como los estándares principales (Jencks, 1984). La fórmula ha sido seguir a la función y al ornamento de modo endémico a la cultura artesanal, lo que es un error (Frampton y Futagawa, 1980). Los valores promovidos por los "antiguos" oficios, ha pronunciado la integración de la sustancia material y los métodos de construcción, una vez dentro de los límites de conocimientos del artesano, lo anterior fue abandonado, mientras que, en su lugar, surgió una producción de diseño basado en los valores de la producción en masa.

31 Con material me refiero al significado primario ontológico y en relación a las ciencias naturales, en su concepto de un conglomerado de materia o masa, además de un elemento que puede transformarse y agruparse en conjunto. El material puede ser simple o complejo, además de heterogéneo.

32 Se hace referencia al Renacimiento en el empleo de formas de la naturaleza a partir de su significado literal por algunos arquitectos de la época. Y a la no consideración de las propiedades propias de los materiales de la naturaleza, como la estructura y materia que los conforma, sino como un medio de representación meramente formal y de construcción con materiales modificados por el hombre, y no construidos por la propia naturaleza a partir de sus componentes físicos y químicos.

33 La materialidad es la característica principal de la cosa que es material, pertenece al mundo físico, posee propiedades químicas particulares, relaciones en conjunto y se puede percibir por los sentidos del ser humano.

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Los tipos de construcción con fundamento en la rapidez, lo barato, lo repetitivo, las partes y su modulación eran sinónimo de progreso del sueño visionario de Henry Ford. En el lado de la victoria de los medios de producción industrial, parecía como si la proximidad del diseño en relación a los oficios y sus expresiones particulares, comúnmente denominadas "vernáculas" eran ahora condenadas a la pérdida y al olvido, y con ello, a una gran separación alejada del contexto íntimo de conocimientos basados en la tecnología de los nuevos materiales. Con el tiempo, este enfoque no material y la automatización en la construcción del diseño fue reforzada bajo el mando del diseño asistido por computadora.

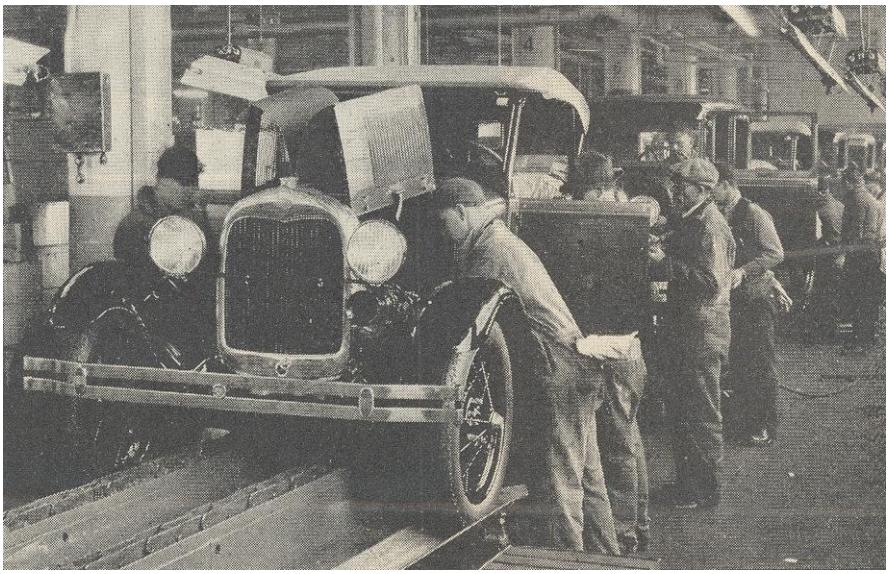


Figura 50. Fábrica de ensamblaje de autos de Ford (modelo A), se convirtieron en un símbolo de la producción en masa efectiva. La "eficiencia" disminuyó el precio de los coches y permitió a Henry Ford aumentar los salarios de los trabajadores. Por lo tanto, los trabajadores comunes podrían comprar y seguir comprando sus propios autos.

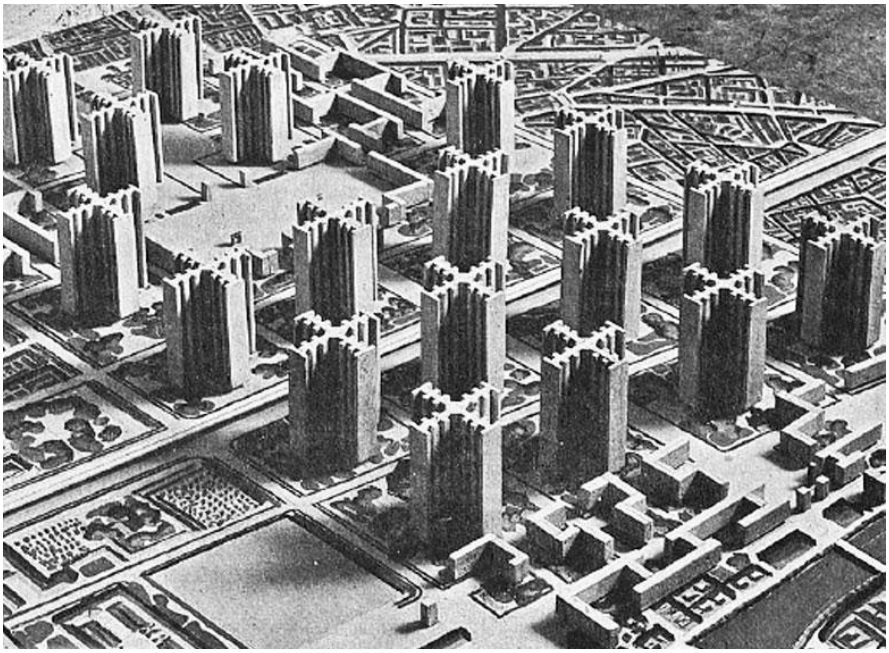


Figura 51. Maqueta del conjunto de la Ciudad Radiante (*La Ville Radieuse*, en francés), propuesta por Le Corbusier y preponderando la incorporación de métodos de producción y construcción estándar y mediante el ensamblaje de partes individuales para la conformación de un conjunto unitario y general.

La Revolución Digital, ha marcado un cambio radical desde la tecnología analógica a la digital, ha transformado el pizarrón de dibujo en una pizarra digital. La forma, al parecer, ahora se encuentra divorciada completamente de la realidad física de su manifestación. Por

supuesto, estos nuevos espacios para la producción del diseño proporcionaron mucha libertad y deliberación en términos de expresión formal, y han también ampliado el límite entre la forma y la materia, además de hacer una separación jerárquica y secuencial de los procesos de modelado, análisis y una producción infinitamente más pronunciada.

La implementación y la amplia adopción de herramientas de diseño computacionales han mejorado la práctica arquitectónica en términos de producción y eficiencia de tiempo; y desde principios de los años noventa, ha motivado un renacimiento de los proyectos formalistas en la disciplina arquitectónica, las formas geométricas complejas se convirtieron en emblemas asociadas a la creatividad de diversos autores, a través de los entornos de diseño digital y mediante el apoyo del diseño en base a la generación formal basada en geometrías complejas con una fundamentación meramente técnica. Esta orientación al diseño meramente formal y geométrico, también ha propiciado un medio de producción enfocado al diseño de “formas libres” y en la arquitectura, este fenómeno ha propiciado, junto con sus tecnologías de apoyo, un cambio muy amplio referido a la “personalización masiva” de la forma, dando lugar a la “no estandarización”.

Actualmente, tal vez bajo los imperativos del creciente reconocimiento de los fracasos ecológicos del diseño arquitectónico e industrial moderno, la cultura del diseño está siendo testigo de una nueva materialidad. En las últimas décadas, tanto en el diseño arquitectónico como el industrial, un nuevo conjunto de conocimientos está emergiendo a través de diversos documentos y experimentos en investigación industrial e iniciativas académicas (Mitchell y McCullough, 1991). Los ejemplos del creciente interés en el potencial tecnológico en la innovación de uso de los materiales como una fuente de generación conceptual y técnica aplicados al diseño, son enfocados al desarrollo de biomateriales y materiales sensibles al entorno climático, materiales compuestos y producidos mediante fabricación computacional y robótica, siempre preponderando el reciclamiento continúo. Con la creciente importancia de la materialización híbrida y medida través del comportamiento natural – artificial, las nuevas fronteras de la ciencia de los materiales, enfocada al diseño, están apoyando y propiciando el surgimiento de nuevas perspectivas en la arquitectura y el diseño industrial. Así, el papel de la investigación en el diseño digital, como el entorno favorable para la experimentación de la transformación a una nueva era de diseño basada en los materiales, puede ser relacionado con vanguardias en diseño a partir de la investigación y experimentación en el diseño computacional. Nos encontramos en la cúspide de la aparición y desarrollo de un nuevo campo de investigación en la interacción de la ciencia de los materiales y el diseño computacional.

Acompañado a este surgimiento, que podría denominarse “nueva materialidad” (Oxman, 2010), este fenómeno puede traer implicaciones en la interacción del desarrollo contemporáneo de la cultura del diseño científico, tecnológico e industrial, que reconoce la relevancia en la relación a un cuerpo único de conocimientos aplicados a procesos de diseño particulares. Las investigaciones innovadoras de algunos pioneros en la materia ahora no tan mencionados u olvidados en su totalidad, han propiciado una búsqueda formal basada en las propiedades y explotación de los materiales y su comportamiento como una nueva fuente de generación computacional y formal (Otto, Glaeser 1982; Otto Herzog 1990; Otto, Rasch 1995), que en la actualidad adquieren una gran relevancia. Los procesos de desarrollo en la biología y morfología, en particular las áreas de caracterización de materiales y sus procesos de desarrollo son investigados como fuentes de principios para el diseño de estructuras en nuevos materiales (Thom, 1975; Ruse y Hull, 1989). De manera más específica, la noción de imitar el comportamiento físico de estructuras naturales en la síntesis de nuevos materiales y procesos ha generado un enorme interés, tanto en la

comunidad científica como en la del diseño (figura 54). A partir de estas redes de fuente y recursos de conocimientos, se ha desarrollado una nueva sensibilidad referida a las escalas de los productos y a la formulación de procesos y metodologías particulares de construcción. Esta nueva sensibilidad lleva consigo ciertas consecuencias ecológicas potenciales y los valores intrínsecos pueden ser de una importancia mayor en cualquiera de las implementaciones futuras basadas en el diseño material computacional.

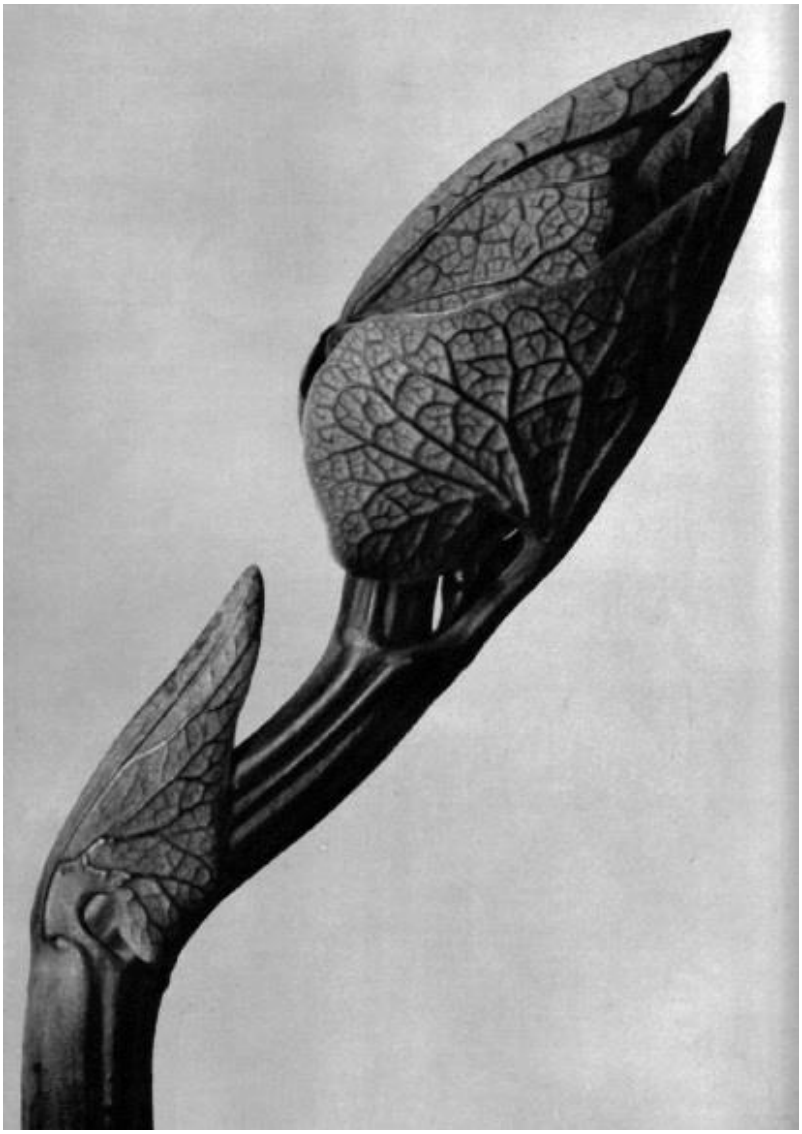


Figura 52. *Aristolochia Clematitis* fotografía de Karl Blossfeldt. La naturaleza ha alcanzado "altos niveles" de personalización morfológica en respuesta a su ambiente, mediante la integración de varios criterios de adaptación y crecimiento. Esta condición es particularmente evidente en el reino vegetal, en dónde las estructuras de fibra son físicamente estructuradas y espacialmente organizadas para combinar funciones estructurales en conjunto (como la estabilidad) con funciones ambientales y fisiológicas (como la absorción y transferencia de agua y calor).

El diseño a partir de la materialidad y la imitación del comportamiento natural a través de lo computacional y digital está fundamentado bajo los criterios del cálculo de las propiedades físicas de los materiales y condiciones ambientales dentro de los procesos del diseño que permite la generación geométrica computacional. Las siguientes anotaciones resumen la evolución de la racionalidad científica y la investigación de este documento, aplicados a los casos de estudio a desarrollar, así como los componentes, métodos y resultados de investigaciones realizadas. Como un medio de integración de precedentes de diversos campos de estudio, este campo unificará una investigación aplicada al diseño arquitectónico; así como los recursos teóricos que darán lugar a las propuestas en base al

diseño computacional no solamente como una herramienta de producción y conclusiones geométricas, basado en la incorporación de las propiedades de los materiales que conforman a ciertos organismos biológicos presentes en la naturaleza y condiciones ambientales en de los procesos de simulación computacionales de generación aplicados al diseño.

- ***El Camino de la Naturaleza o el Diseño Natural***, se relacionarán a través de diferentes puntos de vista de la morfología biomimética, el auto – crecimiento y la auto – generación. Los tres conceptos muestran los procesos de diseño de la naturaleza, así como sus principios morfológicos y cuyos conocimientos son relevantes para la generación de la forma material.

- ***La Ciencia e Ingeniería de los Materiales***, que ofrece precedentes y descripciones de los materiales en cuanto a morfología y estructura. Tales precedentes, sus descripciones formales y la exactitud y rigor científico proporcionará las bases para algunos métodos que se requieren para el desarrollo de los procesos computacionales basados en los materiales propios de la naturaleza.

- ***Diseño Computacional***, el cual ofrece los modelos computacionales lógicos y formales basados en la representación de geometría generada a partir de logaritmos gráficos y textuales, y que pueden ser empleados en la investigación de los casos de estudio, con el objetivo de integrar los procedimientos analíticos en relación a la producción de generación formal, y en relación a un acercamiento al diseño que incorpora propiedades particulares de materiales y su comportamientos de crecimiento y físicos mediante la imitación – simulación computacional como controlador potencial de los procesos de generación formal.

La única integración de estos tres cuerpos de conocimiento científico y de diseño contribuirá al desarrollo de los casos de estudio en este documento. Los conceptos anteriores constituyen los fundamentos teóricos y los recursos metodológicos a explicar y desarrollar en el primer caso de estudio, propuesta de un acercamiento a la generación formal en el diseño en general, y para el diseño arquitectónico como medio de producción geométrico, demostrando el potencial de investigación para una gradual y profunda transformación de los contenidos y métodos a través del diseño computacional. La investigación asumirá un trasfondo particular de conocimientos en problemas particulares del diseño arquitectónico, en una era dominada de tecnologías de producción en masa.

Los procesos de cambio a las impresiones de los problemas que evidenciaban una verdad particular, por medio de los planteamientos son, por supuesto, ventajas recíprocas a las conclusiones y efectos de este trabajo. Al reconocer y hacer repetidamente referencia a estas páginas, podemos reflexionar acerca de un fenómeno de "*crisis formal*" en la arquitectura y en el diseño. La cultura alternativa a este trabajo puede promover un diseño natural y una nueva materialidad como un antídoto potencial de la "*crisis del medio ambiente humano*" que el mismo ser humano propició. Este potencial radica en el énfasis exhaustivo de la importancia y en el entendimiento de conceptos ecológicos de este trabajo, en la voluntad particular del autor en abordar algunas de las raíces y condiciones dañinas de los residuos arquitectónicos, mediante la revisión, lectura y producción de documentos relacionados con diseño arquitectónico como medio de generación, producción y no como mera propaganda formal.

4.2 - El camino de la naturaleza

Fundamentos y desarrollo de un fenómeno cultural de diseño

"En las invenciones de la naturaleza, en ella, nada sobra y nada hace falta"
-Leonardo da Vinci

4.2.1 - ¿Por qué la naturaleza?

Porque la naturaleza es evidentemente sostenible. Sus retos se han resuelto dando lugar a soluciones adaptables y de larga duración, además de un rendimiento máximo utilizando un mínimo de recursos. Sorprendentemente, las "invenciones" propias de la naturaleza han inspirado los logros en cualquier disciplina del ser humano y han llevado a la producción de materiales extremadamente eficaces y eficientes, estructuras, métodos, herramientas, mecanismos y sistemas diseñados a partir de ella. Los conocimientos obtenidos a partir de del estudio de los materiales propios de la naturaleza son importantes, no solo desde la perspectiva científica, sino también en el contexto de su aplicación en entornos artificiales o sintéticos en el diseño.

En este primer caso de estudio revisaremos y discutiremos las principales características de una forma presente en la naturaleza. Esta es una misión muy complicada cuando se lleva a cabo desde el punto de vista del diseño, y en particular, del diseño arquitectónico, ya que el fenómeno de crecimiento, por ejemplo, ocupa vastas implicaciones al considerar los orígenes de la forma: "¿El edificio está [creciendo]?". A pesar de lo anterior, aspiro a examinar la producción y no la creación de la naturaleza, y demostrar que en los caminos vastos de la naturaleza siempre habrá una relación directa entre materia y energía, entre la forma y en entorno, y entre el órgano y la función. Cuando consideramos las formas y materiales naturales desde una perspectiva del diseño, es importante tener en cuenta las estrategias y los mecanismos por los que la naturaleza hace la distribución de los materiales y energía, así como sus propiedades particulares referidas a sus funciones mientras se mantiene en forma estable con las demás formas de vida en el planeta Tierra.

4.2.2 – Principios

4.2.2.1 – Inventario Mínimo – Máxima Diversidad

En el libro *Structure in Nature Is a Strategy for Design* del diseñador Peter Pearce presenta un concepto que hace alusión a los caminos sostenibles propios de la naturaleza. El autor hace referencia a el concepto de *Minimum Inventory – Maximum Diversity* (Pearce, 1981) que se encuentra presente en todos los sistemas naturales. Tales sistemas son definidos por Pearce como inventarios mínimos de tipos de componentes, un conjunto de partes, junto con reglas que combinan a estos componentes. Los sistemas de ajuste son descritos por sus componentes de maximización variable, aunque relacionados genéticamente con formas estructurales. Por otra parte, en sistemas exitosos, las reglas de ensamblaje y los componentes físicos propios se relacionan orgánicamente, de tal manera que las reglas son vistas crecer externamente a partir de sus partes, y las partes crecen fuera de reglas.

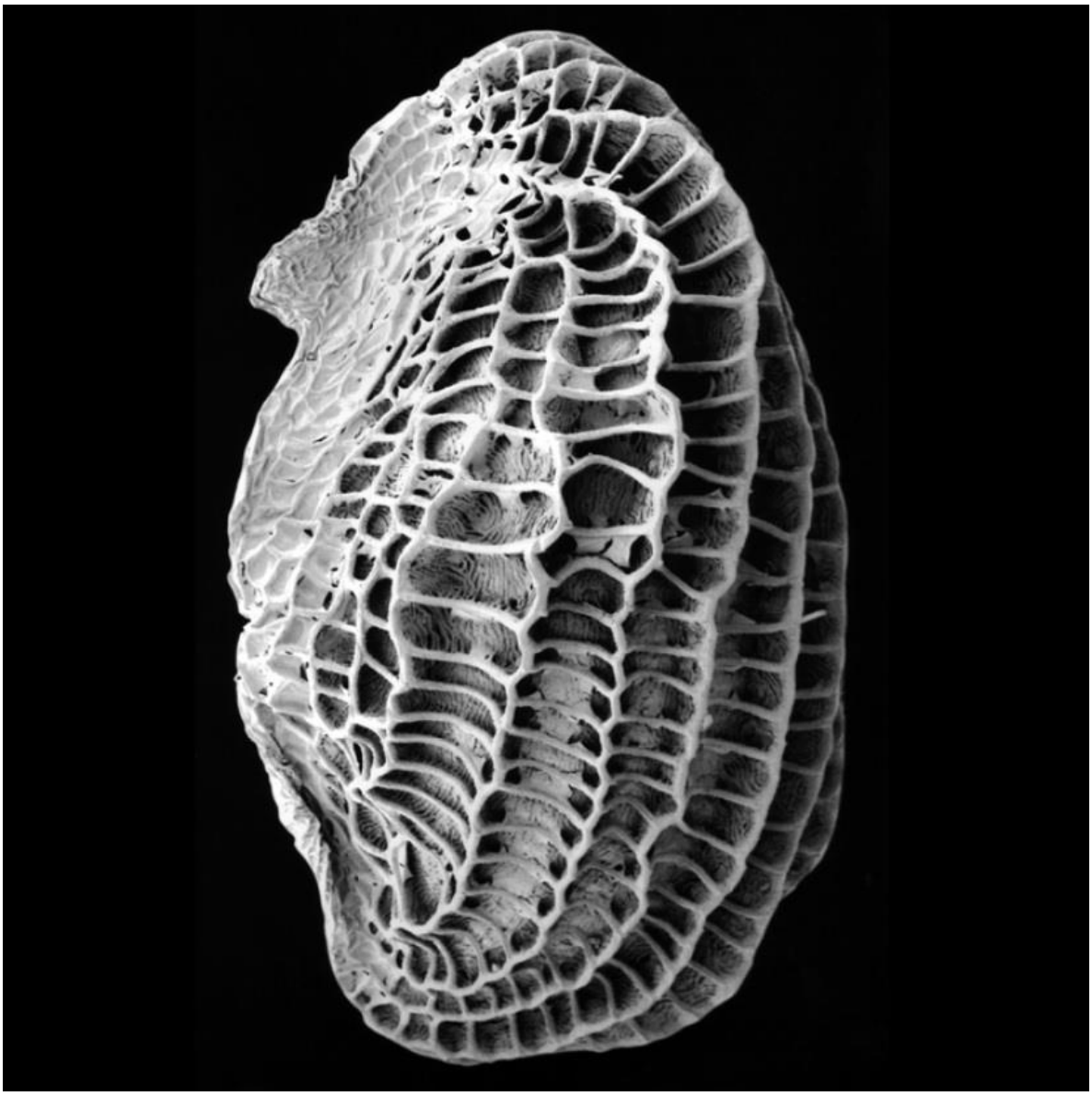


Figura 53. *Orthocarpus Luteus* (Amapola del Búho Amarillo) fotografía por Robert Kesseler y Wolfgang Stuppy. Como se ha señalado anteriormente, la generación de formas por parte de la naturaleza se caracteriza por el alto nivel de personalización en respuesta al medio ambiente. En la naturaleza se encuentra la integración completa entre la forma, la estructura y el material. Tal integración, se consigue mediante la asimilación de la generación formal, la adaptación y el crecimiento análogo y moderado, además del análisis y producción aplicado al diseño mediante medios humanos o artificiales.

4.2.2.2 – Sistemas naturales

Los copos de nieve, por ejemplo, muestran una gran diversidad de formas, que son regidas por ciertas condicionantes físicas, geométricas y químicas, además de temporales. El copo de nieve se considera un arquetipo, que según Pearce, es una expresión particular y formal física – geométrica, ya que un copo de nieve nunca puede ser igual a otro. El caso de los cristales es un ejemplo similar. Aquí se puede encontrar la expresión de la diversidad de la forma en función de la disposición interna de energía y conjuntos de matrices atómicas.

4.2.2.3 – Sistemas Biológicos

En las estructuras biológicas, la molécula de ADN sugiere un principio de *Inventario Mínimo* y de *Máxima Diversidad*, desde el punto de vista de la interacción biomecánica (Pearce, 1981). Los materiales naturales están hechos de un número relativamente pequeño de componentes poliméricos y cerámicos o bloques de construcción, algunos de los cuáles son los mismos compuestos (Vincent, 1982). La parte sólida de la mayoría de las plantas, por ejemplo, se compone de celulosa, lignina y hemi-celulosa, mientras en contraste, un tejido animal se compone en gran parte de colágeno, queratina, quitina y minerales como la calcita, aragonita e hidroxiapatita. De los anteriores, la naturaleza fabrica una notable variedad de compuestos estructurados (Vogel, 2003). En estos casos, como en muchos otros, un sistema genérico singular y limitado en su *Inventario*, promueve la *Máxima Diversidad*.

4.2.2.4 – Conservación de Recursos

El proceso formativo en estructuras naturales se encuentra típicamente caracterizado por la respuesta de ahorro de energía. Este principio se aplica en conjunto al mundo animado y el inanimado. Curiosamente, el principio de ahorro de energía, más cercano es el equivalente a la triangulación en la geometría de las estructuras naturales, y es un hecho que los marcos triangulados presentan una estabilidad geométrica inherente (Thompson, 1942; Pearce, 1981). Además de que se elaboran sin juntas de momentos, lo que asegura a los miembros del sistema una descarga de esfuerzos de forma axial. Esto, a su vez, se traduce en una alta resistencia al peso en la estructura con el principio de energía mínima[34]. Este principio es notable, ya que funciona independientemente de la escala y el material con el efecto de conservar la misma energía. Ya sea en el nivel molecular, el nivel celular, o en los niveles de estructuras hechas por el ser humano, su estabilidad es inherente y siempre establece una potencial condición de gasto de mínima energía.

4.2.2.5 – Energía Mínima

La naturaleza crea formas y estructuras de acuerdo al principio de energía mínima (Thompson, 1942). Lo hace como respuesta a las fuerzas y a las cargas, mientras se lleva a cabo la producción de una gran diversidad de formas en inventarios particulares de cada sistema biológico, a partir de algunos principios básicos y de algunos materiales específicos. Además, la forma de un objeto es análoga al diagrama de fuerzas que la definen (figura 54). Tal es, que esta idea rige la aplicación del principio de *Máxima Diversidad* a partir de un *Mínimo Inventario* para el diseño de una estructura particular aplicada al diseño.

Pero, ¿Cómo se implementan dichos principios del mundo de la naturaleza desde una perspectiva computacional aplicada a un proceso de diseño?, existen muchas maneras para considerar dichos principios en el diseño computacional. Para el diseñador pueden existir intereses particulares en interés de la enorme cantidad de rasgos de la naturaleza que se desarrollarán en este caso de estudio.

34 Haciendo referencia al contexto, al considerar los estudios de cometas y estructuras espaciales de Alexander Graham Bell.

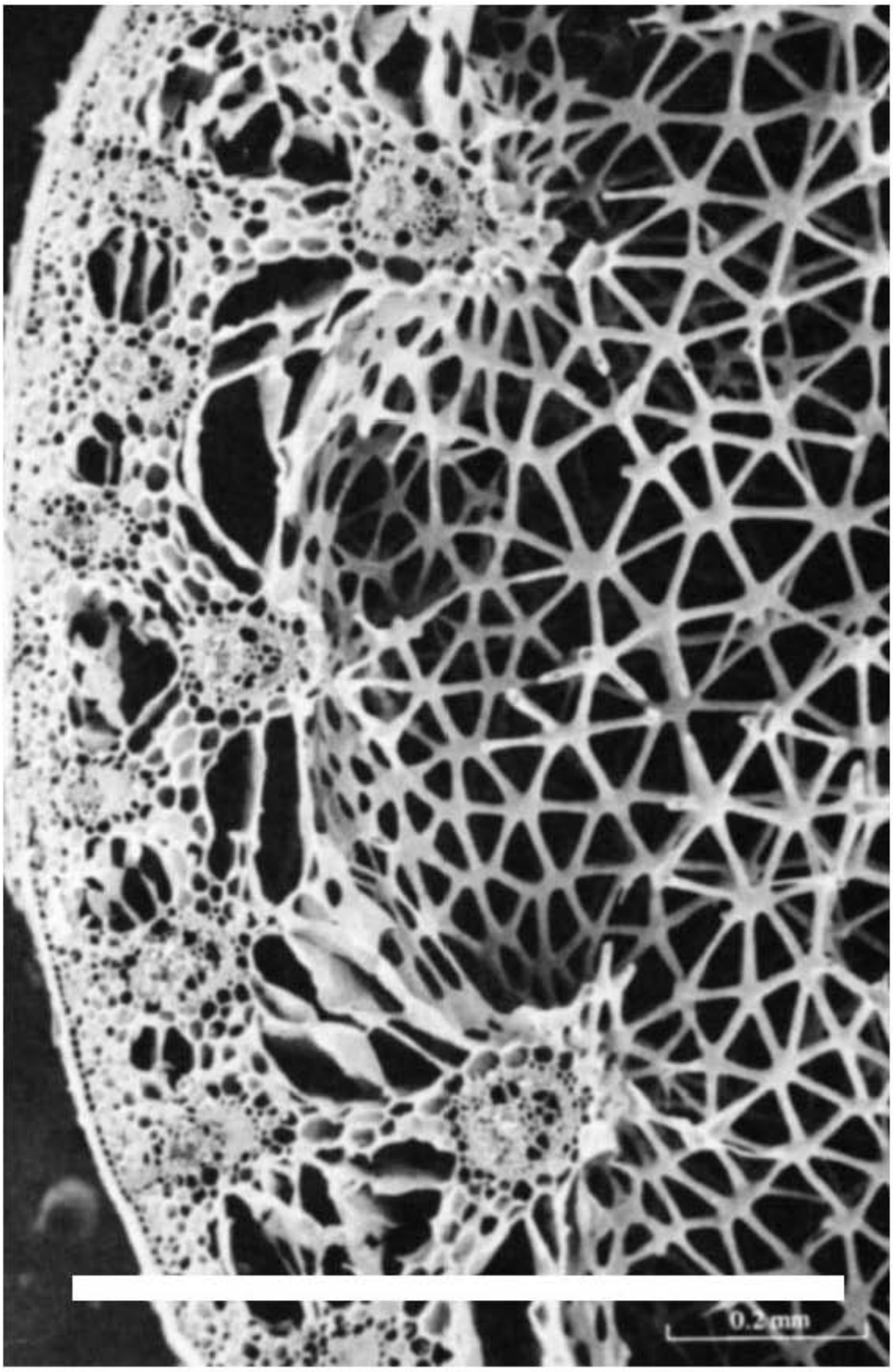


Figura 54. Escaneo de micrografía electrónica de una sección de un tallo (*Juncus Effesus*) a través de una fibra de tejido vegetal (Neville, 1993). La distribución geométrica que llena el centro de la raíz está conformada por una construcción geodésica de células tubulares estrelladas, cuyas paredes son reconocidas a partir de los patrones en arcos triangulados; obtenidos por la microscopia electrónica de transmisión helicoidal. Esta imagen representa las dicotomías entre forma, tamaño, variación en la escala, estructura y el material en los sistemas creados por la naturaleza.

4.2.3 – Estrategias de diseño sustentable

4.2.3.1 – Criterios integrados para el desempeño ambiental: Carga, Luz y Estaciones

En consideración con la utilización de los materiales naturales, se tiene que distinguir un cambio entre los conceptos de estructura y piel, con el fundamento de que muchas formas de la naturaleza asimilan una gran variedad de funciones en relación a su material y sus propiedades. Cuando un ser humano observa con detenimiento su piel, rápidamente asimila que este órgano se encuentra conformado por un tejido que es constante pero que cumple diferentes funciones al considerar que trabaja como conjunto de barrera y filtro de las condiciones ambientales exteriores. Ciertamente la piel tiene propiedades estructurales significantes, las cuales permiten un gran número de distintas funciones. Esas funciones incluyen la captación de energía (por ejemplo, la captación de energía a través de los ojos de algunos insectos), generación de colores (en las pigmentaciones de las alas de las mariposas), transmisión y absorción de calor (en las plumas de los pingüinos), transmisión de masa, reducción de barreras energéticas (como la piel de un tiburón), adhesión a través de superficies (como los gusanos o los caracoles), repulsión de superficies, tacto, actuación endémica, entre otros. El único denominador común de todas las características anteriores es que estos organismos se encuentran construidos de complejas estructuras a partir de fibras de tejidos particulares. De hecho, las estructuras naturales poseen un alto nivel de relación en cuanto a integración y precisión de acuerdo a las funciones específicas de cada una de ellas. Un aspecto muy distintivo en la generación del “diseño” de estas estructuras es la capacidad de adaptación al mundo biológico, al cual responden con la generación de estructuras geométricas complejas construidas a partir de compuestos orgánicos y no orgánicos como las conchas, perlas, corales, madera, huesos, seda, colágeno, fibras musculares y tentáculos (Benyus, 1997). Combinados con matrices extracelulares, estos biomateriales estructurados a través de formas definidas por la naturaleza son diseñados con el objetivo de la adopción temporal o permanente a las condiciones externas, introduciendo en ellas características dinámicas particulares referidas al crecimiento y lapsos de auto – regeneración relacionadas con las estaciones de año del planeta Tierra (Vincent, 1982). Tales constantes generalmente incluyen combinaciones de criterios estructurales, ambientales y morfológicos en conjunto con criterios de desempeño o comportamiento biológico (figura 55).

A partir del hecho de que muchos materiales biológicos se encuentran construidos a partir de fibras, su multifuncionalidad frecuentemente ocurre en una escala nano hasta una macro y típicamente es conseguida por los requerimientos en las investigaciones del comportamiento para las estrategias de estructuración de los materiales y su localización específica en los diferentes contextos en el planeta. La *Forma de la Materia*, por lo anterior, está directamente relacionada a las fuerzas actuando sobre ella de manera constante (Vogel 2003). El material se encuentra concentrado en regiones de alta concentración de fuerzas y se dispersa en áreas dónde un refuerzo no es requerido. Es bien conocido el hecho de que las formas creadas por la naturaleza son más eficaces que el material que las conforma, aunque el material puede ser en algunas ocasiones más eficaz, ya que puede cumplir los criterios de forma a partir de una eficiencia en el aspecto estructural.

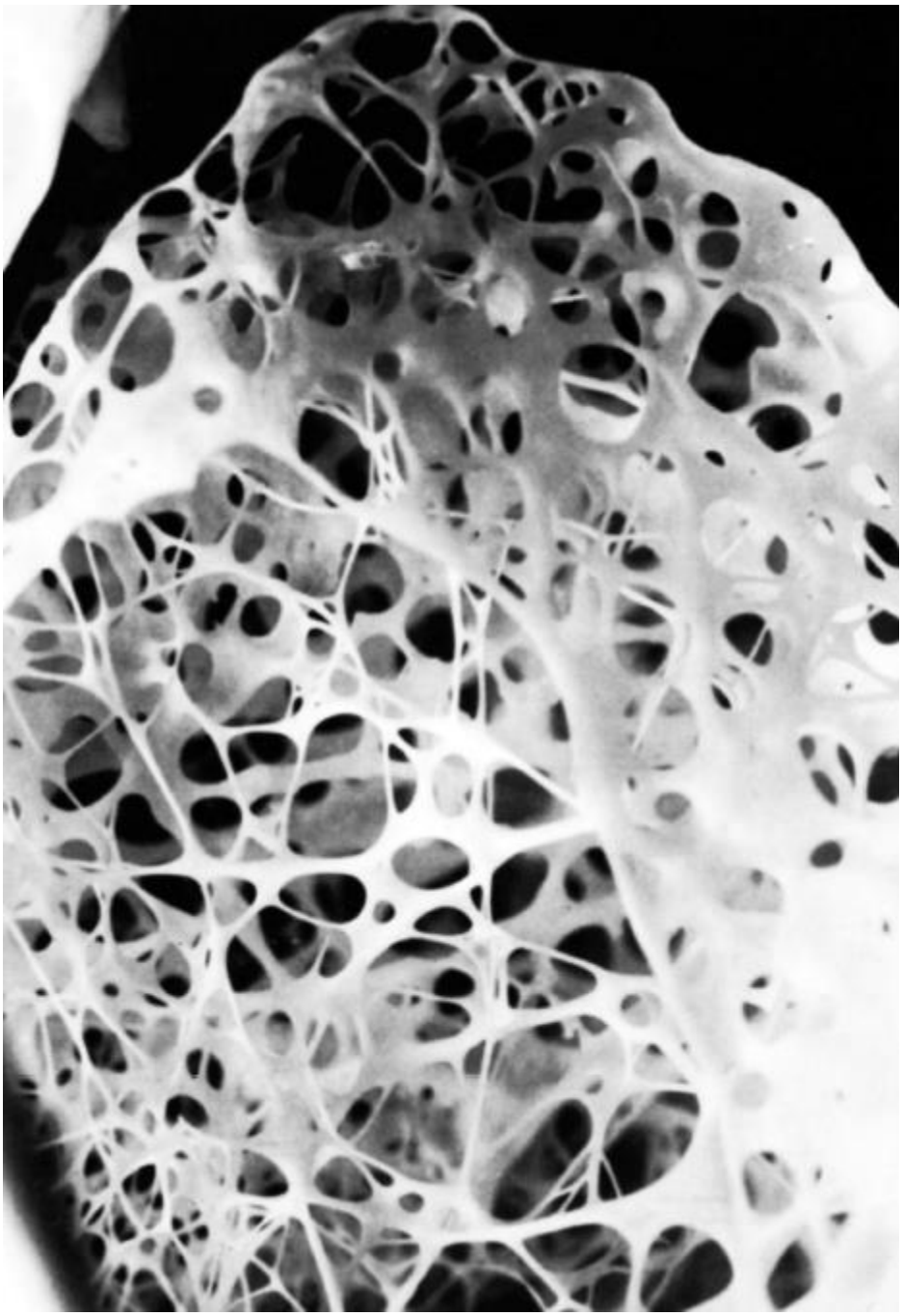


Figura 55. Imagen de acercamiento de un hueso esponjoso de un fémur humano (a pesar de la gran densidad típica de esta región del hueso), fotografía de Klaus Bach, Archivo IL. Fuente: Frei Otto imágenes de investigación. Información del Instituto de Estructuras Ligeras de la Universidad de Stuttgart, Baden-Wuerttemberg, Alemania.

La habilidad de la naturaleza para distribuir el material de manera gradual en un sistema biológico, se realiza a partir de la optimización local de las regiones de la estructura particular natural en relación a los requerimientos externos; como la habilidad de los huesos de cualquier animal para modelar la forma a partir de las cargas mecánicas a las cuales se encuentra expuesto (figura 56), o la capacidad de la madera en modificar su forma en relación a la humedad contenida, y se fundamenta por la habilidad simultánea que tiene al ejecutar los modelos, la simulación y la fabricación de la estructuración del material. Las propiedades estructurales de la madera, por instancia, no son muy diferentes a la de la mayoría de los materiales biológicos, y pueden variar enormemente cuando son medidas en relación al crecimiento de los frutos o cuando una fuerza externa actúa sobre la estructura y es medida en diferentes orientaciones[35] (figura 57).

Figura 56. Escaneo de la sección de un fémur humano revelando un hueso trabecular distribuido a lo largo de líneas de tensión y compresión. (Reproducido por Thompson, D. W. 1992). En el libro On *Growth and Form*. Localizado en el libro en alemán *Das Grosse Buch der Bionik, Neue Technologien nach dem Vorbild der Natur, Deutsche Verlags – Anstalt,* München, p. 243.

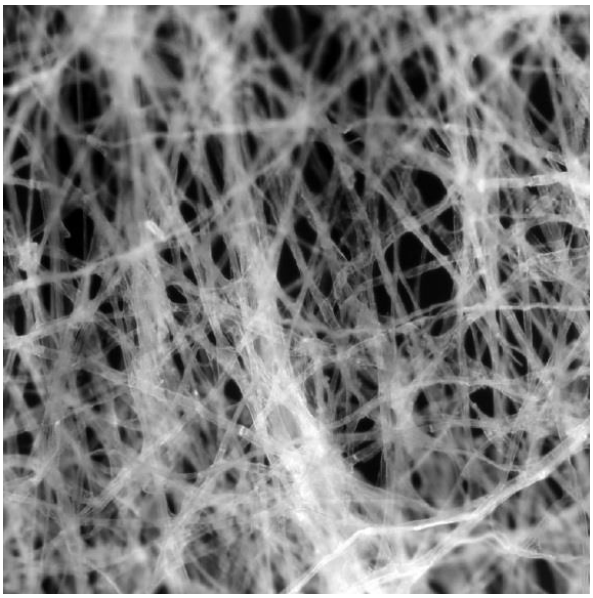


Figura 57. Las fibras propias de la madera dan al material su naturaleza anisotrópica Por medio del control de la densidad y la dirección de las fibras y el comportamiento del material que puede ser significativamente modulado. Fuente: *http: // reference.findtarget.com / search/Solid /*

35 Esta propiedad es conocida como Estructuración Anisotrópica relacionada a la acción de la creación por parte de la naturaleza de estructuras eficientes y sustentables.

4.2.3.2 – Procesos Ambientales Integrados: Crecimiento, Respuesta y Adaptación

Millones de años fueron necesarios para establecer una fina sintonía entre la relación de la estructura y la función en la naturaleza, ya que son soluciones muy notables a problemas complejos estructurales y mecánicos[36] (Weiner y Wagner, 1998). Sin embargo, los sistemas y estructuras de los materiales naturales son capaces de cambiar sus propiedades, forma, color y rutas de carga para considerar la variación estructural y geométrica en relación a las constantes ambientales, tales como el clima del lugar, así como también considerar el daño y la autoreparación. La supervivencia va dependiendo en la mayoría de los organismos en la negociación que han establecido con la naturaleza, entre las múltiples funciones y la evaluación de su importancia dentro de procesos integrales singulares de mecanismos de crecimiento, respuesta y adaptación. Asimismo, lo anterior es de considerar en el aspecto de optimización de recursos para una función simple de un objeto particular, la naturaleza establece negociaciones entre múltiples funciones a partir del significado particular del material de un sistema biológico. La supervivencia como tal del sistema es la que depende de la habilidad de la naturaleza para el control y la distribución de lo más "viable" en su propia construcción, así como de la satisfacción de una serie de propiedades mecánicas como la fuerza, la compresión, la rigidez, la dureza y la resistencia al impacto. En suma, a la generación formal y la respuesta del sistema al contexto natural a través del análisis biológico, una de las características más únicas de los sistemas biológicos es la capacidad para autodiagnosticarse y autorepararse por medio de la localización del daño en cada una de sus estructuras particulares. Claramente, como un atributo deseable para los objetos diseñados y producidos por el ser humano[37].

Es siempre apasionante tratar de entender el impacto que las ciencias naturales han tenido desde la década de 1930 en relación al avance en el diseño, que diseñadores modernos han encontrado a partir de la inspiración en las ciencias de la vida; más emocionante es, sin embargo, el reconocimiento de la conversión de las condiciones establecidas y su aplicación en las disciplinas del diseño, algo que debe de ser celebrado. Tal como fue el caso de la inclinación por la naturaleza del ecologista Julian Huxley y su relación directa con el arquitecto alemán Walter Gropius, uno de los fundadores y directores de la escuela de diseño Bauhaus en Alemania, quién en los tiempos del inicio de la Alemania Nazi, decidió emigrar junto con su amigo a Londres para desarrollar sus ideas[38]. En relación a Gropius, László Moholy – Nagy fue el fundador principal del taller de metales de la misma escuela y un renombrado artista, al igual que sus compañeros diseñadores radicales, quiénes trataron de unificar las artes y los oficios con la universalidad de la producción industrial, encontrando nuevas voces, así como también nuevos representantes en una comunidad de biólogos en Londres, quiénes trataron de adaptar la arquitectura y las artes de la escuela de la Bauhaus, en conjunto con una visión científica del futuro próximo (Anker, 2006).

36 Se tiene que considerar que, a pesar de los distintos lapsos que ha llevado la evolución, que las estructuras biológicas no siempre han estado perfectamente adaptadas a su función en relación con el medio natural o biológico. Esto es en parte debido al hecho de que más allá de las funciones mecánicas de los materiales, algunos, y no todos, de ellos debían satisfacer otras necesidades primarias en vez de las formales o de apariencia.

37 Sin embargo, al utilizar la energía de cualquier fuente para incrementar la rigidez, no siempre es la "solución" más eficaz de la naturaleza. La naturaleza emplea filtros de energía como un mecanismo de control de daño para el diseño de sistemas con grandes capacidades de resistencia en comparación con nuestras construcciones "modernas" a base de acero, concreto y carbón. La adaptación de la impedancia estructural, es quizá, el concepto más fundamental y eficaz de los sistemas de materiales "inteligentes". Por medio de la modificación de la impedancia estructural de un sistema, podemos modificar el comportamiento acústico y de vibración, así como también su resistencia al daño.

38 Julian Huxley fue el secretario de la Sociedad de Zoológicos de Londres. Él tenía un departamento en el zoológico, el cual utilizaba como espacio de exhibición de objetos de diseño moderno. Aquí, científicos, artistas, arquitectos, ambientalistas y escritores de ciencia ficción como H. G, Wells se reunían para discutir cómo salvar a la humanidad de la destrucción ambiental y económica, así como la destrucción de las sociedades humanas.

4.2.4 – Ciencias relacionadas y planteamientos

4.2.4.1 – Bioingeniería

En relación a este campo del conocimiento, el artista Moholy – Nagy encontró su mayor recurso de inspiración en Raoul, Francia. Este país es conocido, además, por la fundación de la ciencia biónica, a partir de la motivación y pretensión de que los humanos deben saber cómo copiar las "invenciones" de la naturaleza con el objetivo de adaptarse de mejor manera y sobrevivir en conjunto con la naturaleza Anker (2006).

Los estudiantes y seguidores del artista Moholy – Nagy fueron instruidos por su maestro desde el punto de vista de "modelos constructivos" y en la investigación relacionada a los "prototipos de la naturaleza" con el objetivo de determinar un mejor acercamiento al diseño funcional. Él propuso que todas las tecnologías humanas están basadas en las tecnologías de la naturaleza. Moholy – Nagy estudió por sí mismo los trabajos de Julian Huxley y sus colegas, y los usó para generar sus propios principios, técnicas y procesos de diseño con el propósito de poder ser aplicados al diseño humano. Su película de 1935 titulada *In the Cradle of the Deep* documenta el crecimiento de las langostas y el esfuerzo de los pescadores para poder retirarlas del agua. Argumenta que la cubierta o concha de un animal prehistórico está construido de una manera maravillosa, tal que podría adaptarse inmediatamente a una Baquelita[39] u otras formas a base de polímeros. El punto principal fue demostrar a los diseñadores que la observación de la vida de los animales y plantas, puede instruir de buena manera a los diseñadores que siguen el principio de "la forma sigue a la función". Por otra parte, las investigaciones de este artista relacionadas a la arquitectura y a la fotografía se basaron en las ciencias de la vida. La arquitectura es como "un componente orgánico de la vida humana" y argumentaba que "la arquitectura podría ser llevada a cabo en su máxima expresión, solo cuando el conocimiento profundo de los sistemas biológicos, sea comprendido y aplicado por los seres humanos" (Anker, 2006).

4.2.4.2 – Biomimética

Con respecto a esta motivación subyacente para traducir los procesos naturales, y en particular el crecimiento a principios de diseño, la Biomimética es una ciencia emergente que tiene como objetivo, el estudio de las soluciones aplicadas a las ramas de diseño, desde el punto de vista relacionado a los problemas del mundo natural como un aspecto potencial y relevante en el diseño y la ingeniería contemporánea. De acuerdo con el autor B. M. Katz Klein, ha definido este campo como el estudio de los procesos naturales y como pueden ser emulados para resolver problemas humanos en un ambiente "amigable" libre de desechos tóxicos y perjudiciales con la naturaleza (Klein, 2009). Hay que distinguir y hacer un contraste de esta disciplina con la ingeniería genética, la cual convierte a los organismos naturales a partir de sus células en fábricas biológicas para la producción industrial; la Biomimética trata de producir los agentes productivos de las sociedades humanas en agentes de la naturaleza. El mundo natural puede ser tratado como una librería de ideas, tal como lo menciona J. F. Vincent, un investigador reconocido dentro de un número limitado de expertos relacionados con este campo de conocimiento, el cual lo define simplemente como "la abstracción de un buen diseño, basado en los diseños primarios de la naturaleza" (Vincent, 1990). Aunque esto puede sonar obvio, se trata de procesos intelectuales y relacionados con investigaciones mucho más complejas.

39 La Baquelita es una resina antigua de fenol formaldehído termoendurecible, fue usada en relleno de harina de madera y desarrollada por el químico belga Dr. Leo Baekeland.

Aunque el campo de la Biomimética se puede remontar a la obra de Leonardo da Vinci, se considera que su formulación final fue realizada en 1991, cuando la oficina de la Fuerza Aérea de los Estados Unidos de América, llevó a cabo una investigación científica que convocó a un grupo de investigadores a considerar las formas y materiales que la naturaleza produce y procesa de manera única. Así, la motivación original era lograr, a través de la inspiración, de los materiales naturales y sus procesos de generación, una fuente posible para innovaciones en el diseño de materiales que pueden ser aplicados como medios de soluciones prácticas. El campo también se conoce con el término de "Biónica" que es la copia humana por medios artificiales de diseños propios de la naturaleza. La definición actual de la Biónica, es también atribuida a la Fuerza Aérea de los Estados Unidos de América, pero basada en la definición que formuló Julian Vincent en el año 1950.

A pesar de que los marcos de tiempos referidos a la evolución de millones de años de los organismos en la naturaleza, puede establecerse que es un rango completamente diferente a la escala de desarrollo de las ingenierías contemporáneas, el diseño de productos, de materiales y de la arquitectura; se ha considerado que los objetivos del diseño en estos campos eran a menudo similares a los que subyacen en los procesos evolutivos generales de la propia naturaleza. Entre ellas se encuentran la funcionalidad, la optimización, la eficiencia energética y la rentabilidad. Ahora podemos estudiar cómo la naturaleza ha optimizado a la arquitectura referida a la producción de materiales en el largo periodo de evolución natural y particular del ser humano. Todos estos factores son comunes a los campos de la ingeniería y diseño contemporáneo. En los estudios relacionados a cómo la naturaleza alcanza una optimización propia, nos podemos concentrar en las propiedades de los materiales, las propiedades estructurales, las propiedades mecánicas, la integración funcional, sistemas de detección – protección, el control, y otros aspectos de diseño que tienen relevancia general para los inventos y soluciones contemporáneas aplicadas a las problemáticas generales y particulares del ser humano, su entorno y su hábitat (figura 58).

En los procesos productivos de la naturaleza hay poco énfasis en la esotérica o en materiales "caros". Existen pocas sustancias químicas utilizadas en la vida de los animales, y que se construyen generalmente usando materiales relativamente de bajo rendimiento. El éxito desde este punto de vista aplicado al diseño, no depende de lo que están hechos, sino más bien, en la forma y en los procesos mediante los cuales se producen. La estructura es más importante que la energía, ya que es el principio general de toda la naturaleza, desde el punto de vista de que la energía se emplea, por parte del organismo, para la optimización de su estructura para el sustento principal. La forma es más "barata" que el material que la constituye, por lo tanto, tenemos todavía mucho que aprender y reflexionar de la naturaleza en relación a los enfoques eficaces e innovadores que se puedan aplicar como parámetros en el diseño descritos anteriormente.

Uno de los estudios clásicos que pueden considerarse como los fundamentos de la materia, es la obra de D'Arcy Thompson, un naturalista y matemático escocés que publicó los resultados de sus investigaciones sobre las formas "originales" propias de la madre naturaleza, titulada *On Growth and Form*, del año 1917. En este trabajo, fue una de las primeras ocasiones en la historia del ser humano, que los seres vivos son estudiados como posibles soluciones de problemas en la disciplina de la ingeniería (Thompson, 1942).

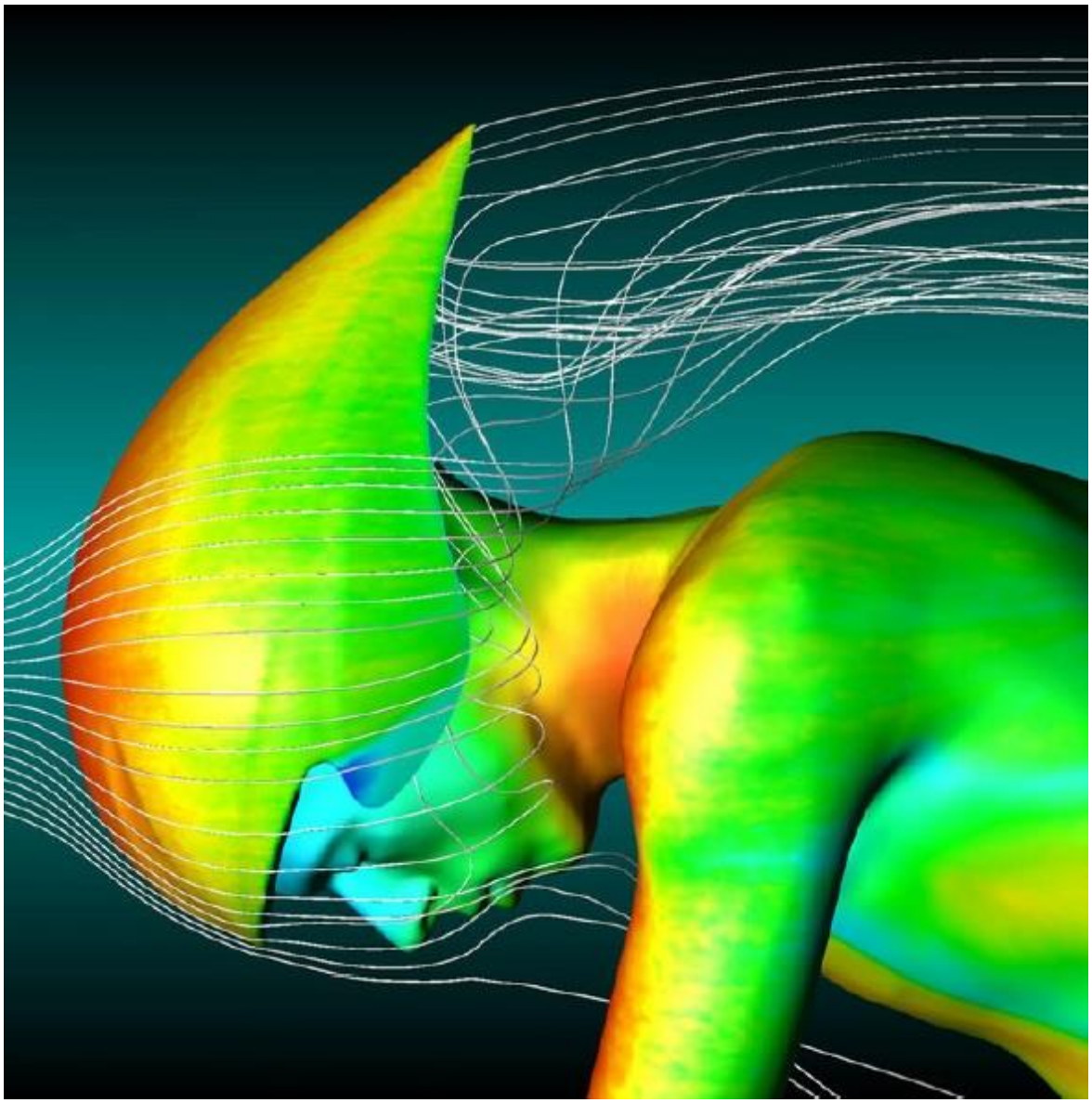


Figura 58. Dinámica de Fluidos Computacional (CFD por sus siglas en inglés), la imagen representa un análisis de modelos ambientales aplicados a un ser humano, en función de variables relacionadas con la termodinámica, viento y energía captada por la piel, así como las zonas más calientes y más frías cuando se encuentra en reposo y movimiento respectivamente. Este tipo de análisis permite a los diseñadores de ropa del atleta, incorporar materiales y constantes ambientales como parte de un proceso particular de diseño. Fuente: *http: // legacy.ensight.com / news / british_cycling.html.* Diseño Computacional [Capítulo 4], el cual introduce los estados – de – el – arte en estrategias computacionales inspiradas en el mundo de la física.

4.2.4.3 – Biogénesis

En este documento vamos a considerar nuevos enfoques de generación formal y geométrica, a través de herramientas computacionales referidas a la programación gráfica, inspirados en la naturaleza y desde un punto de vista de comportamiento propio del material. El primer enfoque, relacionado con un serio acercamiento al término de "Biogénesis" fue elaborado por A. G. Michell alrededor del año 1900 en una publicación que se titula *The Limits of the Economy of Material in Frame – Structures*. Después de eso, la publicación de *On Growth and Form*, fue el primer documento que tomo en cuenta los requerimientos estructurales de las plantas y los animales (Thompson, 1942).

Otros diseñadores e ingenieros han desarrollado aún más la relación entre la biología y la naturaleza, en el contexto aplicado a la investigación de la forma. Entre ellos se encontraba el arquitecto alemán Frei Otto y su grupo de investigación en el Instituto de Estructuras Ligeras [ILEK por sus siglas en alemán) de la Universidad de Stuttgart, Baden-Württemberg, Alemania, durante la década de los años de 1970[40]. Una serie de publicaciones de la época, específicamente en la IL-6, que conmemoraba el décimo aniversario del instituto, el arquitecto ofrece una declaración honesta acerca de este tipo de asociaciones y la productividad a partir de la cual se inspiraba: "todo objeto material tiene la capacidad de transferencia de fuerzas (figura 59). Estamos estudiando la posibilidad de transmitir fuerzas independientes de la forma, material y tipo de cargas, ya sea en todos los objetos conformados de materia orgánica e inorgánica" (Otto, Frei; Trostel, 1973) (figura 60).

En este capítulo hemos revisado la estructura teórica de las relaciones mecánicas en los distintos niveles de organización, destacando siempre que se posible, los procesos más significativos de generación y estructuración de forma sintética.

40 Para mayor información relacionada a los estudios anteriores y actuales de este instituto de investigación, enfocado a las estructuras a partir del conocimiento de la anisotropía material, consultar el siguiente enlace web: http://www.uni-stuttgart.de/ilek/forschung/adaptivitaet/

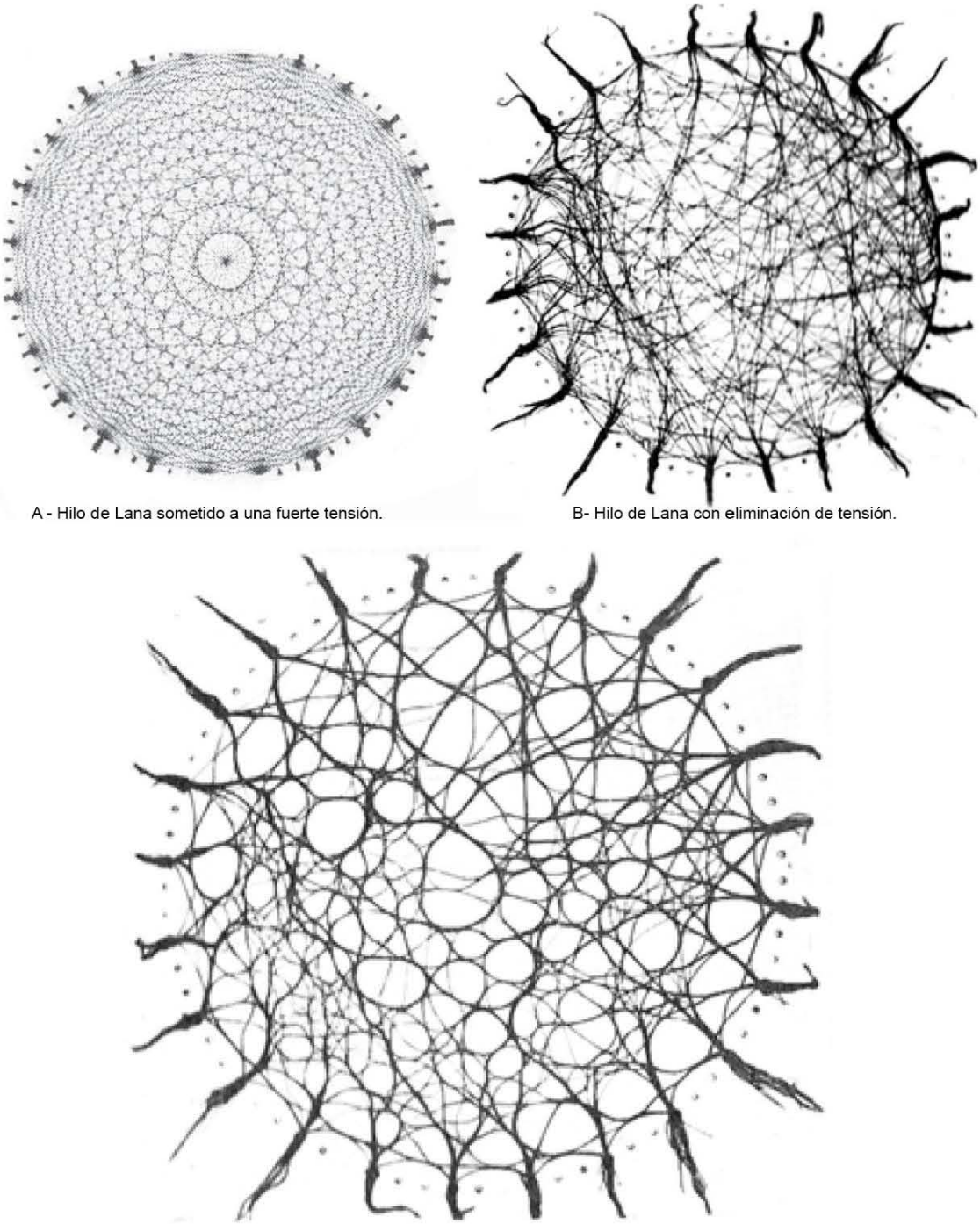


Figura 59. Los experimentos sobre hilo de lana de Frei Otto en los inicios de la década de 1990. Las imágenes muestran el comportamiento a una escala mayor con el empleo de un microscopio de las fibras en unos hilos de lana distribuidos de manera radial y apoyados en la circunferencia exterior de una forma a la cual es aplicada una fuerza de tensión, posteriormente se elimina la fuerza al comprobar su estado plástico máximo y es reforzada, en la última instancia, mediante la aplicación de un fluido viscoso para estabilizar los componentes finales en base a su deformación. Fuente: ILEK, imágenes proporcionadas por el Ing. Prof. Arch. Diplom. Werner Sobek, director actual del Instituto [Julio.2015].

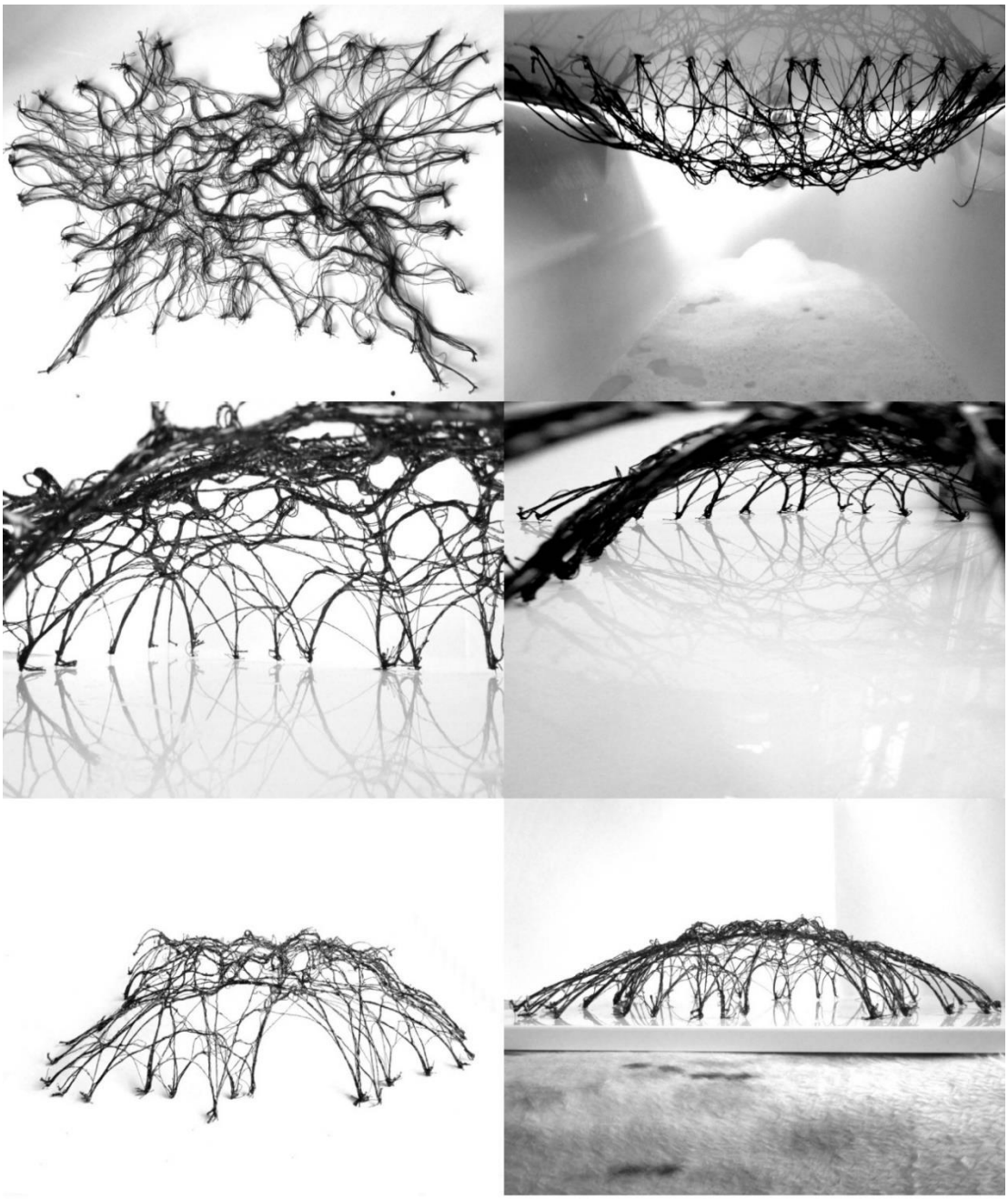


Figura 60. Estas imágenes muestran el comportamiento de hilos de lana sometidos a los esfuerzos descritos en la explicación de la figura 61, pero en el espacio tridimensional. Estos experimentos fueron influenciados por los modelos de catenarias que desarrolló el arquitecto catalán Antoni Gaudí, usadas para diseñar la iglesia de la Sagrada Familia, en Barcelona, España. Estos experimentos fueron significativos para el desarrollo de estrategias para el cálculo de la infraestructura bidimensional de una ciudad experimental, así como el cálculo tridimensional de las estructuras principales. Los hilos de lana que se encuentran comprimidos actúan como agentes estructurales a través del cálculo de grados de deformación y flexibilidad. Los experimentos también fueron considerados análogos para el diseño computacional de estructuras que se desarrollan actualmente por el Instituto de Estructuras Ligeras [ILEK] de la Universidad de Stuttgart. Fuente: ILEK, imágenes proporcionadas por el Ing. Prof. Arch. Diplom. Werner Sobek, director actual del Instituto [Julio.2015].

4.2.4.4 – Estrategia de Diseño Computacional – Natural

Heterogeneidad a través de Simulación Computacional y Material

Cuando los materiales artificiales elaborados por el ser humano, son comparados con materiales naturales, sobretodo del reino vegetal, los retos para el ser humano son superar las propiedades mecánicas de los materiales naturales aplicadas a los materiales más comunes que utilizan los arquitectos y los ingenieros. La madera tiene una resistencia por unidad de peso que es comparable a la de los aceros más resistentes; las conchas, los huesos y cuernos de algunos animales, tienen una tenacidad de un orden mayor que algunos materiales desarrollados por la ingeniería, tales como la cerámica; y las maderas de los tallos del bambú, tienen una relación de esbeltez que es notable en comparación con la flexibilidad que posee, incluso para materiales "modernos" desarrollados por la ingeniería contemporánea (Ashby, 1995). Sin embargo, la mitad de los materiales de la naturaleza no son tan densos como muchos de los materiales artificiales, y se caracterizan por un peso muy bajo y son funcionales, según la mayoría de las perspectivas de los seres humanos, solo para el sustento físico de las plantas[41] (Niklas, 1992).

¿Cuáles son los atributos que hacen de los materiales de la naturaleza tan eficientes?, en los fundamentos de la Biomecánica: los enfoques de ingeniería basados en la forma y función de las plantas, Niklas determina que los organismos viven por las leyes de la física y la química, y como tales, han evolucionado y se han adaptado a las fuerzas mecánicas de una manera consistente a los límites establecidos por las mismas propiedades mecánicas de los materiales que los conforman (Niklas, 1992). Además, en el mundo de las plantas en particular, de manera común se dan los casos de que las propiedades del material influyen en el comportamiento mecánico de la planta[42]. Niklas demuestra, en resumen, a partir del estudio de un fruto conocido con el término científico macadamia [nueces], puede resistir el doble al ser empleado como herramienta para cortar metales de diversos tipos, en comparación con aluminio de alta calidad comercial; y es más fuerte que los vidrios de silicato, concreto armado, porcelana y ladrillos convencionales de construcción.

Los materiales biológicos son de hecho muy versátiles: pueden cambiar sus propiedades materiales físicas y químicas, además de su comportamiento, con el fin de adaptarse a su condición fisiológica inmediata[43]. Por lo tanto, el comportamiento mecánico de cualquier material biológico o natural, solo se puede definir por la variación de sus propiedades, de los cuáles, algunos no pueden ser maximizados en escala. Cada material se utiliza de acuerdo a sus cualidades particulares, los tipos y las magnitudes de las fuerzas mecánicas propias del organismo a sostener.

En las propiedades mecánicas de los materiales naturales, Gibson explora diversas clases de materiales, en relación a sus microestructuras celulares compuestas con altos valores excepcionales de rendimiento mecánico (Gibson, 1995).

41 Desde que la construcción de manera vertical lleva consigo la restricción del diseño de la auto-carga, además de ser muy resistentes, los materiales deben de ser muy ligeros. De hecho, en relación a su densidad, la celulosa es el material más fuerte y ligero conocido e investigado por el ser humano (Niklas, 1992).

42 Dado que el mundo de las plantas carece de sistema nervioso, las observaciones y los exámenes llevados a cabo con respecto a la relación entre las propiedades del material y la función mecánica, pueden ser fácilmente establecidas.

43 La capacidad de respuesta de las plantas a su entorno mecánico inmediato, fue reconocido por los botánicos ya en el siglo XIX, cuando fue descubierto y estudiado por primera vez, que las paredes de las plantas jóvenes son dúctiles, mientras que las paredes celulares de mayor edad, tienden a ser mucho menor elásticas y resistentes (Niklas, 1992).

La función de estos materiales naturales es explotar sus propiedades estructurales más importantes, tales como la madera de las palmas que resiste la flexión y el pandeo, los espacios elaborados con seda que brindan mayor energía de deformación elástica, reservas de tejido muscular animal que liberan energía de deformación elástica durante la locomoción, y así sucesivamente. Estas relaciones tienen importantes implicaciones para el diseño de materiales eficientes por parte de la ingeniería mecánica, al considerar su aplicación al diseño arquitectónico y estructural mediante el empleo de vigas y placas con una rigidez variable, o columnas de una resistencia al pandeo dado; tenemos que hay bosques, palmeras y bambú que son algunos de los materiales más eficientes disponibles (Gibson, 1995). El autor revisa cuatro clases de materiales naturales: la madera, palmas y bambú, tallos y espinas. Los resultados de los análisis sugieren nuevas microestructuras que se pueden aplicar al desarrollo de nuevos materiales de la ingeniería química y mecánica, más eficientes en cuanto a rigidez a la flexión y resistencia al pandeo elástico que se consigue mediante la optimización de la micro-organización estructural para adaptarse a los requisitos de rendimiento, de tal manera que la estructura celular pueda mejorar el rendimiento de la carga paralela a la del grano (Gibson, 1995). Común a todos estos ejemplos son las excepcionales propiedades de los materiales naturales que surgen principalmente a través de nuevas microestructuras basadas en las células generadoras y que pueden aplicarse a la ingeniería de materiales (figura 61). Los bloques de construcción de la naturaleza, por lo tanto, no son tan únicos como su estructuración, en relación a sus propiedades físicas y químicas del material en sus componentes, así como su distribución dentro de los compuestos naturales que dan lugar a una amplia gama de propiedades particulares de cada organismo, animal o vegetal. Por lo tanto, podemos postular que el estudio de la *Estructura del Material* es una propiedad importante para el diseño, así como un importante cuerpo de conocimientos que pueden ser aplicados al diseño arquitectónico y artesanal e industrial.

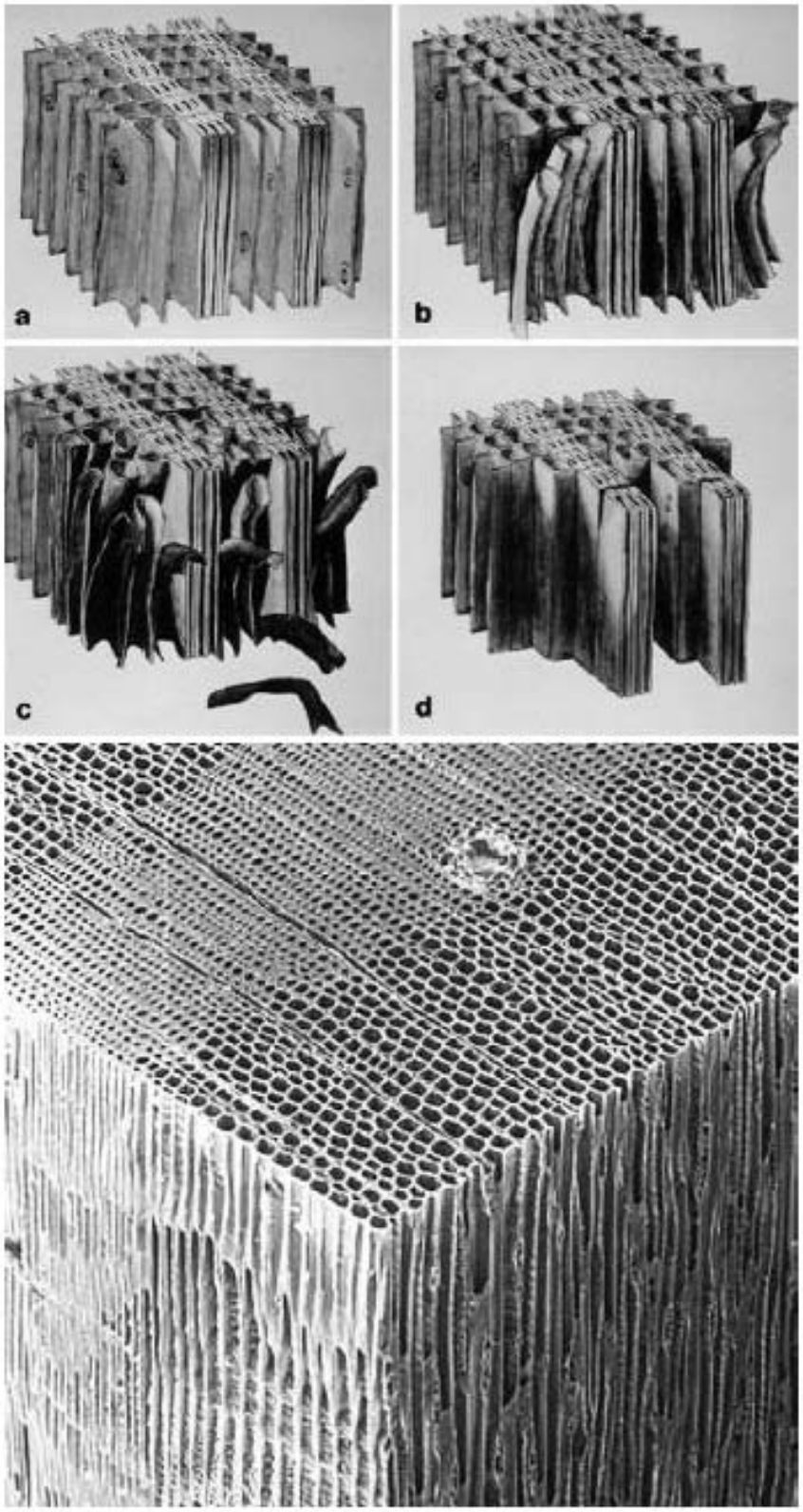


Figura 61. Las fibras propias de la madera, le dan a su estructura particular una característica anisotrópica. Por medio del control de la densidad y dirección en el comportamiento del material, ya que puede ser modulado de manera significativa y particular. Fuente: http: //www: woodmagic.vt.edu / Images / activities / BigFiber: jpg.

4.2.5 - Relaciones entre estructura y función en 8 Niveles Jerárquicos de Organización

4.2.5.1 - Micro-Escala del material

Una estructura conformada por elementos del esqueleto humano no tiene únicamente la característica de adaptarse al tipo de esfuerzo mecánico que se aplica sobre ella. Lo anterior aplica a las mismas extensiones en la distribución y en la densidad del propio material que provee la fuerza mecánica. Actualmente esto es generalmente aceptado, ya que la densidad y la anisotropía de los elementos en los huesos esponjosos del esqueleto humano, dependen de la magnitud y la dirección de la carga que es aplicada a cada unidad (Gibson, 1984; Ashby, 1997).

El investigador Kummer y su grupo de científicos, han desarrollado e ideado técnicas de representación gráficas, tales como los *Material Mountain Ranges* (figura 62), con el objetivo de cuantificar la distribución de la materia en los huesos, así como la distribución del material en relación a los tipos de adaptación particulares a las magnitudes y esfuerzos mecánicos. Los huesos esponjosos del esqueleto humano poseen una estructura celular particular, y se ha considerado la incorporación de propiedades mecánicas similares, así como características de comportamiento estructural parecidos a la familia de sólidos celulares (Gibson, 1984; Ashby, 1995; Gibson y Ashby, 1997). Además, es probable que este sea uno de los ejemplos más “clásicos”, ya que es un sistema que organiza la materia de manera consistente con los patrones de estrés en la estructura, haciendo una distinción que a veces es muy complicada de hacer y comprender.

4.2.5.2 - Heterogeneidad Estructural

En esta relación haré especial énfasis en los árboles, esas estructuras de la naturaleza en las cuales podemos pasar horas observando su belleza, tanto de forma como de comportamiento estructural y función. Entre su extraordinaria belleza, los árboles y los sistemas naturales, como los corales o erizos de mar, son el ejemplo perfecto de sistemas que han sido modelados por las fuerzas de la naturaleza, y que como tales, representan las estrategias de la naturaleza en relación al crecimiento, la generación, la adaptación y la preservación de la geometría y la forma.

Los sistemas que son modelados por la naturaleza, son también expuestos de manera directa a los elementos exteriores e interiores que la conforman. El rango de cargas a las cuales un árbol se encuentra expuesto es muy amplio, e incluye fuerzas de varias magnitudes y direcciones, momentos de inclinación, momentos de torsión y variables térmicas, entre otros. Si un árbol puede resistir las cargas que se aplican sobre él, esas cargas deben de ser contrarrestadas por apoyos que apliquen la misma carga, pero opuesta en relación a las cargas externas.

En las disciplinas de la ingeniería, a menudo se da el caso de que las relaciones entre la fuerza y la forma se simplifican y son abstraídas en modelos geométricos parecidos a las vigas, con el objetivo de reducir la abstracción geométrica y la deformación en el sistema a causa de las fuerzas aplicadas. En la naturaleza, la distribución de la substancia material relativa al control de las fuerzas en un árbol, es realizada y guiada mediante el modelado de la forma a partir de las fuerzas propias de la estabilidad en la estructura del árbol. La relación de la unidad con el todo, y el todo con la unidad de manera directa, el caso del árbol es un buen punto de partida para comprender y entender las fuerzas de la naturaleza.

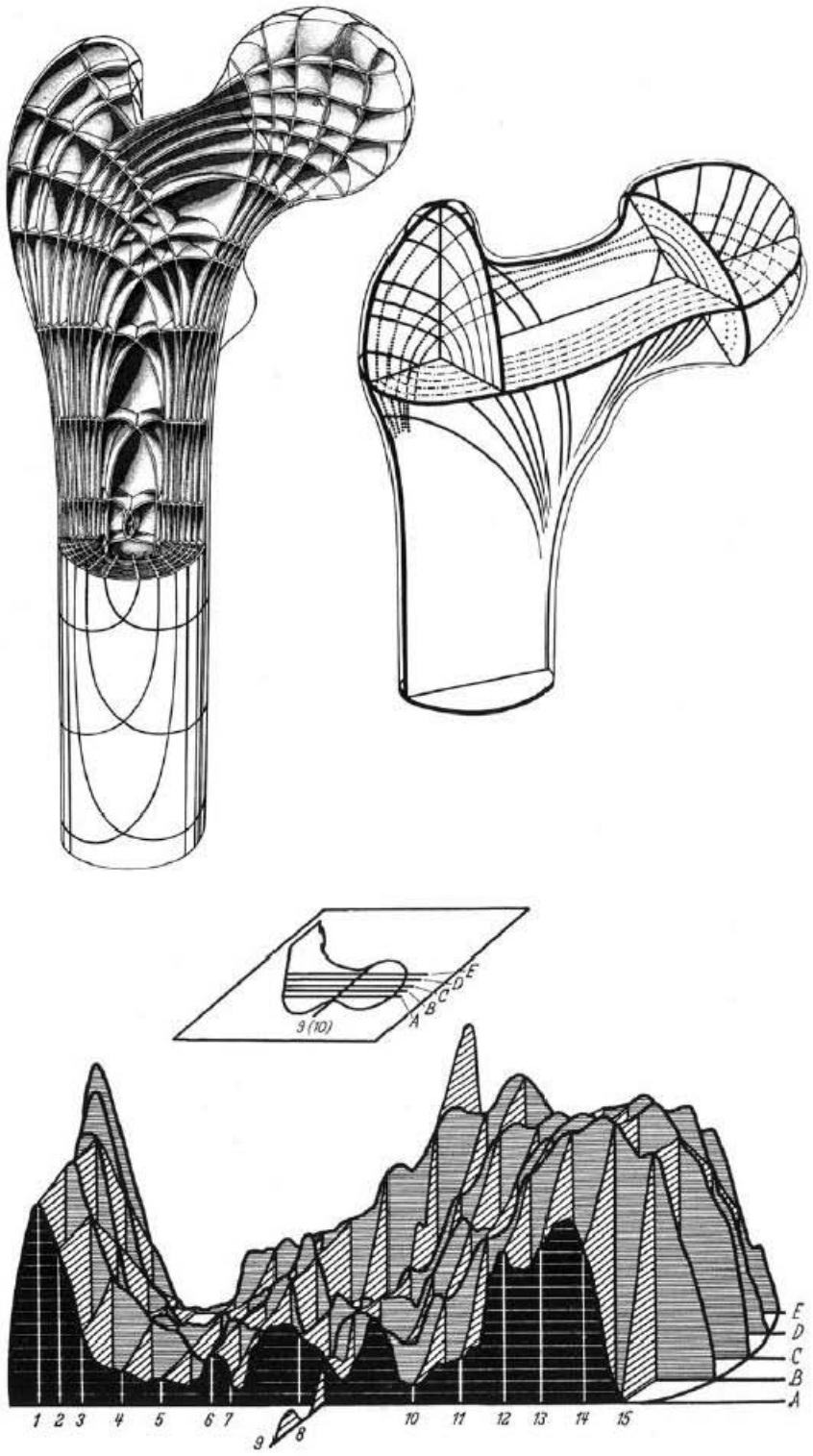


Figura 62. La técnica *Mountain Range* fue desarrollada para ilustrar la distribución de calcio en los huesos, así como la función de la carga aplicada. La imagen superior representa la estructura de un hueso interno que es representada mediante rutas de carga a través del hueso. La imagen inferior representa la distribución del material respecto a la sección anatómica examinada. (Otto, Herzog; 1990).

Las fuerzas de la naturaleza

4.2.5.3 - Fuerzas Axiales

Las fuerzas axiales son un ejemplo muy básico. La materia contenida en las ramas de un árbol ejerce (por su propio peso), una fuerza axial sobre su propio tronco de soporte primario, causando una fuerza de comprensión. Las ramas causan una compresión elástica a partir del tronco y en las direcciones axiales, que resultan en una distribución uniforme espacial de fuerzas de compresión (Mattheck, 1998). Por lo tanto, el suelo que se encuentra en la parte inferior y en donde “nace” el tronco el árbol, debe ejercer una fuerza de igual magnitud, pero en dirección contraria a las ramas, con el fin de solucionar todas las condiciones posibles e hipotéticas en la cuales el árbol se pueda hundir en la tierra. Como reacción, las fuerzas de tracción podrían causar tensiones de tracción en el sentido opuesto. La situación se volverá más compleja de entender si la carga es excéntrica y se encuentra en el punto medio del tronco del árbol o de cada una de sus ramas.

4.2.5.4 - Cargas Excéntricas

Cuando se disecciona a un tronco de manera, es muy sencillo ubicar la localización relativa de las ramas, al establecer áreas en las secciones más densas en relación a tejido propio de la madera que conforma al árbol. En el caso de la carga excéntrica, un momento de flexión uniforme puede estar actuando hacia la parte inferior del tronco del árbol, es decir, las ramas se pandean hacia la parte inferior por la fuerza de gravedad y el peso de las hojas, frutos, aire, entre otros; esto se compensa y es equilibrado por medio de las raíces del árbol en el suelo. En este caso, puede resultar bastante evidente que en el eje central dentro del tronco del árbol y en cada una de las ramas, las fuerzas de tracción y las de compresión no están presentes. A pesar de no estar totalmente cargada, esta zona dentro de la madera y sus limitaciones como material único de los árboles, son capaces de soportar las cargas. Además, es bastante eficaz en lo que respecta a la flexión y, por lo tanto, no hay desperdicio desde una perspectiva de uso de materiales.

4.2.5.5 - Carga Lateral

Los sistemas que actúan en conjunto para formar a un árbol, son capaces de organizar el material de la manera sensible a los límites más estrechos de sus posibilidades. Por lo tanto, si los sistemas del árbol ahora colocan todo el material en la zona donde se concentran las fuerzas de tensión (en el lado de tensión y compresión) para la formación de anillos que anualmente se amplían en estas zonas, esta acumulación entra en la tercera potencia en relación al eje con mayor magnitud de fuerzas (el eje longitudinal que describe la totalidad del tronco), mientras que se propicia un ensanchamiento en la dirección del eje más pequeño en magnitud de fuerzas, ya que solo es lineal. Por lo anterior, el árbol produce una sección transversal no circular que es más rígida contra la carga de flexión que prevalece, y se caracteriza por tener tensiones más pequeñas de magnitud que una sección transversal uniforme, en contraste con un momento de flexión externo idéntico. Las secciones transversales de las raíces pueden incluso suponer una similitud en cuanto a la forma de una viga I, en la que difícilmente cualquier forma de madera, puede tener zonas de flexión neutras. Aquí el componente es forzado a una optimización de la forma (Mattheck, 1998).

4.2.5.6 - Carga Compuesta

Además de las fuerzas de compresión y tensión resultantes de las cargas axiales o la compresión y las fuerzas de flexión (transición de fuerzas de tracción a tensiones de compresión con una definición de distribución espacial, usualmente lineal), existen también tensiones tangenciales. Estas tensiones actúan tangencialmente en el plano de corte – carga, y evitan que los cuerpos separados se deslicen unos sobre otros. Por lo tanto, son cualitativamente muy diferentes a las fuerzas de tensión y de flexión que actúan perpendicularmente a un plano de referencia dado. La carga de torsión causa, por ejemplo, una distribución de los esfuerzos cortantes en el plano de la sección transversal de tronco, el cual se incrementa linealmente desde el centro.

En un árbol, una forma de corona en un solo lado, con una rama que se extiende lejos y en dirección hacia la luz, la carga podría conducir a la torsión del tronco bajo la presión del viento y causar los mismos esfuerzos de corte en el tronco. Cada instancia de carga se compone de un gran número de distintas cargas. Naturalmente, las condiciones de carga son generalmente multi – axiales y diferentes fuerzas de tensión pueden actuar en diferentes direcciones.

4.2.5.7 - Tensión y Compresión en la madera

La madera de reacción, también conocida por su término científico como gravitropismo, es un tipo de madera muy rara que se encuentra en los troncos y en las ramas de los árboles inclinados, que resultan en anillos de tensión y compresión diferentes, inclinadas en dirección contraria a su centro respectivamente. Se encuentra formado por la respuesta de los árboles a la tensión direccional y es un intento de equilibrarla. En las maderas duras, la madera de reacción también es conocida como madera de tensión medida, que se forma en la parte superior de los apoyos de los árboles (raíces). En las maderas blandas, la madera de reacción se conoce como la forma de madera en compresión, que se encuentra en la parte superior en la inclinación del árbol (figura 63).

Aunque la madera de compresión tiene una densidad de materia más alta de lo normal, es débil respecto a algunas propiedades de fuerza. El inconveniente más grave de la madera de compresión es su comportamiento posterior a la evaporación del agua interior. Cuando la madera normal, en estado de madera seca, se contrae en grosor y ancho (por cantidades predecibles). Se encoge muy poco (prácticamente inapreciable) en longitud. La madera de compresión, sin embargo, se contrae en longitud en más de 10 veces que la madera normal, esto causa una inclinación y una torsión en ciertas partes de la madera.

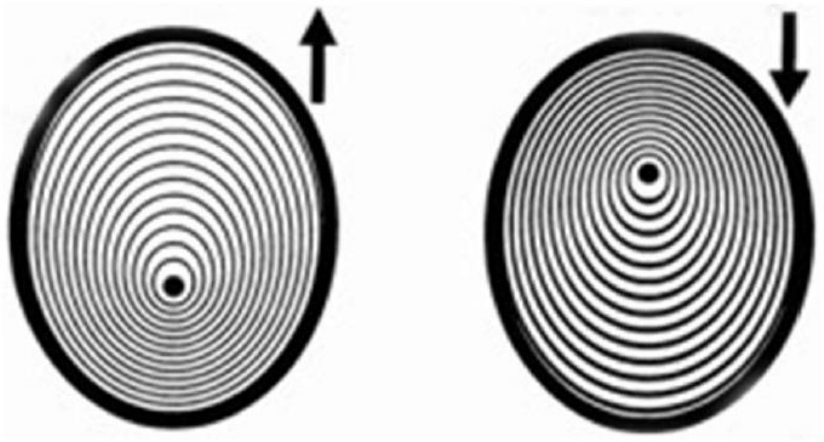


Figura 63. La fuerza de tensión en la madera (izquierda) se inicia en la parte inferior por la inclinación de los árboles de madera dura y se indica normalmente mediante anillos de crecimiento excéntricos. En la madera de menor resistencia la tensión se lleva a cabo por la evaporación del agua del interior de la madera. La compresión en la madera (derecha) se forma en la parte superior al inclinarse los árboles de madera blanda, y por lo general se detecta mediante anillos de compresión anuales, que son más anchos que los de tensión y parecen tener una proporción de materia mayor en el verano. Fuente: *http: // www.forestry.gov.uk / fr / GGAE – 5GGEJN*

4.2.5.8 - Distribución del material a través de la Optimización

La madera de reacción es un caso interesante de la formación de materiales como causa de los estímulos ambientales y de fuerzas mecánicas que son externos al genotipo general de los árboles. En este sentido, los procesos de optimización de materiales que se producen para contrarrestar las fuerzas de dirección se relacionan con casos de carga particulares en cada tipo de árbol. La capacidad del árbol para adaptarse a las condiciones externas es esencial en condiciones de incertidumbre, en cuanto a un futuro análisis de cargas en casos particulares.

La optimización de los materiales ocurre en las fibras de las estructuras dentro de los estímulos y cálculos de la naturaleza morfo - mecánica en tejidos leñosos, y se traduce en un proceso conocido como transducción, que se caracteriza por el movimiento de los granos de almidón dentro de la célula de la planta que corresponde a la dirección y magnitud específica de la carga aplicada al tejido. Un mecanismo de regulación precede siempre a una respuesta mecánica. La adaptación de la forma depende, naturalmente, del ángulo de desplazamiento desde el eje vertical, así como de la orientación de la fibra que emerge a lo largo del tiempo de procesamiento o vida del organismo particular.

4.2.6 - Niveles Jerárquicos de Adaptación Formal

El mecanismo básico para el cambio de la forma en las plantas se basa en la reestructuración de las estructuras de las fibras en múltiples escalas. La adaptación formal, que se caracteriza por los cambios incrementales en la curvatura, se produce como un laminado no simétrico en los árboles, que responden a las cargas externas que se producen a partir de las fuerzas de compresión y de tensión en las zonas superiores e inferiores de la madera, así como en las zonas superiores e inferiores del tejido propio de la madera. Por lo tanto, los cambios estructurales particulares afectan a la curvatura del sistema total (figura 64) (figura 65).

Tres niveles de adaptación formal pueden ser observados como una reorientación mecánica que se produce en las paredes celulares (nivel local), el tejido (niel regional) y el tronco del árbol (nivel global). En este caso, la manipulación geométrica es dependiente del crecimiento, deposición y reorientación de las estructuras que componen las fibras.

4.2.7 – Micro – Escala del Material

Los sistemas de generación a partir de ramificaciones en maderas blandas y fibras de maderas duras, por ejemplo, proporcionan el soporte mecánico de los árboles. Gibson ha idealizado una estructura celular mediante el modelado único de las ramificaciones y fibras, como un panal de miel de abeja. Los modelos sugieren una complejidad en las orientaciones de las fibras en relación a su incremento, que causa una orientación más paralela a la normal en los ejes de las células (Gibson, 1995). Dado el almacenamiento celular, el módulo de Young para el cálculo de las paredes de la célula, se puede llevar a cabo por medio de la teoría del material compuesto.

Otra característica notable que ilustra Gibson, es la densidad vascular en las fibras de la madera, distribuida a lo largo de una matriz de células de parénquima que componen la mayor parte de las estructuras no leñosas. Las secciones longitudinales revelan una alineación de las fibras en relación a la función mecánica particular de cada árbol. Algunos tejidos, por otra parte, como los tallos de las palmas, tienen una distribución radial no uniforme de los espacios vasculares, con mayor dimensión en el diámetro exterior que en el interior. A diferencia de la madera, los tallos de las palmas carecen de un cambio radical en relación al crecimiento radial que puede proporcionar un apoyo en conjunto con el aumento de altura. En cambio, la palma depende en gran medida del aumento en espesor y el grado de lignificación de las paredes celulares que conducen un número mayor de gradientes de flujos radiales y longitudinales en la densidad del tejido. El tallo de bambú tiene una fracción de volumen radialmente variable de “fibras”, que aumentan su rigidez a la flexión (Gibson, 1995). Las plantas de tallos pueden resistir tanto las cargas axiales (de su propia masa) y los momentos de flexión (cargas de viento), estructuralmente, que actúan como vigas y columnas. Las plantas de tallos son axisimétricas: sus tubos cilíndricos son capaces de resistir la carga generada por el viento para formar cualquier forma ramificada y en cualquier dirección. El pandeo local de las plantas se explica por la presurización interna del tallo y el apoyo de una masa densa en la capa exterior rígida, mediante una base elástica de un material similar a la espuma sintética producida por el ser humano (Gibson, 1995).

De manera más específica, la madera se compone de fibras de celulosa en una matriz de lignina, y hemicelulosa en forma de células prismáticas huecas. Por otra parte, la piel se compone en gran parte de tejidos, formados en su mayoría por el colágeno que se intercambia entre membranas internas y una epidermis de queratina; en los tendones, las fibras de colágeno se encuentran alineadas para formar estructuras similares a cuerdas que constituyen la mayor parte del volumen; y en el cartílago, las fibras de colágeno se encuentran en una matriz de proteoglicanos[44] con una pequeña fracción de volumen de fibras de elastina. El pelo, las uñas, los cuernos, la lana, las escamas de los reptiles y las corazas están hechas de queratina, mientras que la cutícula de los insectos está conformada en su mayoría de quitina. Los huesos, las conchas y las cornamentas de algunos animales son compuestos de calcita, aragonita de hidroxiapatita o plaquetas dispersas en una matriz helicoidal de colágeno (Vogel, 2003).

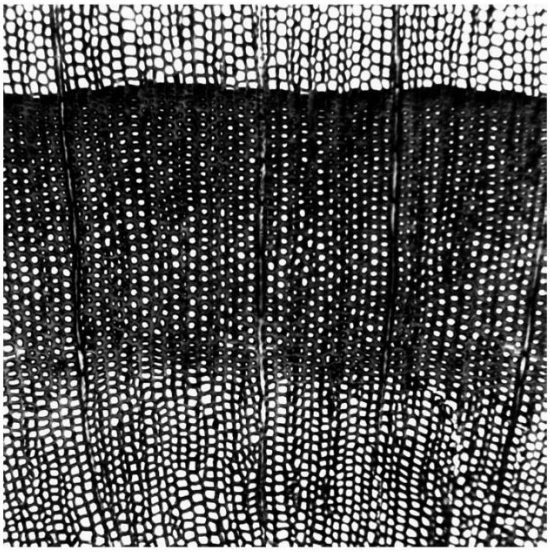


Figura 64. Variación de densidad en la fuerza de compresión aplicada a la madera, ilustrada en un espectro *Juniperus*. Fuente: *http: // www: wsl.ch/staff / janesper / pics / anatomy5_high.jpg*

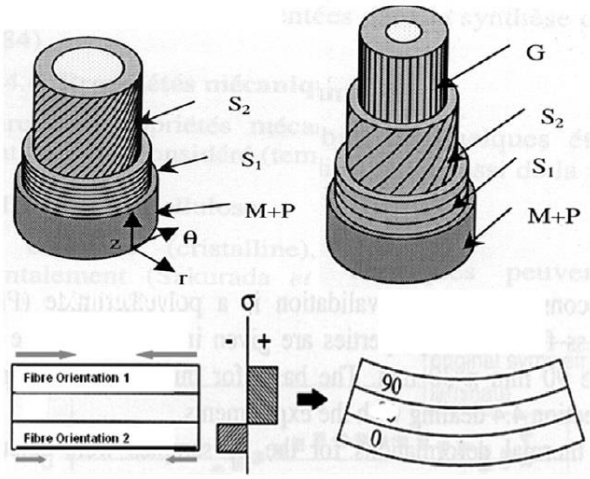


Figura 65. La madera de tensión (derecha) se contrae más que la madera normal (izquierda) durante el proceso de crecimiento y maduración del sistema (árbol). (Clair y Thibaut, 2001; Clair y Thibaut, 2005)

44 Los proteoglicanos son una clase especial de glicoproteínas que son altamente glicosiladas. Las moléculas se encuentran formadas por un núcleo proteico que se encuentra unido covalentemente a un tipo especial de polisacáridos denominados glicosaminoglicanos. Son el componente fundamental de la matriz extracelular animal, constituyen, por así decirlo, la principal sustancia que "rellena" los espacios que existen entre las células del organismo. Aquí forman grandes complejos, tanto con otros proteoglicanos, hialuronanos y proteínas de la matriz fibrosa (como el colágeno). También están involucrados en la unión de cationes (tales como el sodio, potasio y calcio) y agua, y también regulando el movimiento de moléculas dentro de la matriz.

4.2.8 – Las ventajas de la *Distribución del Material* sobre el *Ensamble de Materiales*

Las fibras se pueden ver por todas partes en la naturaleza, y a través de múltiples escalas. Así como los materiales producidos por el ser humano, los materiales naturales también pueden agruparse en clases a partir de un denominador común, ya que todos los materiales naturales están hechos de fibras. Estos materiales incluyen:

1. **Cerámicas naturales y de compuestos cerámicos:** que incluyen a los huesos, las conchas, los corales, el esmalte y la dentina. Todos se componen de partículas cerámicas tales como la hidroxiapatita, calcita y aragonita en matrices de colágeno.
2. **Polímeros naturales y compuestos:** que incluyen a la celulosa, la quitina, la seda, la cutícula, la queratina y los tendones.
3. **Elastómeros naturales:** tales como la piel, los músculos, los cartílagos, las arterias, la resilina y la elastina.
4. **Materiales celulares naturales de fibras:** como la madera, los huesos esponjosos, las palmas, el corcho; que poseen bajas densidades de materia debido a las grandes fracciones de volumen de espacios vacíos que contienen. Casi siempre son anisotrópicos debido a la forma y a la orientación de las células y a las fibras que los conforman; las células prismáticas de la madera, por ejemplo, brindan una mayor rigidez a lo largo del tronco que conforman como parte de un sistema (árbol, por ejemplo).

Las estructuras de materiales compuestos fibrosos en la naturaleza se utilizan principalmente para tres funciones:

1. Para introducir y explotar la heterogeneidad y la anisotropía (local y global).
2. Para modular los tejidos en base a sus propiedades físicas y mecánicas.
3. Para crear formas basadas en estructuras funcionales a través de jerarquías.

Estas tres estructuras se encuentran claramente interrelacionadas en un sistema unitario.

La capacidad de la naturaleza para estructurar sus organizaciones materiales es amplia, ya que consiste en una gran variedad de maneras que corresponden a la eficiencia y eficacia de múltiples funciones que tienen una gran relevancia y ventajas en el mundo natural sobre el mundo artificial producido por el ser humano. Estas ventajas son la anisotropía, la heterogeneidad y la estructuración jerárquica.

4.2.8.1 – Anisotropía

La anisotropía es la propiedad de ser direccionalmente dependiente. Esto implica la heterogeneidad de las propiedades físicas y mecánicas en relación a sus funciones. La anisotropía se puede definir como una diferencia, cuando se mide a lo largo de diferentes ejes, es una propiedad particular de un material físico (nivel de absorción, índice de refracción, densidad de materia, entre otras). En la madera, como se demostró anteriormente, es un material anisotrópico, ya que sus propiedades varían mucho cuando es medida en relación a las propiedades mecánicas de los tejidos y de las células cristalinas que los conforman. Por ejemplo, la fuerza de la madera y su dureza serán diferentes para una misma fuerza si son medidas en diferentes orientaciones axiales. La ventaja en el control de la direccionalidad de las propiedades nos dará resultados físicos y mecánicos variables en estructuras adaptables a su entorno externo y una medida para apoyar una gama de restricciones introducidas a los sistemas por el entorno natural inmediato.

4.2.8.2 – Heterogeneidad

La heterogeneidad permite la generación de las propiedades del material a partir de su posición dependiente. Es aquí, donde la anisotropía es considerada como la de más alto nivel, ya que representa una distribución no homogénea de fibras con múltiples direcciones (figura 66).

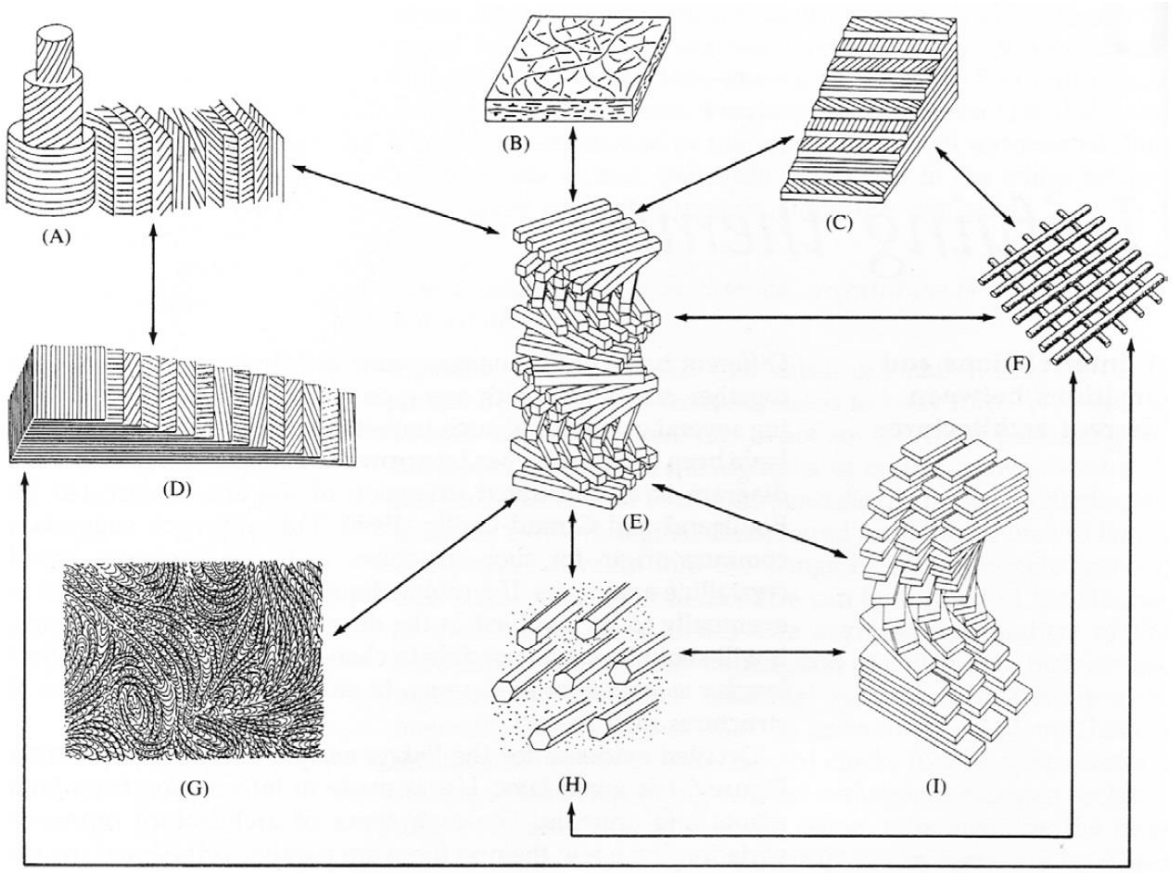


Figura 66. La evidencia de un origen en común de la diversidad de formas de compuestos fibrosos. (A) cilíndrico helicoidal (doble torsión) en la clasificación helicoidal planar (torsión individual), ejemplo: sistema de huesos Haversiano. (B) capa random planar, ejemplo: algunas partes de las células de unas plantas. (C) helicoides a 45 grados, ejemplo: capa de los huevos del salmón. (D) torsión ortogonal, ejemplo: escamas de los pescados. (E) monodominio helicoidal (ángulo pequeño de rotación), ejemplo: cutículas de los insectos. (F) ortogonal y perpendicular, ejemplo: cutículas de insectos con el cuerpo cilíndrico. (G) polidominio helicoidal, ejemplo: proteínas en los cascarones de huevos de una mantis. (H) paralelo – unidireccional, ejemplo: tendones de los artrópodos. (I) pseudo – ortogonales, ejemplo: ramas de madera. (Neville, 1993).

4.2.8.3 – Estructuración Jerárquica

La aparición de estructuras jerárquicas en los tejidos naturales y sus muestras se promueven mediante la estructuración del material anisótropo que resulta en organizaciones heterogéneas, que cuando son superpuestas, mejoran el rendimiento mecánico de los sustratos. Los ensamblajes de la parte inferior hacia la parte superior de las estructuras e interfaces aparecen como múltiples organizaciones de materiales que conforman finalmente superestructuras biológicas en macro y micro escalas particulares.

4.2.9 – La Dicotomía Natural hacia el Diseño por la Distribución Material

La distinción entre los materiales y las estructuras en el contexto del diseño no es trivial. Se encuentra presente como una figura de la medida relacionada con las escalas de la función y el comportamiento, así como del aspecto formal inherente a sus propiedades estructurales. Cuando proporcionamos escala a un objeto adquiere un nuevo significado y da sentido nuevo al contexto y medio ambiente en el cual está inmerso. En este sentido, la escala nos permite ir más allá de las representaciones formales que siguen siendo otra manera abstracta de categorización: con la escala viene implícito un compromiso con la física, las matemáticas, la computación, el diseño y la biología.

En su ensayo seminal *On Magnitude*, el autor D'Arcy Thompson argumenta que cualquier concepción de la forma debe ser referido en términos de magnitud y dirección "para el efecto de escala, que dependerá no solo de un objeto por sí solo, sino de la relación de él con sus alrededores ambientales y físicos" (Thompson, 1952). De ahí que cualquier manifestación de forma debe examinarse no como una mera representación, sino como (y literalmente) una entidad incrustada dentro de un entorno determinado, ya sea físico o programático. Dado que estaré tratando objetos referidos a partir del diseño, que son de naturaleza física, la diferencia entre el "material" y la "estructura" es fundamental para una lógica referida a la organización material, determinada por el rendimiento del sistema, basado y transferido a un lenguaje de programación gráfico con el fin de su representación geométrica.

Al analizar, referir y tratar con temas de la biología en relación a un punto de vista a partir de la ingeniería, Julian Vincent afirma que la madera o las hojas de los árboles tienen una "estructura" obvia cuando son tratados principalmente a partir de la propiedad de materia. La solución a este problema, de acuerdo con el autor, por lo general se encuentra en el comportamiento[45]: cuando se trata de un material, entonces se tendrá la misma rigidez en la fuerza de tensión, en la flexión y en la compresión en todas las direcciones axiales. Si no es así, la estructura particular que investiguemos, trataremos erróneamente con la rigidez que se está midiendo, el cual no es un parámetro material (Vincent, 1982). Los materiales se clasifican típicamente por sus propiedades en conjunto, tales como la resistencia, la densidad, la dureza, la elongación, la resistencia eléctrica y el contenido de energía propia, mientras que las estructuras se clasifican por el comportamiento físico – mecánico (Gordon, 1976). Por ejemplo, algunas estructuras son buenas para soportar fuerzas de compresión (conchas, ramas de árboles), mientras que otras son buenas para soportar fuerzas de tensión (membranas, estructuras a base de cables, entre otras).

45 En este aspecto puede haber también excepciones, ya que algunos cristales pueden ser anisotrópicos.

4.2.10 – La Dicotomía entre la Forma y el Tamaño

La forma y el tamaño son dos de las inquietudes tradicionales que tienen los morfólogos al comparar estos dos con el aspecto funcional, ya que la forma – material resultante está relacionada intrínsecamente con el aspecto de la carga del propio objeto. La relación entre la forma y el tamaño es tal, que el tamaño de un organismo dicta el peso que debe de ser soportado por la estructura, mientras que la forma del organismo define las áreas en la sección transversal a través de las cuáles se encuentra distribuido el peso. Del mismo modo, la distribución espacial de los materiales dentro de un objeto se traduce en la anatomía de una estructura biológica, que define los segmentos locales que resultan de una carga aplicada. La forma y el tamaño de este modo, influyen en las magnitudes relativas a las tensiones que se desarrollan en el interior de los miembros de soportes estructurales, tales como los tallos de la planta y en las hojas tubulares. La importancia de la forma y el tamaño para la biomecánica del sistema natural es evidente si consideramos los parámetros de la ingeniería denominados momentos de área, que son descripciones matemáticas de la distribución espacial del material dentro de un objeto (Niklas, 1992).

Por otra parte, en relación a las características físicas de los organismos, el aspecto formal solamente se aproxima al tamaño en términos de cambios internos; y la forma resulta mucho más difícil de reducir a una o a unas variables numéricas mínimas o máximas con pocas especificaciones. La mayor cantidad de datos pendientes detrás de los exponentes numéricos referidos a la escala, son dependientes de la variabilidad de la forma final y no de sus procesos de cambio a través de la disminución o incremento de la escala. ¿Cómo lidiar con la forma en nuestra búsqueda de reglas generales aplicadas a la organización biológica y la operación? Algunas condicionantes de formas que han demostrado ser útiles son: centro de gravedad (o centro de masa), momentos de área, centros de empuje (centro del volumen), dimensión fractal, índice planar, entre otros.

4.2.11 – La Dicotomía entre la Estructura y el Material

Dado que las estructuras durables y rentables son el resultado de negociaciones llevadas a cabo a través de miles de años, los materiales biológicos no suelen hacer una distinción racionalizada entre el material y la estructura (Rogers, 1993). Además, en la naturaleza, la distinción entre los sistemas de materiales y estructuras inteligentes es básicamente irrelevante. De hecho, sólo se puede aplicar superficialmente el término referido a la escala de macro y micro estructuras de un material que rige su comportamiento, independientemente de su tamaño (Rogers, 1993).

Sin embargo, tal distinción entre la estructura y el material se ha enfatizado en repetidas ocasiones, debido a que el comportamiento mecánico de una estructura se puede entender sólo cuando su geometría y las propiedades del material, a partir del análisis de sus constituyentes sólidos o líquidos, se consideran como un conjunto. Por ejemplo, la geometría en la pared de una célula dentro del tejido de una planta puede, en algunos casos, reemplazar en importancia las propiedades del material en relación con la estructura interna. Por lo tanto, podemos disociar el comportamiento mecánico de un tejido en su misma configuración anatómica (Niklas, 1992).

El término de hueso se refiere a una familia similar de materiales, los cuales son generados a partir de fibrillas de colágeno mineralizadas. Tienen estructuras muy complejas, que se describen en término de hasta siete niveles jerárquicos de organización. Estos materiales han evolucionado para cumplir una variedad de funciones mecánicas, a las cuales se han

tenido que adaptar formalmente. La relación entre la estructura y la función es un gran reto para los organismos y para el razonamiento y aplicación en el diseño por parte del ser humano (Weiner y Wagner, 1998).

4.2.12 – La Dicotomía entre el Tamaño y la Escala

Los organismos ocupan una extensa gama de tamaños en la naturaleza en una proporción de alrededor de cien millones de veces[46]: desde las bacterias más pequeñas de alrededor de 0.3 micrómetros de largo a las ballenas de una longitud aproximada general de 30 metros. Vogel ha determinado que al momento de aproximarse a cómo el tamaño afecta al diseño biológico, nada es más importante que la relación entre la superficie y el volumen (Vogel, 2003). Esto se debe principalmente al hecho de que cualquier contacto entre un organismo y su entorno son una función de superficie, mientras que sus procesos y su estructura interna depende de manera predominante de su volumen[47]. Sin embargo, en la naturaleza, la superficie y el volumen no mantienen una proporción sencilla. En otras palabras, no se puede doblar simultáneamente tanto el área superficial y el volumen de un cuerpo sin cambiar su forma[48] (Vogel, 2003). En otros términos, la forma de un organismo determina la relación con el medio ambiente y es funcionalmente significativa entre las áreas de superficie a volumen.

Las escalas operativas en los sistemas biológicos han evolucionado con el fin de establecer una coincidencia con escalas naturales y jerarquías que a menudo se repiten en la naturaleza. En la variedad de un hueso adulto laminar[49], por ejemplo, las barras interconectadas se conocen como trabéculas, y las placas del tejido óseo, se redimensionan y son remodelados por los osteoclastos (células óseas que destruyen) y los osteoblastos (células formadoras de hueso nuevo). En la micro – escala, existe una similar estructura de madera “contrachapada” de capas (láminas) de aproximadamente tres milímetros de fibra de colágeno. Los cristales de sal de hueso (apatita de carbonato impuro) llenan gran parte del espacio de agua en la matriz del colágeno, ya que forman las estructuras con mayor jerarquía de los huesos. Del mismo modo, cuando examinamos a simple vista y con el lente del microscopio, las fuerzas de unión atómicas en una pieza de acero o de cerámica, se mostrará una jerarquía de escalas. Con el fin de aumentar la magnitud de escala que incluyen las superficies en relación a su rugosidad, las estructuras de grano se observan con ciertas dislocaciones e inclusiones de otras fases de desarrollo, las regiones de carga espacial o atmosferas de impurezas alrededor de las dislocaciones o interfaces segregadas, y por último, las estructuras atómicas se desvelan a la simple y poderosa vista al salir a la luz (figura 67). Las características mesoscópicas son las que tienen escalas muy características entre la atómica y la macroscópica, y son importantes al analizar el comportamiento real de los materiales (Harding, 2009).

46 El rango indicado es el equivalente a ocho órdenes de magnitud. Cuando comparamos la industria de la construcción se podría considerar un rango de solo tres órdenes de magnitud, que van desde un grano de área a un ladrillo, por ejemplo.

47 Esto es también muy común en los edificios o en la industria de la construcción.

48 Aquí es importante señalar que las superficies son proporcionales al cuadrado de longitudes; los volúmenes son proporcionales a las longitudes de los cubos.

49 También conocido como esponjoso (o trabecular) en los huesos.

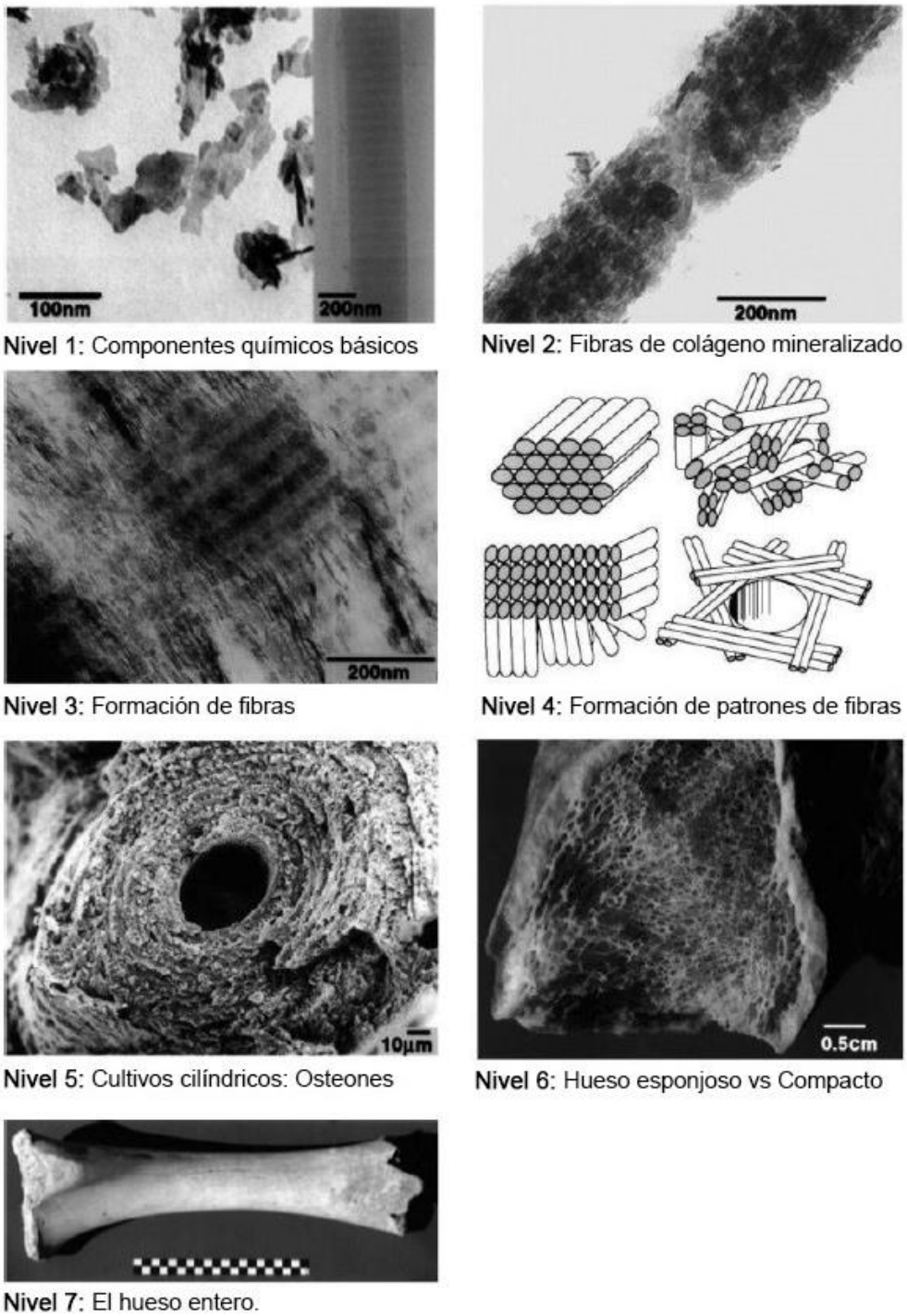


Figura 67. Los 7 niveles jerárquicos de la organización de generación en un tipo de hueso. Nivel 1: el aislamiento del mineral de calcio a partir de hidroxiapatita es separado del fosfato de otro hueso humano. Nivel 2: las fibras de colágeno son convertidas en secciones delgadas de fibras. Nivel 3: las secciones de fibras son mineralizadas. Nivel 4: son producidos cuatro modelos de matrices o patrones de fibrillas de organización que se encuentran en la familia de los materiales del hueso. Nivel 5: la micrografía muestra una sola generación por parte de los osteones en cultivos de tejido cilíndrico. Nivel 6: La luz pasa a través de la micrografía para mostrar una sección fracturada a través de un fósil en un hueso esponjoso de un fémur humano de unos 5 550 anos. Nivel 7: un hueso bovino total (escala: 10 cm). (Weiner y Addadi, 1997; Weiner y Wagner, 1998).

La idea de una escala adecuada para un sistema natural y la propiedad dada surge repetidamente en estos sistemas. Además, puede haber varias de estas escalas, dependiendo de las propiedades físicas – mecánicas y biológicas – químicas de los organismos animales o vegetales. En un tronco de árbol, la escala adecuada para la fuerza difiere de la escala adecuada para el transporte de agua. Para unas gafas de sol fotocrómicas, la escala de los coloides metálicos que contribuyen al oscurecimiento es mucho menor que la escala de la robustez mecánica. La identificación de las escalas operativas adecuadas es un primer paso en el camino hacia la optimización de mesoestructuras que guiarán a un sistema natural o artificial para adoptar una escala adecuada (figura 68).

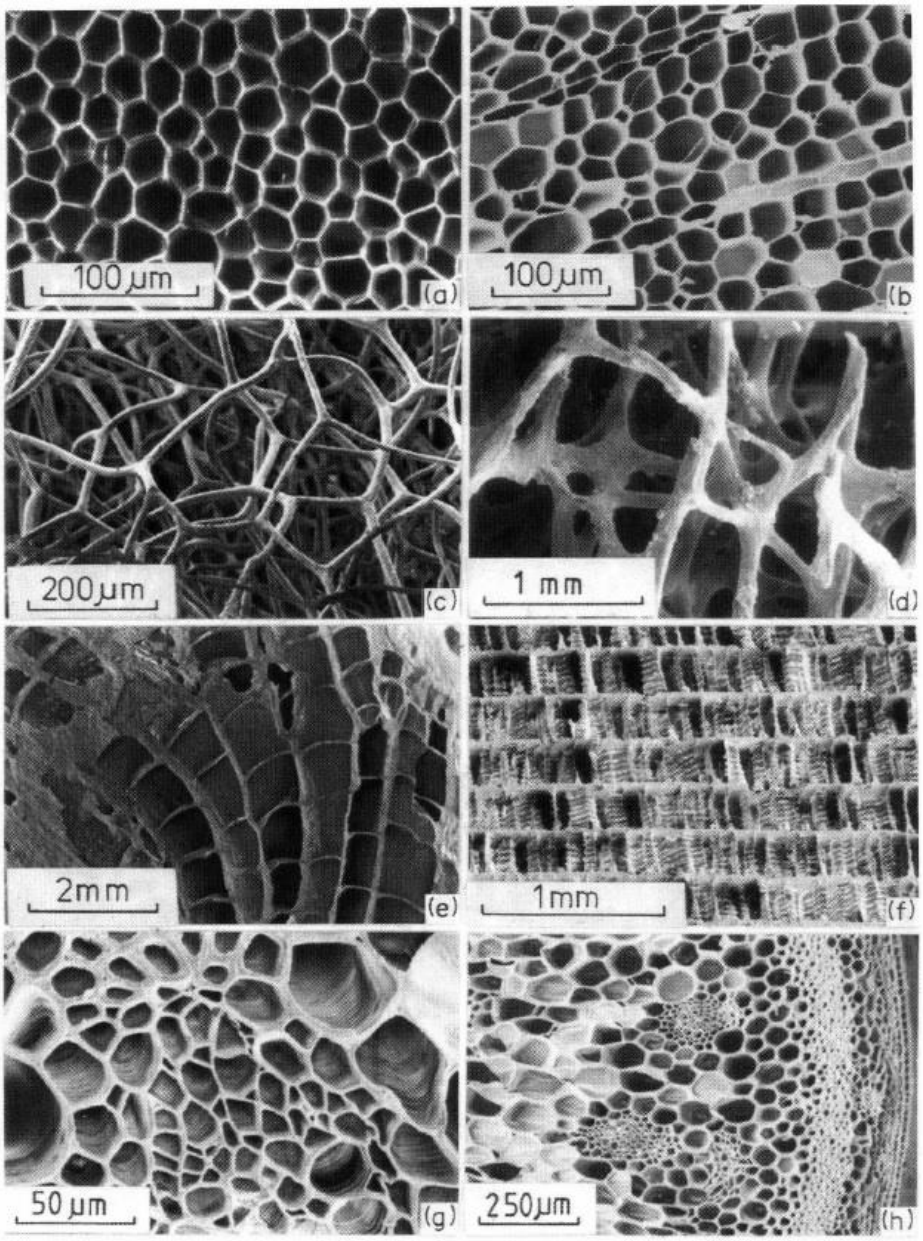


Figura 68. Materiales celulares naturales, incluyendo (de la izquierda a la derecha y de arriba hacia abajo): (a) corcho, (b) madera de balsa, (c) hueso esponjoso, (d) hueso trabecular, (e) tejido en ramificación de un coral, (f) tejido cutáneo de un calamar, (g) una hoja de iris y (h) una hoja de una planta. Los pequeños cambios en ángulo, dirección, proporción y densidad acaban repercutiendo en la forma final del ente del que se trate. (Gibson y Ashby, 1997).

4.3 – La Evolución Material aplicada al Diseño

Afirmar que la prioridad en las propiedades de los materiales aplicados al diseño es un descubrimiento reciente, sería profundamente erróneo. La historia de la ciencia de los materiales y la ingeniería es un testimonio de la producción cultural y un protocolo de la forma en la que los seres humanos han interactuado con su entorno en busca de un refugio, salud y bienestar (Ashby, 1995). La importancia en el empleo de los materiales en el desarrollo de las primeras civilizaciones se encuentra registrado en la manera que la arqueología ha dividido a los avances tecnológicos en épocas definidas por las propiedades endémicas a ellos: la Edad de Piedra, la Edad de Bronce, y la Edad de Hierro. La actualidad en la cual nos encontramos viviendo podría ser descrita como la Edad de Materiales, como resultado de una serie de investigaciones y tecnologías desarrolladas a lo largo de los últimos treinta años, que han alterado radicalmente la forma de hacer, construir y vivir (Ashby, 1995). De hecho, uno de los precedentes más interesantes en el diseño de materiales hasta hoy en día es la evolución y producción de materiales compuestos en todas las clases y escalas. Los compuestos son el capítulo fundamental en la historia de la tecnología del siglo XX, como se demostró en la exposición *"Mutant Materials in Contemporary Design"* de Antonelli, en el Museo de Arte Moderno de Nueva York, hace más de dos décadas (Antonelli, 1995). Sin embargo, incluso con las anteriores investigaciones y tendencias emergentes de una cultura del diseño material, el problema es muy complejo, ya que, para dar inicio, el diseño como fenómeno cultural aún perdura y es característico de la teoría y la práctica en el diseño contemporáneo.

Las obras literarias relativas al diseño son muy amplias en número, aunque hechas por investigadores, que la mayoría de las veces, están relacionados con la disciplina, además de que existe una fragmentación abrumadora en cuanto a postulados y tendencias grupales y personales. Un estudio abierto y profundo de este campo indica una amplia gama de áreas de investigación, ya sea la representación, la exploración, la clasificación, entre otras. Existen muchos puntos de vista científicos y de diseño, que pueden incluir características físicas y de comportamiento de materiales, e incluso obtener interpretaciones propias basadas en estas dos premisas. Sin embargo, lo más importante, es que cada uno de estos puntos de vista y opiniones pueden ser útiles de manera particular, se pueden aplicar a una construcción teórica propia para poder ser aplicados a un caso de estudio determinado (Addington y Schodek, 2005).

La amplitud de interés y actividad en relación a los nuevos e innovadores materiales, además de las tecnologías, en la actualidad son influencias dominantes en lo que respecta al diseño arquitectónico e industrial contemporáneo. Kwinter se ha referido a los materiales como los "nuevos espacios" (Kwinter, 2001). Esta tendencia es tan prominente, que el impulso para la innovación en la producción de los materiales en los últimos años ha absorbido, en términos generales, a la industria del diseño y la investigación relacionada a la práctica como medio generador principal (Baldmond, Smith, 2001; Addington y Schodek, 2005). Las profesiones relacionas con el diseño en su conjunto han experimentado un "renacimiento" de los materiales en los últimos años. Además de las publicaciones claves y extractos pertinentes en artículos de revistas especializadas, exposiciones y publicaciones que destacan el papel importante del uso de los nuevos materiales en las tendencias de diseño más innovadoras. En particular, el interés por el diseño y por la experimentación con materiales compuestos y la aplicación de materiales de tipo textil al diseño (McQuaid y Beesley, 2005) han sido abundantes y exponen la necesidad de consolidar nuevos métodos y nuevas prácticas.

Históricamente, existen dos maneras diferentes para influir en el diseño. La primera es la rápida absorción como un fenómeno estilístico y formal, mientras que la segunda es más profunda en la exploración de complejas interrelaciones entre la ciencia y la tecnología aplicada al diseño que ha surgido en la última década (Antonelli, 1995; Antonelli, 2008).

La literatura sobre el tema de “materiales inteligentes” y “sistemas de materiales” es vasta y ha acumulado a lo largo de la última década, la incorporación de denominaciones tan variadas como los contextos disciplinarios a partir de los cuales se originaron (Gordon, 1976; Addington y Schodek, 2005; McQuaid y Beesley, 2005; McQuaid; Beesley, 2005). Sin embargo, existen dos paradigmas prevalentes que pueden usarse para clasificar a la mayoría de tales designaciones (Rogers, 1993). La primera definición se basa en el paradigma tecnológico y se relaciona con los actuadores, sensores y controles incrustados dentro de un material o componente estructural. Tal como la definición lo específica, los elementos que comprenden el material son analizados, además de investigar su utilidad y los medios por los cuales ha sido fabricado o ensamblado. La segunda definición se basa en un paradigma científico, y trata de relacionar, de manera predominante, a la microestructura del material con la capacidad de reducción de materia, además de la adaptación a las limitaciones energéticas y a la forma en la cual ha sido producido. Una investigación detallada de fuentes literarias especializadas, puede comunicar una rápida relación muy conocida en el campo de la arquitectura y el diseño, mediante el cual los materiales se encuentran seleccionados de manera típica, pero no son diseñados. Varias rutinas de selección de materiales y formas de clasificación, han evolucionado a lo largo de los años y para la aplicación teórica y práctica de varias disciplinas. La elección del material no se puede realizar independientemente de la elección del proceso por el cual se produce el material, desde su componente químico más sencillo hasta el acabado final del material (Ashby, 1995; Ashby, 2005). Del mismo modo, el renacimiento referido a la conciencia del material se asocia muy probablemente con el acelerado ritmo de las tecnologías de producción de materiales de construcción, que se relacionan de manera directa con otras disciplinas, tales como la ingeniería, la química, la biotecnología, la computación y las ciencias de la información (McQuaid y Beesley, 2005). Esta relación de conocimientos aplicados al diseño es indispensable para encontrar rápidamente un camino para la formulación de preguntas relativas a la generación de la forma.

4.3.1 – El síndrome del componente: Las Expresiones de una Crisis Cultural en el Diseño

4.3.1.1 – La Forma como Cuerpo Icónico

Hablando de manera “arquitectónica”, cuando pensamos en la “forma”, pensamos directamente en el aspecto físico de un objeto; además de su descripción en términos de geometría. Durante miles de años, las estructuras más grandes del planeta Tierra habían sido las pirámides de Egipto, pero rara vez se refieren a la distribución de las cargas cuando se describe al tetraedro monumental. La naturaleza de esta afirmación sigue siendo constante a través de los periodos del antiguo Egipto hasta el Cairo contemporáneo. La forma es una categoría de descripción en oposición a una condición causal, ya que podría ser descrita a partir de la naturaleza.

Además, es conocido en la práctica arquitectónica, que los espacios preconcebidos de diversas escalas, formas y magnitudes han sido convencionalmente designados por clases genéricas de funciones geométricas y matemáticas. Para desafiar a esta “tradición” en la toma de decisiones “clásica”, los arquitectos han empezado a explotar las tecnologías emergentes de la industria de la construcción, con el apoyo de una parte en las invenciones de producción de los materiales. Y, de hecho, la hipótesis general que prevalece, entre muchos arquitectos y diseñadores, es que todo lo que es imaginable es edificable. Nos acercamos a una era en que esta hipótesis no está demasiado lejos de la realidad. Pero junto con ello, hay consecuencias menos emocionantes e inspiradoras, sobre todo cuando consideramos al medio ambiente. Lo anterior, puede apasionar la búsqueda del arquitecto en relación al aspecto formal y en la sostenibilidad ambiental. Un enfoque basado en materiales para diseño y en los objetos virtuales de diseño, podría reemplazar la actual crisis ambiental, y tener un impacto significativo en el medio natural del cual el ser humano es parte, a través de una reformulación de estos síndromes dañinos actuales en la industria de la construcción.

4.3.1.1 – La Forma Ensamblada: El Hábitat como Máquina para producir Obsolescencia

Los edificios a partir del ensamble de componentes o partes se encuentran en todas partes. Ellos son los medios por los cuales se describen y construyen manifestaciones simples formales. Dado que aún no podemos “cultivar” literalmente un diseño y edificio posterior, como lo hace la naturaleza, hemos optado por ensamblar las partes. Los ensamblajes de los materiales en la industria de la construcción se han realizado desde hace muchos siglos antes de la Revolución Industrial, sin embargo, a finales del siglo XVIII cultivó las semillas de los componentes, así como la lógica de la producción en masa de productos estándar.

Con la producción en masa, se desarrollaron ideas de diseño que concibieron el hábitat humano como mecánico, es decir, como una máquina viviente. Sus ventajas eran claras y económicamente viables: todas las piezas podrían ser replicadas, reemplazadas y volver a montarse para la adaptación al cambio. Sin embargo, esta concepción lleva consigo ciertos inconvenientes graves, que pronto iban a perjudicar el medio ambiente natural. La redundancia en la producción de materiales de construcción y los desechos que traía consigo solo son un pequeño segmento de lo que podría considerarse una patología condicional de la producción industrial. La industria de la construcción se puede sentir todavía identificada con los valores impuestos del sistema de la Revolución Industrial: estandarización, homogenización, modulación, reducción y repetición. Sin embargo, la

mayoría de las características anteriores de la industrialización en la producción en masa, son el camino “antiético” a la naturaleza. Por lo tanto, no es de extrañar la sorpresa de que la economía y la cultura relacionada con el diseño de la obsolescencia se encuentran en profunda contradicción a la investigación contemporánea de una práctica de diseño sostenible de verdad.

La forma que es controlada por el diseño y el control de los materiales a través de múltiples escalas, puede permitir acercamientos a respuestas ambientales, de estructura, de función y de no – forma predeterminada a través de una crítica y reflexión de los conceptos anteriormente descritos.

4.3.1.2 – La Selección: Las Expresiones Metodológicas de un Síndrome Cultural de Diseño

El material que se incorpora típicamente a un edificio juega un papel secundario en la expresión referida a la forma, esta consideración de “envolvente del cuerpo” no es de extrañar. Como tal, un material es característicamente seleccionado a partir de una variedad de posibles opciones, sea por sus propiedades de desempeño, o sub-formas preconfiguradas. Ashby menciona que la selección y proceso de producción de un material no puede estar ajeno a la elección (Ashby, 1995; Ashby, 2005). Como menciona el autor, se tiene que hacer razonable, interpretar y entender que cada material presenta propiedades únicas y ofrece posibilidades particulares al diseñador, así como también constantes estructurales y geométricas independientes. Sin embargo, esta lógica de la forma sobre la materia, aún se continúa manteniendo.

4.3.1.3 – Entre el Material y la Envolvente Formal

Existen varias maneras para poder clasificar los materiales. Las clasificaciones de la ciencia y la ingeniería de los materiales típicamente se parecen en relación a la composición y propiedades intrínsecas del material. Lo anterior se refleja en clasificaciones llamativas que proveen una manera de describir las propiedades específicas o calidades de un material, tales como la rigidez, la dureza, la conductividad eléctrica, entre otras; que difieren de un objeto a otro[50]. Consecuentemente, mientras los científicos que se encargan del estudio de los materiales relacionando las estructuras atómicas de la materia o las escalas moleculares y sus propiedades macroscópicas, el ingeniero estructural se encarga del comportamiento material. En la selección de un material para una función dada, el ingeniero debe medir en base a categorías tales como el estado de la materia, la estructura, las técnicas de producción, las condiciones ambientales y sus aplicaciones (Addington y Schodek, 2005).

50 El Volumen Representativo de un Elemento (RVE, por sus siglas en inglés), es una manera de clasificar las propiedades de un material dentro del volumen singular del elemento. Este concepto es importante en relación a los mecanismos y la física “indeterminada” en la heterogeneidad de los materiales, que puede proporcionar un acercamiento para conocer sus propiedades efectivamente. El tamaño del RVE está asociado con la estimación de las propiedades de los números de iteraciones de procesamiento en un volumen dado (V) en una microestructura que puede ser considera para este tipo de análisis. Esto demuestra que la dependencia morfología del objeto estudiado, o su propiedad física están interrelacionados de manera intrínseca; este contraste se puede llevar a cabo con los materiales compuestos, y con sus respectivas fracciones volumétricas. En los diferentes tamaños de los RVE se pueden encontrar un rango muy amplio de propiedades físicas, incluyendo constantes elásticas de electricidad y conductividad térmica, así como también propiedades geométricas como fracciones de volumen o fractales tridimensionales. “El volumen representativo del tamaño de un elemento para un estudio de componentes elásticos: es un cálculo numérico” - *Journals of the Mechanics and Physics of Solids* - (Gusev, 1997).

Las tablas de selección de materiales por sus propiedades elaboradas por Michael Ashby proporcionan al diseñador y al ingeniero unas plantillas altamente eficientes en cuanto a clasificación (figura 69). En la selección de materiales del autor, realiza una clasificación con aplicación al diseño mecánico (Ashby, 2005), e introduce una herramienta de diseño para la selección de los materiales en la ingeniería. Además, proporciona una recopilación muy coherente y sistemática de los materiales y sus aplicaciones en el diseño.

Ashby clasifica la familia de los materiales de ingeniería en seis familias que incluyen a los metales, polímeros, elastómeros, cerámicas, vidrios e híbridos. Los miembros de las familias tienen un número de características en común, tales como propiedades, rutinas de procesamiento y frecuentemente aplicaciones similares. La familia de los materiales híbridos incluye combinaciones de dos o más materiales en una configuración de escala predeterminada. Ellos incluyen compuestos de fibras y partículas, estructuras uniformes, estructuras de cables, plásticos, cables y laminados. Los compuestos de fibras reforzadas son, por supuesto, los más comunes e increíbles por el hecho de poder realizar combinaciones de ligereza, rigidez y resistencia.

Las tablas y gráficas que realizó Ashby fueron desarrolladas como una respuesta a la necesidad de los diseñadores en relación a un eficiente proceso de selección de materiales, basado en varios criterios. A partir de que lo anterior, es un caso que se presenta en las ocasiones que los diseñadores están interesados en una combinación de propiedades diversas en una propiedad particular; cada material es presentado como una lista de atributos independientes representativos de un objeto. De esta manera, los rangos que van desde la ligereza hasta la rigidez máxima, pueden ser considerados para la incorporación en otros rangos de algún material de características similares o diferentes. Las gráficas resultantes son útiles ya que resumen un gran cuerpo de información en esquemas compactos accesibles, además de que revelan las correlaciones entre las propiedades de los materiales que indican y proporcionan datos estimados a lo largo de valores numéricos de características (Ashby, 1995; Ashby, 2005).

4.3.1.4 – De la Selección a la Formación

Los materiales por sí solos, sin embargo, requieren de la aplicación de tecnologías contemporáneas, materiales y técnicas que los puedan procesar dentro de formas deseadas en base a su función. En otras palabras, la manera por la cual un material puede ser tratado, afectará directamente su desempeño y sus características funcionales. La distinción entre *material* y *materia* trae consigo mucha relevancia en este contexto, siendo este último asociado tradicionalmente con la noción de una sustancia uniforme que puede hacerse o producirse en cualquier forma deseada (Smith, 1980; Smith, 1981). La manera de “hacer” o “estructurar” la materia, implica un conocimiento profundo de las propiedades de los materiales y sus capacidades de manipulación física.

Los avances recientes en la ingeniería y en la ciencia de materiales han introducido una noción de diseño a partir del comportamiento material. En el campo de los materiales compuestos, específicamente referido a la ciencia material, ha aumentado la capacidad del diseñador para influir y controlar el comportamiento del material. Todas las industrias son igualmente inspiradoras y afectan a esta rama de conocimientos, mientras que la mayoría de los campos del diseño están contribuyendo a la generación de propuestas estimulantes relacionadas, tales como la arquitectura, la construcción, la ingeniería civil, el transporte, la medicina, la agricultura, los deportes y la moda, que están imponiendo retos apremiantes sobre el estado de la reinterpretación del diseño en función de los materiales.

Como herramientas para la generación, producción y construcción de la forma, los materiales deben de convertirse en parte integral de los ambientes computacionales aplicados a los campos del diseño. A través del concepto de comportamiento de los materiales, actualmente se están desarrollando nuevas interpretaciones en relación de las cualidades físicas y a la relación implícita que puede ser aplicada a un proceso de diseño general o particular.

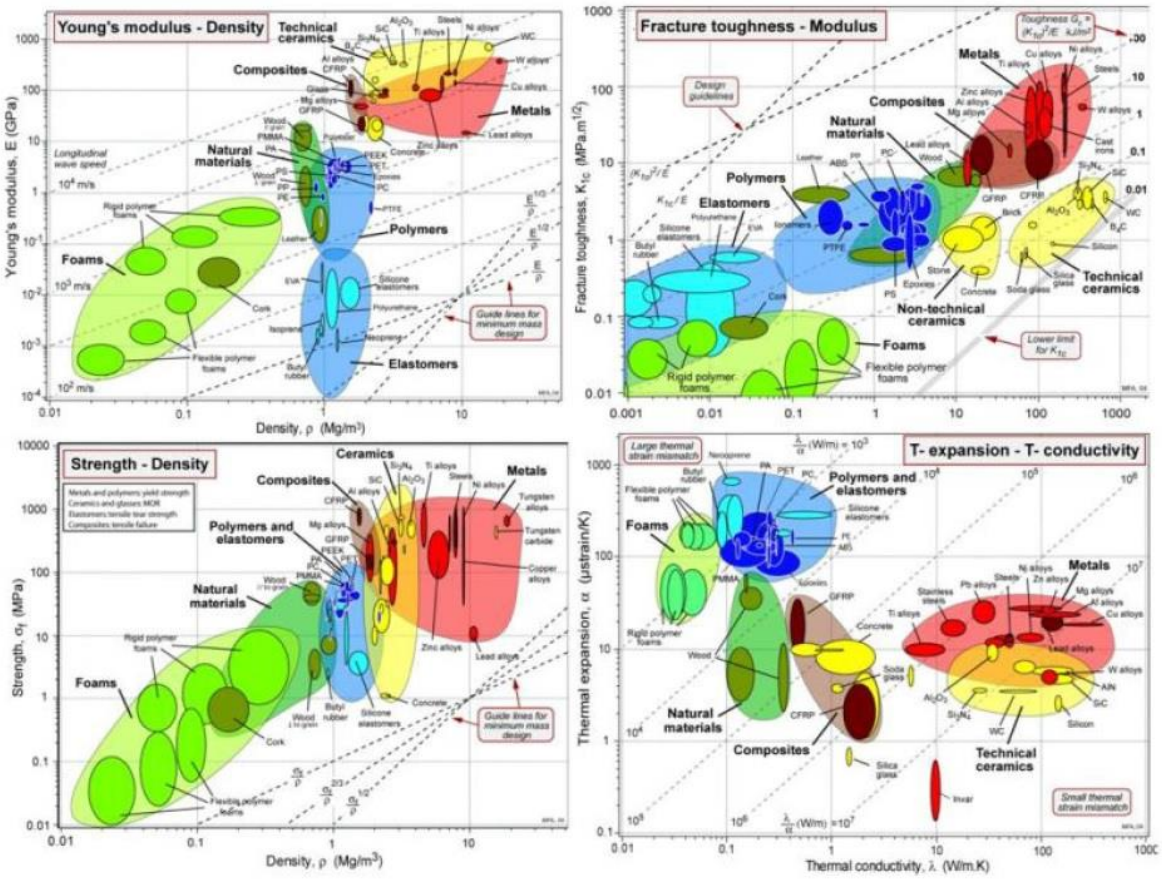


Figura 69. Las tablas y gráficas de Michael Ashby de selección de materiales (de derecha a izquierda, de arriba hacia abajo): tabla de la densidad del módulo de Young, gráfica de módulo de dureza y fractura, gráfica de la densidad de resistencia y la gráfica de expansión y conductividad de los materiales. *Fuente: http: //www.grantadesign.com / Ashby.htm.*

4.3.1.5 – Lo Implícito en el Diseño Computacional - Material

Los materiales biológicos muestran un gran control estructural basado en jerarquías, diseñados y adaptados a condiciones específicas de carga, incluyendo el uso recurrente de componentes fundamentales tales como la cristalografía controlada, la anisotropía e isotropía, la orientación de los elementos naturales, gradientes, interfaces de relación mecánica de amplia duración entre materiales de diferentes formas (compatibles, dúctiles y rígidos), (Addadi, 1997; Weiner y Wagner, 1998; Currey).

Los componentes fundamentales de los materiales biológicos normalmente exhiben propiedades débiles mecánicas a macroescala (cerámicas biológicas frágiles y macromoléculas incompatibles), sin embargo, son capaces de lograr órdenes para el incremento de magnitud en la fuerza de resistencia y rigidez; la “amplificación” de esta propiedad mecánica se produce de una manera no aditiva que va más allá de reglas preestablecidas sencillas relacionadas con la unión (West y Ashby, 2004).

A lo largo de este capítulo he optado por centrarme en las relaciones estructurales y de propiedades de materiales continuas referidas a la macro y microescala compuestas, inspiradas y observadas en la biología. Hemos visto que muchos sistemas naturales reúnen múltiples capas de diferentes microestructuras, que pueden formar figuras geométricas de órdenes superiores que culminan en diseños jerárquicos en macroescala y que trabajan en conjunto con las propiedades de los materiales inherentes para lograr una función mecánica requerida. En particular, me he centrado en las estrategias diseñadas por la naturaleza para promover y prestar atención a la heterogeneidad y a la distribución a nivel del material (a través de la anisotropía, la heterogeneidad y las jerarquías), a nivel de estructura (a través de la adaptación formal junto con los cambios dimensionales y topológicos) y, finalmente, en el nivel de sistema (a través del comportamiento emergente y la adaptación). También he mencionado que tales rasgos distintivos entre el “material”, la “materia” y la “estructura” son a veces muy difíciles de analizar y aplicar a un proceso de diseño determinado (figura 70).

Las implicaciones de diseño en la estructuración de diferencias, o heterogeneidades estructurales se pueden lograr a través de la distribución controlada de fibras, lo cual sostiene implicaciones significantes desde el punto de vista del diseño. La orientación de la estructura de fibras permite un rango de posibilidades casi ilimitado que puede ser aplicado a casos en los diferentes campos del diseño en términos de variación geométrica y topológica, promueve altos niveles de integración funcional a través de la asignación de propiedades graduales, es compatible con las correspondencias entre la distribución de las propiedades químicas materiales y las rutas físicas de cargas continuas, que permite al diseñador considerar la oportunidad de desarrollar respuestas de adaptación, junto con posibles exploraciones de estrategias ilimitadas para la optimización, y una fácil integración de funciones de detección y adaptación a un entorno natural.

Como he redactado en los ejemplos anteriores, así como en los dos ejemplos naturales para fundamentar el diseño posterior, de huesos y de los árboles, las fibras representan elementos de líneas físicas que ofrecen caminos para la transición, transferencia y difusión de la información mecánica y química en las estructuras. Los grupos de fibras pueden ser organizados en uno, dos y tres dimensiones para crear equivalentes físicos de líneas, superficies o sólidos, como los que se pueden obtener a partir de la producción de fibras textiles (que preservan la movilidad de las fibras) o tecnologías de materiales compuestos (dónde las redes fibrosas se rigidizan al estar unidas).

A medida que nos desplazamos desde el mundo de los materiales naturales hacia los materiales artificiales producidos por el ser humano, la búsqueda de nuevas estrategias que se ajusten a los caminos del diseño sostenible, la ruta de la naturaleza seguirá siendo una fuente reveladora de inspiración.

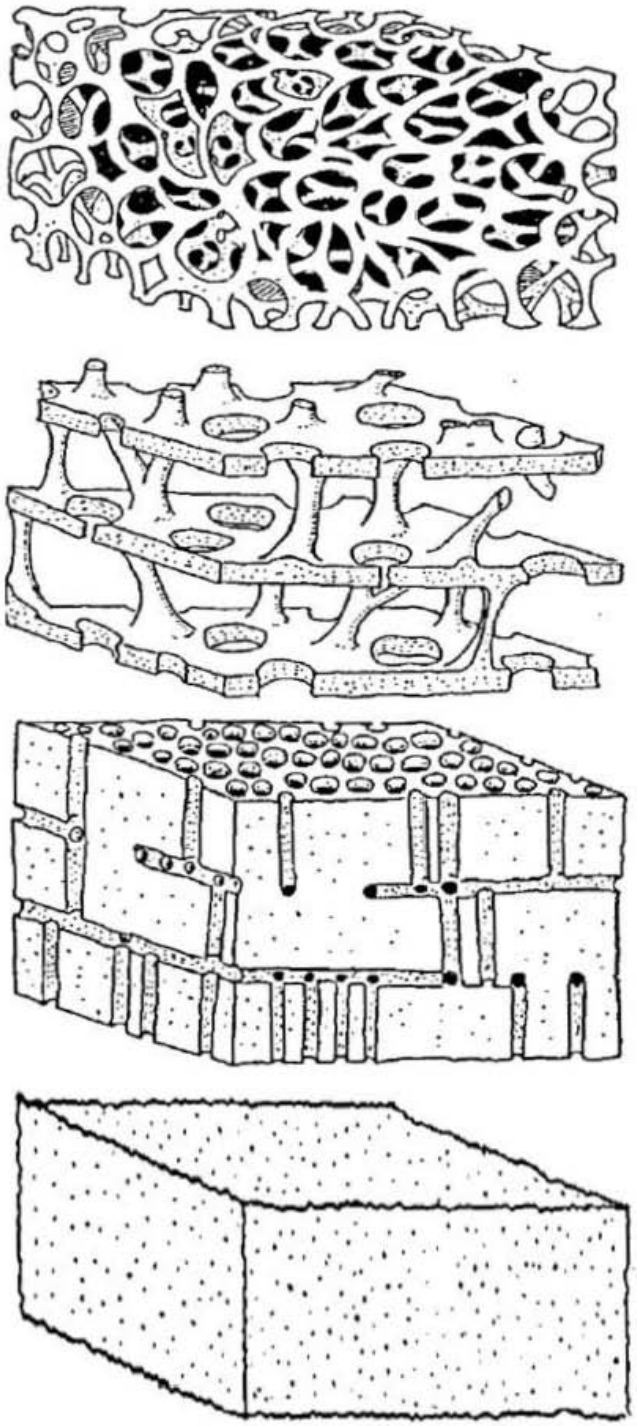


Figura 70. Esquemas de varias microestructuras que se encuentran en pruebas e imágenes microscópicas en las espinas de los erizos de mar (Echinodermata). El principio de organización se explica de arriba hacia abajo: en laberinto, microperforado e imperforado (Smith, 1980; Carnevali, 1991).

4.4 – La Herramienta Digital Computacional en la Estructuración Material

El uso de la tecnología computacional para el diseño de objetos virtuales y reales, ha estado apoyado por un gran número de aplicaciones asociadas al Diseño Asistido por Computadora, también conocido por sus siglas en inglés como CAD. Desarrollado inicialmente como una herramienta digital de sustitución de dibujo a mano para dibujos técnicos o de ingeniería; los resultados CAD típicamente representan información simbólica, tales como composiciones de formas, dimensiones, y tolerancias basadas en convenciones de aplicaciones específicas. El CAD es altamente utilizado en varias disciplinas con un dominio sobre otras aplicaciones bastante alto, que incluyen a la industria automotriz, naval e industrias aeroespaciales, diseño industrial y arquitectónico, entre otras. Dentro del campo del diseño arquitectónico, el CAD es la mayor fuerza de investigación en la industria del software computacional, que involucra a la geometría, las gráficas computacionales y la geometría discreta diferencial. Por lo tanto, el CAD ocupa una gran variedad de aplicaciones para diseño desde el modelado (generación digital en un dominio digital), análisis (mapeo digital de los dominios físicos) hasta la fabricación digital (generación física de los dominios digitales).

El uso omnipresente y ubicuo de las herramientas de diseño asistido por computadora, la ingeniería y la fabricación (respectivamente conocido como CAD, CAE y CAM) en la arquitectura, con frecuencia sirve en la práctica actual del diseño como tecnología de facilitación de producción, por la cual se materializan las predicciones contemporáneas de geometrías complejas y llamadas, de forma libre. En este sentido, este tipo de entornos asistidos por computadora pueden ser considerados como la extensión automática de una descripción referida a una forma tradicional, que forman los procesos que precedieron a la invención del CAD (Oxman, 2007; Oxman, 2008).

4.4.1 – De la Ayuda al Motor

Las extensiones digitales de los procesos de diseño convencionales han ganado legitimación y valor como ayuda a apoyar a un diseño mediático. Ayudando a la reformulación o readaptación de una forma, dichas herramientas están a la altura de la incorporación de las propiedades del material y las limitaciones en cuanto a su comportamiento particular, así como los procesos de fabricación y construcción inherentes a ellos. De hecho, al estimar la capacidad que tienen las herramientas computacionales para apoyar los procesos de generación de forma, fundamentados por distintos motivos, el diseño asistido por computadora aparece en su conjunto para facilitar un estado sin cambio, dicho de otra manera “más de lo mismo”. Como resultado, el mundo del diseño arquitectónico, ahora se encuentra tan saturado con expresiones meramente formales y excesivas, que siguen estando dominadas por decisiones que comienzan a partir de la forma sobre el material, con las consecuencias posteriores en el entorno construido; parece, sin embargo, que existen otras alternativas más interesantes.

En el desarrollo de procesos de diseño inspirados en la naturaleza, en esta investigación se busca emplear un proceso computacional alternativo de apoyo a la generación formal basada en la interacción entre el material, el medio ambiente y la programación gráfica para diseño. Esto implica un cambio de procesos asistidos computacionalmente de carácter generativo y de desempeño y simulación natural. En otras palabras, mi objetivo es pasar del diseño asistido por computadora a uno basado en el diseño de cálculo computacional – biológico. Esta transformación permitirá potencialmente al diseñador realizar los procesos de generación formal al abordar consideraciones materiales y ambientales. Este “nuevo

medio" de rendimiento y de generación basado en los materiales permitirá, también, producir formas de carácter "material – ecológico", es decir, un nuevo vocabulario de diseño, de otro modo no entendible.

4.4.2 – Computación Natural: Diseño Computacional inspirado en la Naturaleza

La Computación Natural[51] es un término introducido por de Castro para abarcar tres enfoques en la relación entre la naturaleza, la computación y el diseño (De Castro, 2006). En estos enfoques, se utilizan modelos de procesos naturales como fuente de inspiración y de desarrollo de nuevas herramientas, técnicas y tecnologías para resolver problemas complejos en diferentes ámbitos de la ingeniería y la biología. Estos tres enfoques pueden ser clasificados de acuerdo con el tipo de problemas que se tratan de resolver, entre ellos: la investigación, la simulación, el procesamiento y representación digital – física.

4.4.3 – Búsqueda y Optimización

En este enfoque, los algoritmos son desarrollados a partir del entendimiento de generación formal y función mecánica, además de aspectos biológicos, y con el objetivo de poder ofrecer soluciones a problemas complejos que no pueden ser resueltos de otro modo, como la programación lineal y dinámica; la idea central aquí es diseñar modelos teóricos que pueden ser implementados a partir de la programación gráfica computacional con un enfoque de resolución de problemas de diseño en lugar de una "forma" teórica. Ya sea a través de la inspiración biológica o biológicamente generada, las técnicas de computación se componen de modelos altamente abstractos, a veces denominados como metáforas (Paton, 1992), y están diseñados para imitar las características y mecanismos particulares de la biología. Algunos ejemplos son las redes neuronales artificiales y algoritmos evolutivos, los cuales son sistemas de procesamiento de información para la búsqueda y la optimización inspirados en el sistema nervioso central humano y la biología evolutiva respectivamente, con un especial énfasis en la resolución de problemas. Un trabajo innovador en el área de las redes neuronales fue iniciado por McCulloch y Pitts en un artículo titulado *"A Logical Calculus of the Ideas Immanent in Nervous Activity*" (McCulloch y Pitts, 1943), que presentó el primer modelo matemático del comportamiento de una neurona y que contribuyó al campo de investigación de las redes neuronales artificiales, utilizadas en la actualidad como parte de los algoritmos de programación para el control en extremidades humanas artificiales (prótesis), (Kohonen, 1998; Fausett, 1994; Haykin, 1994; Bishop, 1995). La Computación Evolutiva fue pionera en la década de los años de 1960 con los trabajos de investigación de I. Rechenberg, L. Fogel, A. Owens y J. H. Holland; brindando un auge importante en el campo de la Computación Evolutiva. La mayoría de los algoritmos evolutivos tienen su origen en la teoría neo – darwinista de la evolución, que propone que una población de individuos capaces de reproducirse y estar sujetos a variaciones genéticas seguido por el resultado de la selección natural en las nuevas poblaciones de individuos cada vez más adaptados a su entorno. Los algoritmos evolutivos (AE, por sus siglas en inglés), la inteligencia colectiva y los sistemas artificiales inmunológicos también se incluyen dentro de esta categoría (De Castro, 2006).

51 Existen muchas investigaciones anteriores a la del Diseño Computacional basado en la naturaleza - material, las cuales fueron llevadas a cabo por varias disciplinas, tales como la ciencia de materiales, la ingeniería y la computación en el contexto del diseño y de los procesos del diseño. Los procesos computacionales han sido identificados a través de la integración de criterios de desempeño mecánico natural con procesos de generación formal (Oxman, 2005).

4.4.4 – Simulación y Emulación

Este enfoque incluye los procesos de diseño sintéticos destinados a producir patrones, formas, comportamientos y organismos artificiales que se parecen a la vida natural tal cual la conocemos. Con sus resultados y procesos de generación formal, son utilizados para imitar diversos fenómenos naturales, que tienen el potencial de aumentar nuestra comprensión de la naturaleza a través de modelos computacionales. De Castro especifica dos sub-enfoques principales para la simulación y la emulación de la naturaleza a través de las herramientas de programación computacionales que incluyen la aplicación de técnicas de "vida" artificial o mediante el uso de herramientas para el estudio de la geometría fractal tan presente en la naturaleza. Los sistemas de "vida" artificial se han desarrollado para estudiar los retrasos respecto al tráfico de los automóviles (Resnick, 1993), el comportamiento de los sistemas biológicos sintéticos (Ray, 1994) y la simulación de comportamientos colectivos (Reynolds, 1993), entre otros. Su principal preocupación se orienta hacia el estudio, investigación y aplicación de modelos de sistemas de "vida", así como a la construcción de "vida" artificial a partir de elementos o partes "sin vida".

4.4.5 – Computación con Materiales Naturales

Este enfoque se relaciona con la utilización de materiales naturales para llevar a cabo un cálculo y simulación computacional. Además de que pertenece a todos los desarrollos de hardware inspirados en la naturaleza, y por lo tanto, contribuye a una verdadera fuente de investigación de paradigmas informáticos que pueden ser de sustitución o de complemento de los programas computacionales. La computación con materiales naturales está orientada hacia nuevos métodos de cálculo basados en las propiedades físicas, químicas y mecánicas de los materiales. Estos métodos resultan en una computación no estándar, al superar algunas de las limitaciones de software y computadoras secuenciales comunes. Dado que los dispositivos bio – electrónicos pueden alcanzar dimensiones que son cien veces más pequeños que los dispositivos convencionales, las computadoras bio – electrónicas se pueden utilizar para aprovechar la rapidez de procesamiento, ejecución e interrupciones de ciclos activados por la luz (Forbes, 2004). Dado el supuesto de que cualquier operación matemática se puede dividir en bits, y cualquier función lógica se puede construir utilizando los comandos *AND* y un acceso *NOT*, cualquier entidad computable se puede resolver apropiadamente mediante estos comandos. Esta independencia de un sistema de representación específica hace posible el uso de nuevos conceptos para procesos computacionales basados en materiales naturales, tales como las reacciones químicas, las moléculas de ADN y los dispositivos mecánicos cuánticos (De Castro, 2006). Aunque todavía en su inicio, la bio – electrónica tiene el potencial de proporcionar importantes ventajas sobre los dispositivos tradicionales hechos con silicio. El diseño y la construcción en la bio – electrónica a través de la ingeniería genética o la síntesis química pueden permitir un mayor grado de control (Forbes, 2004). Gran parte del trabajo relacionado se está llevando a cabo desde hace diez años en el MIT por Tom Knight, del laboratorio de Inteligencia Artificial y Neil Gershenfeld del programa CBA[52]. He leído recientemente que Knight estaba trabajando en el montaje a escala molecular de circuitos integrados con el objetivo primario de programar el comportamiento de las bacterias de la misma manera que un científico o ingeniero de la computación puede programar a un microprocesador (Forbes, 2004).

52 Centro de Bits y Átomos (por sus siglas en inglés) de MIT (Massachusetts Institute of Technology).

4.4.6 – El Problema con la Computación Natural

Los procesos de cálculo naturales proporcionan al diseñador una nueva mirada del mundo natural, ya que le proporciona nuevas formas de explorar y entender los procesos que se encuentran en el núcleo del mundo primario de la naturaleza. El principal problema inherente a la computación natural, sin embargo, sigue siendo su utilización en el diseño a través de técnicas numéricas y algorítmicas, quedando exentas en varias ocasiones los experimentos formales, que están a menudo divorciadas de los problemas del mundo real. En este sentido, se puede encontrar a menudo que una red de venas de una hoja o las formas de las burbujas de jabón se traducen en materiales de construcción únicamente como formas geométricas, sin necesidad de entender la lógica estructural y el material inherente a su manifestación.

4.4.7 – El Problema de la *Pre* y *Post* Geometría

Al suponer una relación simbiótica geométrica, el diseño incorpora muchas cuestiones que son independientes de cualquier configuración formal específica. Estos problemas pueden definirse como el "espacio de parámetros" para una problemática específica dada. Estos "espacios" pueden ser considerados como *"pre – geométricos*" en los organismos vegetales y animales de la naturaleza; habiendo llegado a una configuración en particular, existen potencialmente diversas interpretaciones materiales alternativas de esas configuraciones particulares, que pueden considerarse como cuestiones *"post – geométricas"* (Oxman, 2007).

4.4.8 – El Problema de los Ciclos de Síntesis y Análisis

La Geometría Computacional o generada a través de programas de computadora habitualmente se ha utilizado como un medio para la descripción, representación y análisis final de la forma. En menor medida se ha utilizado como un medio instrumental para efectos de generación y producción en el diseño. Dada la importancia de este tipo de herramientas en la exploración de los cuerpos geométricos y de la forma, la limitación mantiene una división entre los modelos metodológicos de la descripción y los modelos de la generación o producción. La integración de herramientas y técnicas, que sean más analíticas que descriptivas, puede potencialmente proporcionar al usuario final (diseñador) la capacidad de explotar su trabajo a través de geometría computacional como un controlador para un proceso de diseño particular integrado y que posea una función que tome en consideración rendimientos relacionados, tales como la función y la adaptación al medio natural externo.

El objetivo principal de este trabajo, como requisito previo fundamental para un posible cambio de paradigma en el diseño computacional generativo, es promover una metodología diferente y novedosa que apoye la integración perfecta de la geometría y el rendimiento. La representación con multi – objetivos se basa en las entidades geométricas (o descripción de la forma) para promover una especulación sobre el comportamiento estructural y ambiental de un modelo que respalda a un proceso y objeto de diseño que toma como aspecto generativo a la naturaleza y como medio de producción a la programación computacional gráfica.

4.4.9 – El problema de la Optimización

Respecto a los procesos digitales siguiendo constantes físicas, se han llevado a cabo muchas investigaciones en relación a la optimización en el diseño, enlazado con la evaluación del rendimiento en un diseño ya existente, sea este digital o real. Investigaciones anteriores tienen por objeto la asignación de inteligencia artificial para unidades discretas locales, tal que cada sub – dominio tiene la capacidad de regularse por sí mismo de acuerdo a uno o varios criterios de comportamiento antes programados (Hanna y Mahdavi, 2004). Bajo las distintas condiciones variables de cargas estructurales, cada elemento es definido por un algoritmo, que puede modificar su espesor y orientación en relación a las condiciones particulares de carga. En microescalas estructurales, Hanna percibe un sistema modular repetitivo, en el cual el tamaño se relaciona de manera directa con cada elemento del sub – dominio. Por cada elemento de geometría dado, estos cambios se producen como consecuencia y respuesta a las fuerzas ambientales externas. En combinación con los Algoritmos Genéticos (*GA's* por sus siglas en ingles) y Algoritmos de Aprendizaje Mecánicos, los elementos modifican sus atributos geométricos y topológicos. La anterior investigación aún carece de la incorporación de las propiedades de los materiales físicos como posibles variables que afectan a la distribución de materia en la geometría para resolver y dar respuesta a las fuerzas locales externas (Hanna y Mahdavi, 2004).

Una investigación anterior del Departamento de Ingeniería Mecánica del MIT, combina un acercamiento práctico de los GA's con una función objetiva que consiste en la deformación de la geometría a través de la salida de energía para eliminar material en zonas que no son necesarias (Shimada y Gossard, 1998). Esta línea de trabajo se superpone con la optimización de la forma dónde la geometría es optimizada según las cargas óptimas locales y de todo el sistema. También existen paquetes de "regulación" y flujos de trabajo comerciales como el "phoenix" y el "insight" que pueden ajustar y modificar el código de los algoritmos en relación a elementos finitos de simulación[53] para llevar a cabo la optimización del resultado o consecuencia de la forma.

Esta investigación elimina las "Jerarquías de procedimientos", las cuales existen potencialmente entre la "pre" y "post" operaciones geométricas en el diseño (por ejemplo, primero generación de la forma y después evaluación de rendimiento) y ofrece frecuentemente una nueva metodología para la incorporación del "Rendimiento material directa y explícitamente dentro de la representación geométrica". Un trabajo de investigación anterior y en relación a esta línea ha sido llevado a cabo a través de modelos que promueven la exploración de diseño promoviendo diferentes formas de representación geométricas que puede ser enlazadas para soportar y descubrir nuevos procesos de diseño (Killian, 2004).

53 Los métodos de elementos finitos están incluidos dentro de un gran rango de métodos de análisis estructural (métodos analíticos, resistencia de materiales, métodos de elasticidad, entre otros), que son utilizados en análisis de rendimiento. El Método de Elementos Finitos (FEM, por sus siglas en ingles), es un método de análisis principalmente empleado dentro de las disciplinas de la ingeniería que permiten cálculos estructurales de unos elementos sólidos o fluidos dados. El objeto en general de estos sólidos puede en ocasiones ser muy difícil de calcular, sin embargo, la discretización de tales formas, reduce de manera significativa la complejidad física y matemáticas de los análisis. El método fue desarrollado en la década de los años de 1940 por Alexander Hrennikoff y Richard Courant. Originado a partir de la necesidad de resolver grandes análisis complejos de elasticidad y problemas de comportamiento estructural, el FEM ofrecía un nuevo acercamiento que promovía un acercamiento de mallas geométricas en un dominio continuo (definido como el cálculo matemático de un objeto a analizar en relación a su ambiente) dentro de una lista discreta y selecta de sub – dominios. En el campo de la mecánica estructural, los FEM son frecuentemente la base de los principios de trabajos energéticos virtuales o el principio del potencial mínimo de energía que brinda un análisis general, intuitivo y basado en la física a los ingenieros estructurales.

El objetivo principal de este trabajo, como requisito previo fundamental para un posible cambio de paradigma en el diseño paramétrico, es promover una nueva metodología que apoye a la integración perfecta de la generación geométrica a partir del rendimiento y simulación algorítmica computacional. La representación de multi – objetivos, dónde las entidades geométricas (o descripción de formas) promueven especulaciones sobre el comportamiento estructural y ambiental de un modelo que respalda un proceso de diseño que es generado y emulado a partir del comportamiento de la naturaleza. Con el objetivo de lograr procesos variables como nuevas formas requeridas de representación.

4.4.10 – Escalares, Vectores y Tensores: Nuevas formas de representación

Varias cualidades matemáticas están representadas claramente utilizando herramientas visuales y formulaciones conceptuales en lugar de utilizar simplemente números y funciones (figura 71). Los aprendices visuales frecuentemente prefieren los métodos formativos eludiendo a las cualidades en lugar de las cantidades. El concepto de “tensores” es uno de ellos, y está dado por su considerable relevancia para la comprensión de la computación basada en materiales.

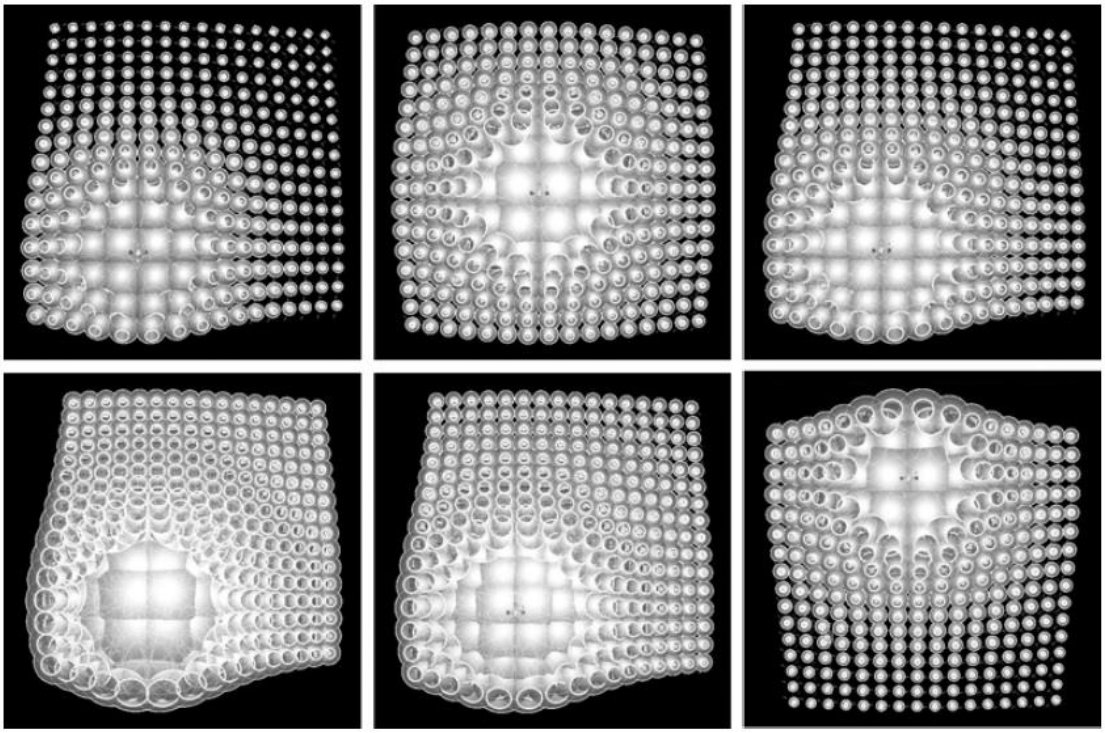


Figura 71. Ejercicio de diseño paramétrico que explora la relación de un simple componente y su ambiente “incorporado”. En este caso, un punto relacionado define la conformación entera de todos los demás componentes definidos como esferas de tamaño y escala parcial. Las exploraciones en el diseño paramétrico como esta, están generalmente conformadas y definidas por constantes matemáticas y geométricas. Sin embargo, es claro que es estos casos, las rutinas de edición computacional define la forma consecuente en lugar de generar alguna arquitectura “pre – concebida”. Imagen programada y concebida mediante el software GenerativeComponents, año 2014.

Las entidades expresadas numéricamente, como los precios y las fuerzas se pueden caracterizar por objetos matemáticos que incluyen cantidades o direcciones. Tales objetos se denominan escalares y vectoriales respectivamente.

Una cantidad escalar puede ser representada por un número único (por ejemplo, la temperatura, peso, volumen, tiempo, entre otras). Una cantidad vectorial representa un conjunto de tres cantidades escalares o números, denominados de manera común como un vector. En combinación, comparten la descripción de una o varias cantidades que contiene descripciones de magnitudes y direcciones determinadas. Los ejemplos incluyen fuerzas, velocidad, aceleración, gradientes de escalares, entre otros. Todas estas entidades están descritas a través de algún número y alguna dirección que indica su camino en el espacio cartesiano. En cada uno de estos casos, se deben utilizar tres números para describir de manera completa la cantidad en mano. Una coordenada específica es seleccionada para describir un vector particular (figura 74), así que, si alguien describe una fuerza dada "F" en una coordenada rectilínea, deberá ser requerida para especificar los componentes de tal fuerza (Fx, Fy, Fz) para cada una de las tres coordenadas mutuas perpendiculares (Danielson y Noor, 1997).

Los tensores son entidades geométricas ocupando los dominios de las matemáticas y la física, tal que brindan extensiones conceptuales y técnicas de valores escalares, vectoriales y matriciales. Las múltiples propiedades físicas pueden ser expresadas como correspondencias entre dos o más conjuntos de vectores. La rigidez, por instancia, toma un vector como entrada y produce otro vector como salida, con tal de que una expresión entre la entrada y la salida sea generada (figura 72). Lo anterior es debido a las expresiones de relaciones entre los vectores, por lo que los tensores son independientes por sí mismos de una elección o modificación particular en un sistema coordinado. Por lo tanto, el tensor es una cantidad que obedece a reglas de transformación en cuanto a tensores. Lo anterior se conoce como naturaleza de coordinación independiente, ya que los tensores hacen que cada uno de ellos adquiera la forma de una ley de transformación "co – variante", lo que significa que una relación entre matrices se calcula en un solo sistema de coordenadas de otras matrices, calculadas en otro sistema de coordenadas. Cada matriz se presenta como una matriz multidireccional de valores numéricos. Las fuerzas de deformación y elásticas son ejemplos de tales tensores (Danielson y Noor, 1997).

Cada una de las tres caras mutuamente perpendiculares del cubo (figura 73) representa tres elementos de tensión, que colectivamente constituyen al tensor. Los tensores tienen órdenes o grados, que son asignados, que corresponden a una dimensionalidad de las matrices que representan a cada uno de ellos. Por extensión lógica, un escalar podría considerarse como un tensor del orden cero (su magnitud es su único componente, por lo que puede representarse como una matriz dimensional cero), y un vector de un solo orden (el vector está representado en coordenadas como una sola matriz unidimensional de los componentes). Según esta lógica, una matriz de 3 X 3 es un tensor de segundo orden (siendo representando en una matriz de dos dimensiones).

Los tensores son objetos y conceptos matemáticos muy útiles en los dominios de la ingeniería y la física. La industria de imágenes médicas, por ejemplo, se ha desarrollado rápidamente durante la última década con el creciente interés de la difusión por tensores de imagen de resonancia magnética (MRI – DT, por sus siglas en inglés). En la difusión de imágenes tensor, por ejemplo, las exploraciones del cerebro humano se producen mediante la expresión de la permeabilidad diferencial de los órganos en el agua, en diferentes posiciones y direcciones utilizando cantidades de tensor. Por lo tanto, proporcionan un gran marco de referencia para modelar la variabilidad anatómica del cerebro.

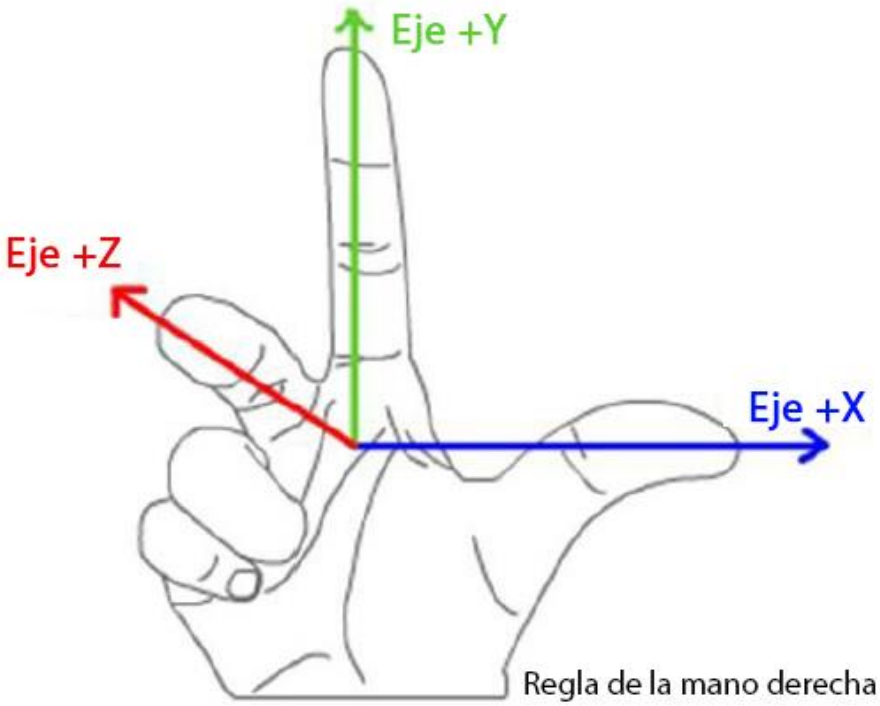


Figura 72. La regla de la mano derecha en matemáticas y en física es una expresión a través de la mano del ser humano para la expresión de convenciones de notación para vectores en 3 dimensiones.

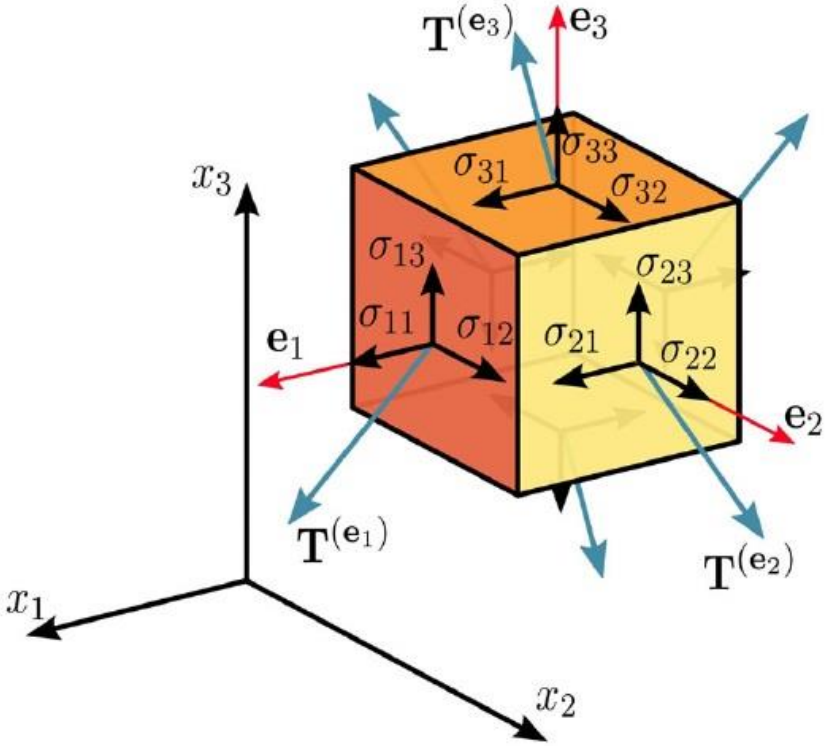


Figura 73. A diferencia de los vectores, los tensores se relacionan con más de tres dimensiones como es el caso de la expresión de tensiones o esfuerzos. El tensor puede tener un número arbitrario de los índices relativos a varias características físicas, así como a limitaciones. El tensor de tensiones se ilustra en la imagen de arriba, el cual es un tensor de segundo orden. Dado que estamos operando generalmente en tres dimensiones el tensor de tensiones tiene 9 elementos.

También son utilizados ampliamente en la mecánica, por ejemplo, con las deformaciones o tensión en los tensores, y se están convirtiendo en una herramienta común en el análisis numérico para generar mallas adaptadas para reducir el costo computacional de la resolución de ecuaciones diferenciales parciales en 3D (PDE, por sus siglas en inglés) (figura 74).

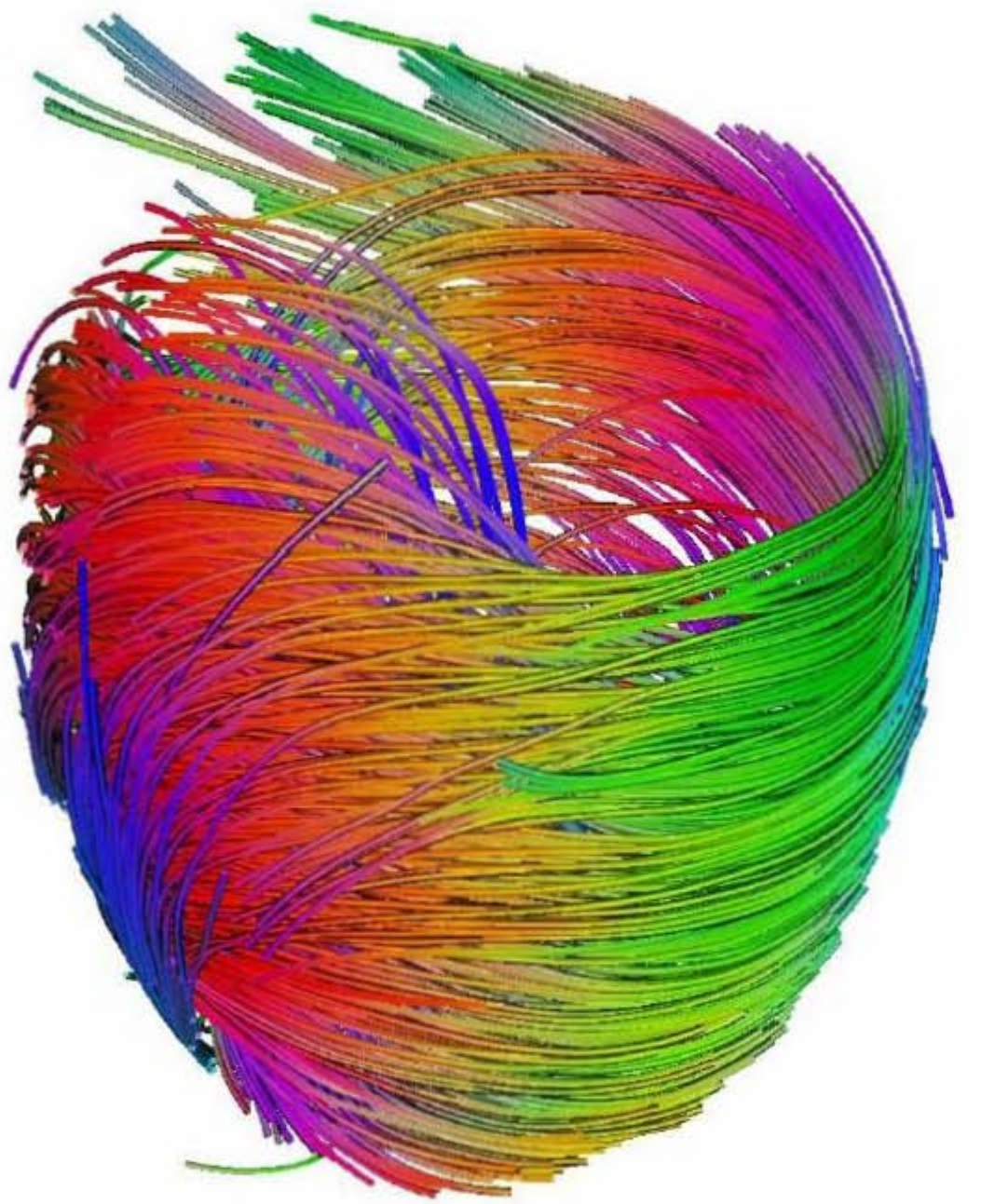


Figura 74. La imagen fue generada a partir de un escaneo MRI – DT de un corazón humano, utilizando el método de Imagen de Difusión de Tensor. El escaneo sigue el movimiento de las moléculas de agua a través de los músculos del corazón y revela como las células de este músculo son alineadas. Las líneas representan la orientación de las fibras del músculo en las áreas más grandes, es decir, el ventrículo izquierdo. Las técnicas de representación de tensores orientados permiten a los profesionales, en esta ocasión, de la medicina, poder observar el comportamiento físico de datos en conjunto con formalismos geométricos en tiempo real. Fuente: *http: // www.ox.ac.uk / media / sciencelog / 100209.html*

La técnica de imagen a través de Tensores de Difusión en Resonancias Magnéticas puede utilizarse para reconstruir la geometría de algunos corazones humanos, así como su estructura de fibras laminares en una alta resolución. En esta técnica, un tensor que representa la difusión 3D del agua en cada vóxel[54] de la imagen se calcula de tal manera que está alineado con la estructura de la fibra cardiaca del corazón. De esta manera, es relativamente fácil determinar una enfermedad repentina del músculo cardíaco (Helm, 2005). Varios marcos de cálculos existen para los métodos de tensores, que pueden convertirse en formas euclidianas como tensores que se transforman en logaritmos de matriz propios, lo que hace del procesamiento clásico euclidiano particularmente fácil de llevar a cabo.

4.4.11 – Puntos, Líneas, Superficies, Sólidos y Mallas: El problema de las jerarquías

El aprendizaje de CAD no es diferente a aprender un nuevo lenguaje de programación: “usted puede conocer a alguien, que lo conozca”. Ocupando un lienzo digital, el cerebro del arquitecto o diseñador es el mundo de la geometría, que se rige por un sistema de coordenadas universalmente eminente (CS, por sus siglas en inglés), para poner en relación un punto (0, 0, 0) por cada nueva iteración o producción. Los menús desplegables que contienen funciones, métodos y herramientas de generación de formas diversas, acompañan a la ventana principal en la que se dibujan nuevas formas como puntos que se combinan para dibujar líneas y curvas que son levantadas para crear sólidos, superficies o mallas tridimensionales.

En este universo euclidiano, cada punto que compone un diseño particular incluye los ejes cartesianos X - Y – Z, los valores correspondientes con su ubicación universal, en relación con el CS. Las cosas se vuelven sencillas una vez que uno ha llegado a un entendimiento de las jerarquías de diseño geométricamente generado. Los puntos hacen líneas o curvas, las líneas hacen superficies, y las superficies hacen sólidos o mallas. Si bien este tipo de representaciones del espacio son suficientes para llevar a cabo formas euclidianas primitivas y básicas, la representación se convierte en un gran reto para elementos espaciales más complicados.

En el mundo de la naturaleza, la generación de la forma parece saltar normalmente los pasos requeridos por el diseñador: líneas y superficies de espesor cero son elementos demasiado complicados de rastrear o, mejor dicho, estas entidades apenas pueden ser percibidas por el ojo humano en la naturaleza. El mundo físico puede ser descrito y construido por el decreto y dictado de la geometría, pero hay mucho más para dar forma a la forma por sí misma. Consideremos, por ejemplo, la producción de tejido óseo por medio de la geometría. Las líneas o curvas no serán suficientes para la descripción o representación del tejido como una función de los cambios que responden y son consecuencia de las cargas que se le aplican, además de los diferentes movimientos. Lo mismo aplica para el crecimiento de los árboles, corales, plantas, tejidos cutáneos de animales e insectos, formación de tejido muscular y así sucesivamente.

54 El vóxel (del inglés volumetric pixel) es la unidad cúbica que compone un objeto tridimensional. Constituye la unidad mínima procesable de una matriz tridimensional y es, por tanto, el equivalente del píxel en un objeto 2D.

4.5 – Estrategias de la naturaleza computarizadas

4.5.1 – Materia sobre forma

Objetivo: Desarrollar un proceso de diseño sostenible que prioriza el material sobre la forma (en contradicción con el enfoque de diseño tradicional, dónde la materia es un aspecto secundario a la forma).

Los procesos convencionales de diseño se basan en la generación de una forma geométrica 3D típicamente utilizando como herramienta digital software CAD. La asignación de un material de diseño inicial, como punto de partida, y para poder fabricar el producto (o la construcción de la parte de un edificio), se deriva en un compromiso para el diseñador meramente formal. Al emplear este medio de trabajo, el entorno de modelado computacional permite al diseñador generar la forma mediante una secuencia de comandos textuales o gráficos, que por lo general no guardan relación con su manifestación física y de materia, o mediante la subdivisión de la geometría en componentes unitarios o individuales. En ambos casos, los procesos de generación y modificación de objetos o teselaciones (subdivisiones en la geometría 2D y 3D) se conciben como un medio de simplificar un problema de diseño particular, siempre teniendo en cuenta las estrategias y medios de fabricación y montaje.

La propuesta que hago es conseguir productos de diseño más sostenibles mediante procesos de generación de forma, que deben de incluir herramientas de apoyo a la integración del software para el análisis permitiendo la iteración en paralelo entre la forma digital y la evaluación física de su rendimiento y comportamiento correspondiente. En última instancia, el objetivo del diseñador es servirse de los entornos de computación que puedan apoyar potencialmente los atributos físicos relevantes para el arquitecto, como la rigidez y la traslucidez que pueden contribuir a la generación de la forma en sí. Varias estrategias para lograr este objetivo se desarrollarán y presentarán en este capítulo.

4.5.2.1 – Rígido y transparente – Suave y opaco

Una función fundamental (o la función objetivo) está generalmente asociada con la optimización y resolución de un problema que determina la buena solución que es (Atallah, 1999). La optimización ambiental y estructural, son ejecutadas a través del Método de Elementos Finitos (FEM) y la Dinámica Computacional de Fluidos (CFD) respectivamente, lo que permite la evaluación y modulación de elementos locales en un volumen y características de superficie en un objeto bajo un conjunto de constantes asignadas (ejemplo, cargas puntuales y distribuidas en una columna).

4.5.2.2 – El entorno de fabricación - generación

Los procesos integrados de diseño ahorran tiempo y cálculos computacionales, y permiten trabajar en un ambiente de trabajo digital considerando espacios para tales procesos, mediante las interfaces, gráficas o textuales, como ambientes unificados, dónde el modelado, el análisis y la fabricación están interrelacionados. Actualmente, sin embargo, tales plataformas de diseño se encuentran discretas y separadas por métodos de trabajo y herramientas: los diseñadores típicamente modelan una forma que ya está predeterminada en el programa, la evalúan y en base a este resultado, la modifican con el objetivo de acercarse a una simulación de comportamiento o simulación espacial determinada. Tales

protocolos determinados deben ser modificados o replanteados, si se tiene en cuenta la interrelación potencial entre la geometría, el material y la estructura como partes integrales de todo proceso de diseño, mediante el modelado, el análisis y la respectiva rutina de fabricación. En el marco de este trabajo de investigación trataré de proponer una nueva forma de trabajo que sustenten las anteriores integraciones y que promuevan un método más sostenible de hacer diseño.

4.5.2.3 – Los tejidos – teselaciones continúas

Esta pregunta desencadena una serie de dudas e investigaciones sobre la definición de "material" en el diseño. Más allá de su definición clásica como una sustancia particular, se puede clasificar como un conjunto de materia física bajo los aspectos de modificación físicos estructurales. La propuesta busca controlar y modificar las propiedades de la forma paramétrica como la elasticidad, la rigidez, la transparencia, la conductividad térmica, el crecimiento, entre otros.

El motivo de responder a esta cuestión radica en la clasificación sistemática de las propiedades de los materiales y cómo se pueden relacionar de manera directa con la manipulación geométrica paramétrica computacional. Aquí comenzamos a trazar propiedades macro y micro – estructurales junto con las descripciones rigurosas de los atributos formales que pueden ser producidas potencialmente mediante las herramientas paramétricas.

La propuesta también pretende demostrar la diferencia entre material, que se define aquí como la capacidad de variación de las propiedades de los materiales correspondientes a un conjunto determinado de condiciones y se rige por una función objetivo en particular, a través de la selección de materiales y a partir de la asignación de constantes ambientales y estructurales a cada uno de ellos. El principal objetivo es llegar y proponer un algoritmo que pueda cambiar y simular las propiedades físicas y geométricas de un objeto de la naturaleza de acuerdo al entorno de datos paramétricos que serán definidos por el usuario.

4.5.2.4 – Geometría física

La representación digital de un comportamiento físico ha estado relacionada con los campos de la ciencia computacional, ingeniería de materiales y las matemáticas. Sin embargo, esto no se ha definido y utilizado en el diseño como factor determinante inicial y como generador de una forma considerada. La investigación explicará la validez y utilidad de un proceso computacional interrelacionado que integre el comportamiento físico (rigidez, tensión, energía, entre otros), así como la alta integración de datos dimensionales en la representación geométrica digital. Los mapas escalares, así como los vectores y el alto rango de los tensores son campos que pueden influir potencialmente durante el proceso de diseño en el entorno computacional de programación gráfica a través de la variación de sus propiedades. Cada vóxel es un par de definiciones de datos no idénticas a otras y transporta información potencial que influirá en la forma, como un conjunto de direcciones.

4.6 – Problemas de investigación: el diseño como segunda naturaleza

4.6.1 – Diseño sin representación

Problema 1: ¿Puede la generación de la forma, la evaluación y la representación ser computarizadas en un ambiente integrado de programación mediante algoritmos gráficos?

En la naturaleza, la forma es el resultado de las propiedades compuestas de la sustancia con varios tipos de interacción con el ambiente externo, que es generado por medio de los diferentes ecosistemas en el planeta tierra. Colectivamente, el "material" y el "ambiente" están compuestos y relacionados con el objetivo de generar y optimizar las sustancias en formas orgánicas.

Así, como fue expuesto a lo largo de este capítulo de esta tesis, la capacidad de la naturaleza para integrar los procesos de la generación formales, la simulación y el análisis garantiza que la forma está conscientemente optimizada para contrarrestar los requerimientos ambientales y las cargas a las que es expuesta. Los precedentes ilustran que los mecanismos generadores que gobiernan la forma de los árboles y los huesos han sido revisados y es un hecho que cuando los procesos de modelado, simulación, análisis y representación están consolidados y unidos, dentro de una coherente definición de programación gráfica o textual; la consecuencia de forma aplicada a un problema de diseño, puede permitir a los arquitectos y diseñadores llegar a un resultado más eficiente, efectivo y sostenible durante un proceso de diseño "natural".

Con las consideraciones teóricas anteriores, estamos de cara de resolver las preguntas relacionadas con el proceso "computacional - natural" aplicado al diseño. He mostrado anteriormente que cuando un proceso está llevado a cabo de manera sostenible, la simulación, el material y la consecuencia formal están siempre relacionados, los productos que tal proceso produce son por sí mismo sostenibles, de modo tal que pueden ser utilizados para estructurar un objeto a partir de sus propiedades materiales en correspondencia con la negociación de multi – rendimiento.

¿Cómo, entonces, podemos lograr la consolidación de parámetros algorítmicos computacionales en la generación de la forma?, en otras palabras:

¿Cómo se puede lograr la calibración computacional de la materia y la energía, ligadas a la creación de la forma, sin crear primero la forma en sí?

4.6.2 – Enfoque de propiedades graduales

Problema 2: ¿Cuál es el papel de la variación y heterogeneidad en la naturaleza y, ¿cómo puede promover un enfoque en el diseño en un entorno artificial?

Buscamos definir qué funciones pueden cumplir la variación y la heterogeneidad en cada una de las disciplinas examinadas a lo largo de los capítulos de esta tesis (computación, programación gráfica y textual, materia, materiales y biología). El objetivo es centrarse en los mecanismos y métodos utilizados para ayudar y lograr la producción de composiciones geométricas computacionales gráficas heterogéneas que correspondan a los factores y limitantes ambientales.

4.6.3 – La forma vía comportamiento material

Problema 3: ¿Cómo priorizar el proceso de material sobre materia en la generación y representación de diseño?

La imagen del arquitecto como aquel profesional que “daba” la forma ha dominado por siglos nuestra profesión. En la mayoría de los casos, las estrategias estructurales estaban enlazadas a manera de pos-racionalización en apoyo a la utilidad del edificio capturado por sus propiedades espaciales. En este caso, la aplicación y selección de materiales eran dependientes de las soluciones estructurales proporcionadas por el ingeniero. Tales puntos de vista enfatizan la naturaleza jerárquica de los procesos de diseño con la forma que es el primer artículo de producción por encima de las propiedades estructurales y de comportamiento básico de los materiales. La arquitectura monumental de Frank O. Gehry, provee muchos ejemplos de lo anterior (Oxman, 2007; Oxman, 2008). Paralelo al enfoque de la “forma primero”, e influenciado por el trabajo de ingenieros muy reconocidos como el grupo ARUP y la oficina Happold, un esquema alternativo prioriza la función de la estructura como el principal controlador de la expresión formal. El acercamiento de la “Estructura primero” es manifestado particularmente en grandes proyectos complejos de ingeniería, tales como puentes y edificios de gran altura. Por el contrario, el aspecto material ha estado tradicionalmente relacionado como una característica extra de la forma, pero no como originador. En la naturaleza, la secuencia jerárquica *forma – estructura – material* es inversa, de tal forma que el material genera la estructura, la cual, a su vez, produce la forma de los especímenes naturales. Parece que entre las principales aportaciones de la naturaleza al ser humano como modelo para diseño, es el enfoque de lo que favorece a la materia (materiales) como factor central y significativo asociado con la generación de la forma en el mundo artificial humano. Tal es el caso, por ejemplo, de los huesos y otras estructuras celulares que producen formas, que son generados directamente por las propiedades de los materiales que están hechos. En la naturaleza, en la mayoría de los casos, el material es lo primero, ¿cómo puede un enfoque de “materia primero” ser óptimo, simulado y producir un resultado formal a partir de la programación logarítmica gráfica?

Para poder vislumbrar un acercamiento en esta investigación de manera más explícita y gráfica, he elaborado un diagrama que relaciona las áreas de estudio involucradas en este documento, como un resumen de ideas y fundamentos al caso de estudio por desarrollar en este capítulo (figura 75).

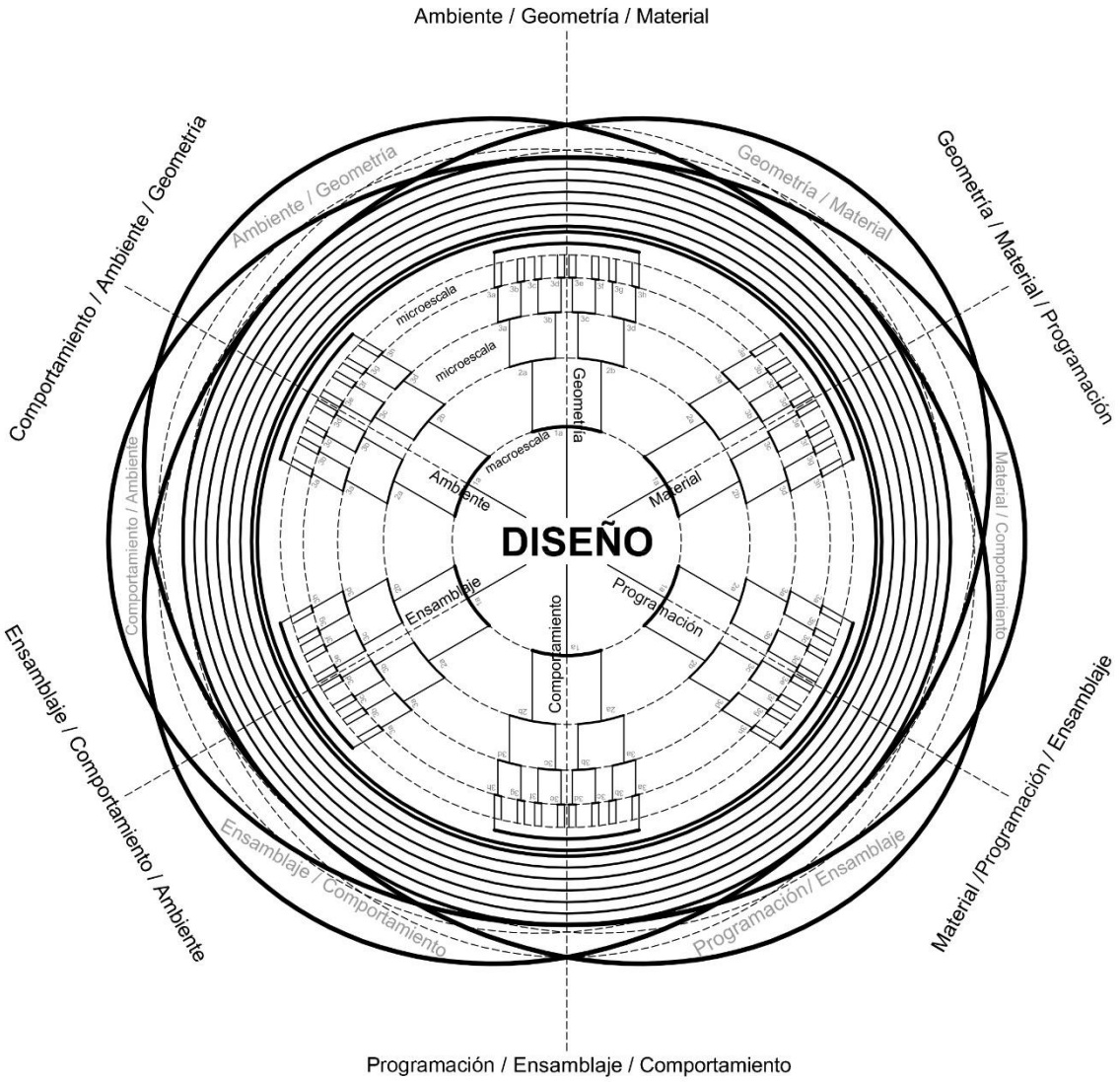


Figura 75. El diagrama ilustra cómo distintas disciplinas pueden ser integradas a un proceso de diseño con el objetivo de poder llevar a cabo el diseño basado el rendimiento de una estructura presente en la naturaleza. Se han tomado en cuenta consideraciones a partir de la geometría, el material, la programación, el ensamble lógico, el comportamiento y el ambiente en relaciones que son yuxtapuestas para dirigir el proceso de generación y producción digital formal. La suposición base es que este proceso de diseño sea capaz de integrar las constantes de modelado digital algorítmico y gráfico, simulación y análisis durante y como consecuencia de proceso de generación; en lugar de concebir tales procesos como linealmente dependientes.

Como punto de inicio del caso de estudio en este capítulo, he desarrollado un esquema que resume las particularidades intrínsecas que serán aplicadas y servirán como conceptos iniciales al proceso de diseño, con enfoque en la programación gráfica computacional, además de tomar en consideración a todos aquellos conceptos expuestos y explicados a lo largo de este documento (figura 76).

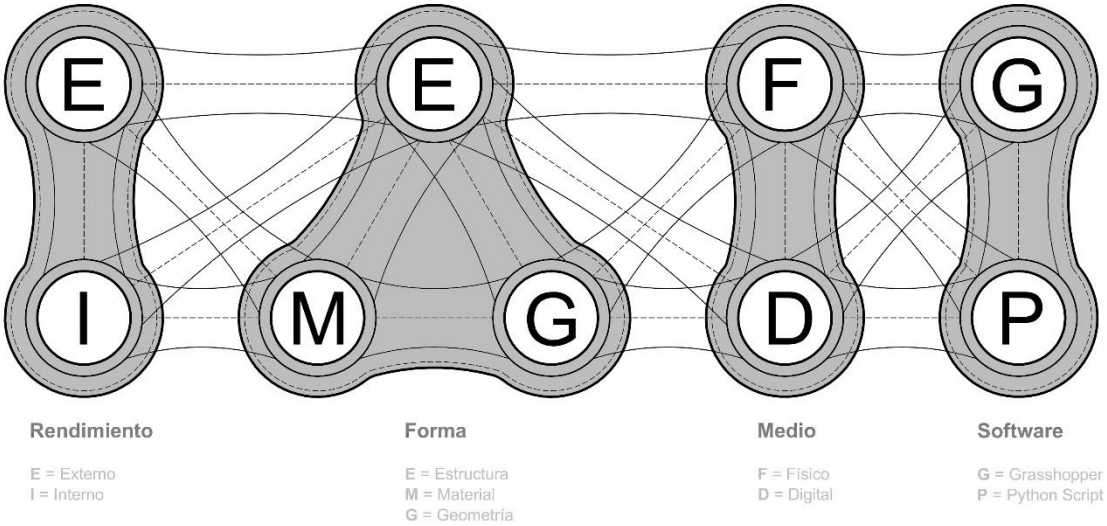


Figura 76. Este diagrama muestra las posibles relaciones que pueden ser llevadas a cabo con el fin de concebir un proceso de diseño a partir de la simulación y comportamiento de la naturaleza, para una búsqueda del modelo de relación más apropiado para desarrollar el primer caso de estudio. El primer grupo tiene que ver con el rendimiento propio de la estructura orgánica, sea este interno a partir de las células, o externo, por medio de la variación de los tejidos en repuesta al medio ambiente natural externo. Por otra parte, el segundo grupo está relacionado con todos los aspectos que pueden dar origen a un objeto en el aspecto formal, tal como el material, la estructura y la geometría, analizados y revisados a través de este capítulo. El tercer grupo se compone del medio de representación, sea físico o digital. Por último, el cuarto grupo representa la herramienta computacional digital de trabajo para llevar a cabo dicho proceso de diseño, explicadas a detalle en los primeros capítulos de este documento.

A partir de una serie de combinaciones del esquema anterior y mediante la relación lineal de cada concepto que los grupos representan, puede ser llevado a cabo un gran número de combinaciones que permitan al arquitecto o diseñador acercarse y proponer diversos métodos de trabajo durante un proceso de diseño arquitectónico general o particular. Tales combinaciones pueden representar un punto de partida en cada proceso, además de una selección de conceptos a investigar antes de cada etapa de diseño y su posterior representación, sea física o digital (figura 77).

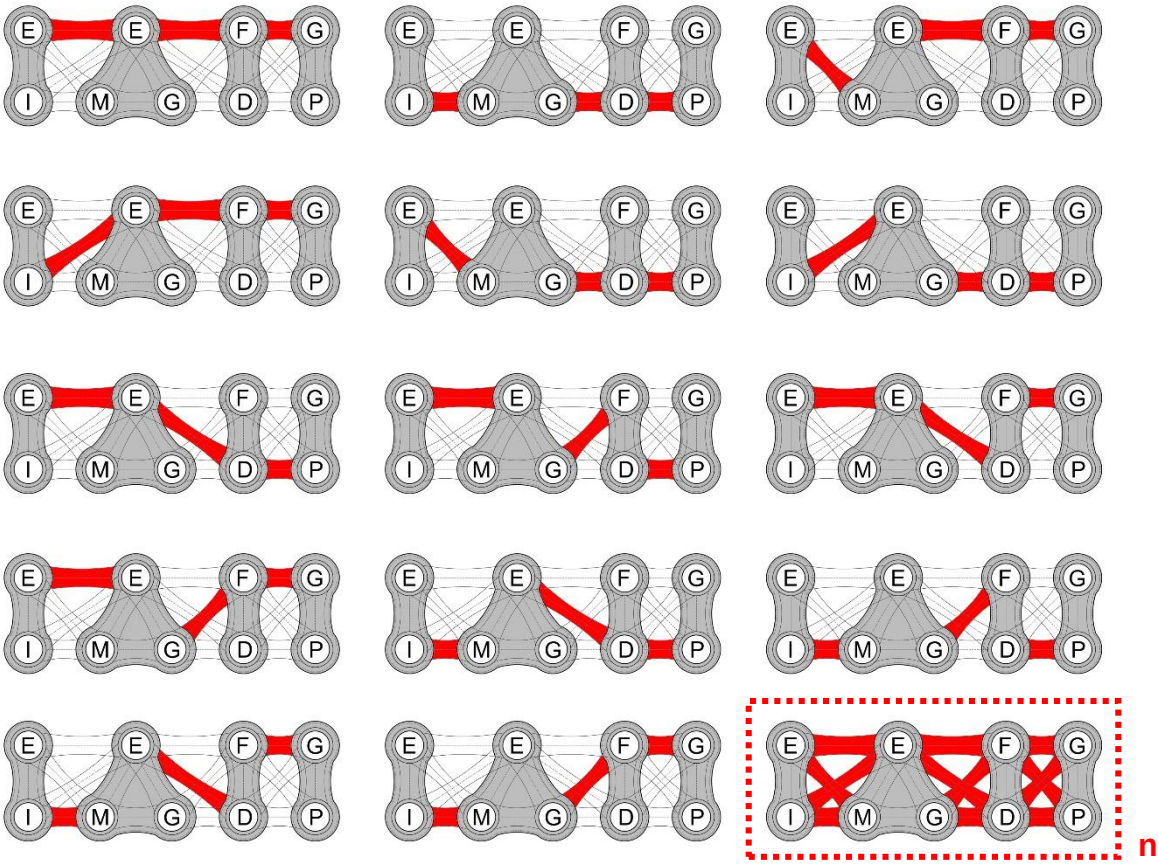


Figura 77. Cada experimento de combinaciones para la generación de un proceso de diseño es presentando en este diagrama con fin de demostrar algunas combinaciones entre el rendimiento, el comportamiento, la forma, el medio y la herramienta digital a utilizar. Bajo el grupo de rendimiento, están incluidos los criterios externos e internos relacionados con las constantes que son definidas como dependientes o independientes de material o materia a representar y que es parte del organismo. Bajo el grupo de forma están incluidos los parámetros en relación a la geometría, la estructura y el material. Por otra parte, bajo el grupo de medio de representación se incluyen los medios para llevar a cabo dicho experimento o proceso, sean estos físicos o digitales. Por último, bajo el cuarto grupo se exponen las herramientas digitales y su entorno de trabajo para llevar a cabo los cálculos de la simulación computacional a través de logaritmos de programación gráficos, como Grasshopper y logaritmos de programación textual, como Python, así como su representación geométrica digital final. Cada uno de estos experimentos de relación, exploran la relación entre dos o más elementos de cada grupo. En el último caso, esta combinación puede ser variable, hasta tener un número máximo de combinaciones posibles, representadas a través de la constante “n”.

La combinación seleccionada para llevar a cabo el primer caso de estudio es la siguiente:

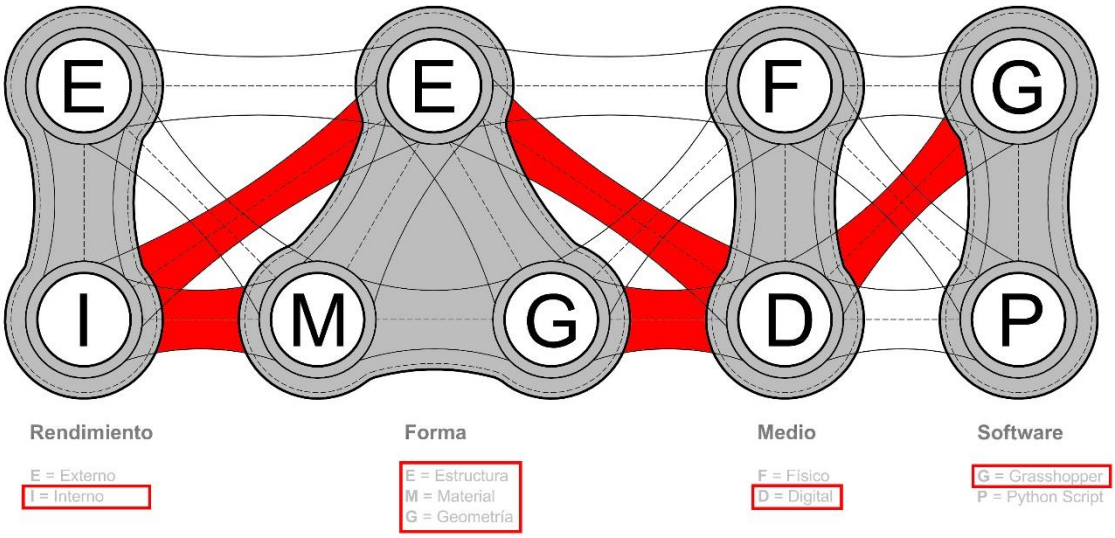


Figura 78. La combinación final seleccionada a través de un proceso de experimentación con las relaciones de grupos y sus variables particulares, referidas al rendimiento, la forma, el medio y la representación a través del software. Como primer concepto para análisis en el primer caso de estudio, se tomará como fundamento inicial el comportamiento y el rendimiento interno de una estructural natural. El segundo aspecto a considerar se relaciona con el material del organismo natural, la estructura a partir de las propiedades de la materia y la geometría como consecuencia del proceso de la generación artificial de la forma. El medio de representación será a partir de conceptos y herramientas computacionales digitales, a través del uso de software de programación gráfica de algoritmos *Grasshopper* que interactúan en conjunto con el programa *Rhinoceros 5.0.*

A partir de la referencia desarrollada a través del anterior diagrama, se desarrollará el caso de estudio de un coral de mar, de la familia Acrópora[55], por medio de un algoritmo que simulará el comportamiento del sol para controlar la generación del objeto geométrico (entorno natural externo) en un medio acuoso, como el mar (entorno natural de generación), y producir calcita como material formador de estructura; las propiedades de este organismo, así como su posterior estudio, simulación y representación por medio de un logaritmo de programación gráfica se explicarán a continuación.

55 El Acropora es un género de corales de la familia *Acroporidae*, orden *Scleractinia*. Su esqueleto es macizo y está compuesto de carbonato cálcico. Tras la muerte del coral, su esqueleto contribuye a la generación de nuevos arrecifes en la naturaleza, debido a que la acción del CO_2 convierte muy lentamente su esqueleto en bicarbonato cálcico, sustancia ésta asimilable directamente por las colonias coralinas. En este sentido, las especies de éste género son las principales constructoras de arrecifes del planeta. Entre las especies de *Montipora* y las de Acropora, suman más del tercio del total de especies coralinas constructoras de arrecifes.

4.7 – Proceso de diseño: fundamentos biológico - matemáticos

4.7.1 – Objetivo

El objetivo de la presentación final de este capítulo es explorar patrones y sistemas de crecimiento ramificados naturales, además de los atributos particulares (material) y externos (medio ambiental) que afectan y generan estos organismos a través de un algoritmo de programación gráfico. La especie particular de coral que servirá como inspiración y analogía a lo largo de este caso de estudio, además de la metodología a seguir será la *Acropora cervicornis*[56] (figura 79). Por otra parte, investigaremos instancias en la naturaleza como la luz y el movimiento del sol, geología de soporte del organismo y física para explorar cómo patrones "estándar" de la naturaleza pueden ser manipulados por medio de múltiples procesos y cambios en los parámetros de un algoritmo gráfico, ya sea en generación o crecimiento y en resultados geométricos variables que sean aplicables a problemas de diseño en el ámbito de la arquitectura.

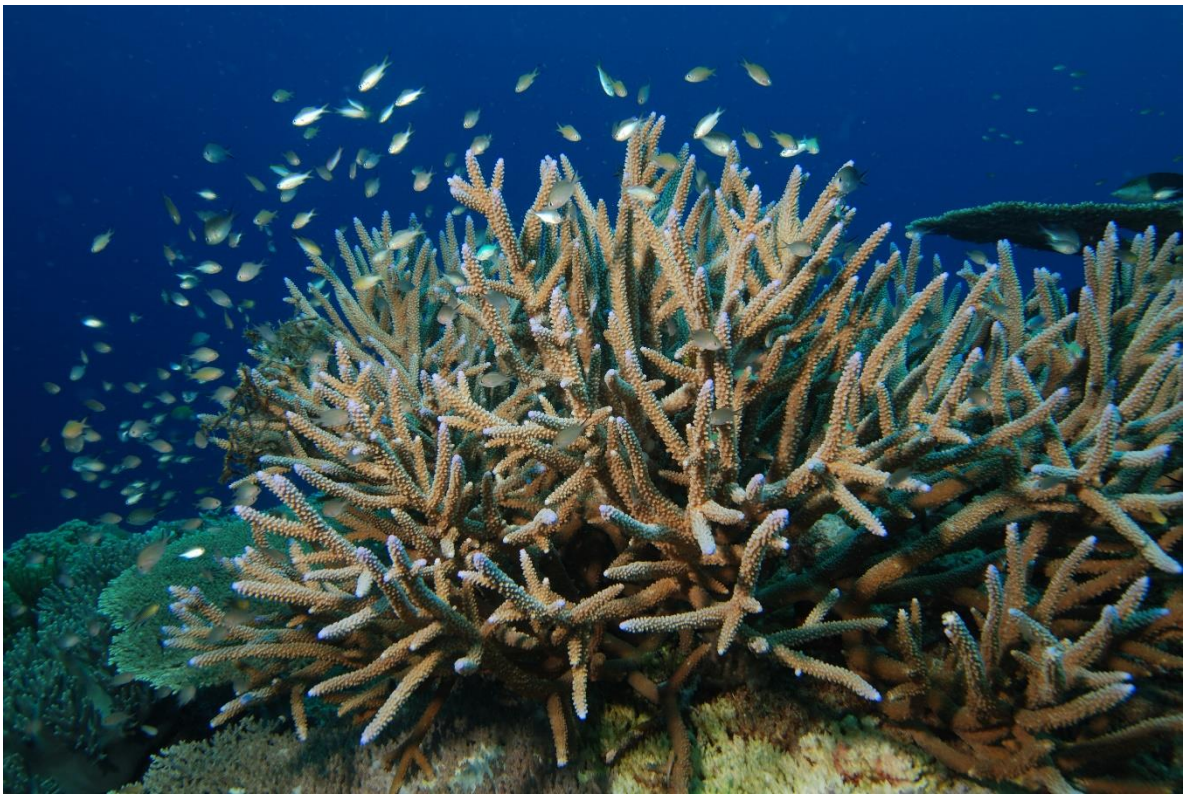


Figura 79. Fotografía de una familia (arrecife) de corales del género Acropora Cervicornis, en dónde las ramas del sistema reaccionan a cada cambio en los flujos del agua y tienden a elevarse a consecuencia de los rayos del sol que penetran a profundidades que van de 5 m a 25 m de profundidad. Una característica especial de este género es la capacidad del organismo para ramificarse según la necesidad de energía y alimento como mecanismo de adaptación y supervivencia en el mar. Fuente: Bonaire, 2007.

56 El coral cuerno de ciervo o coral cuerno de venado (Acropora Cervicornis) es una especie de coral que pertenece a la familia Acroporidae, orden Scleractinia. Su esqueleto es macizo y está compuesto de carbonato cálcico.

4.7.2 – Hábitat y distribución

Suelen vivir en arrecifes en zonas poco profundas, de 1 m a 40 m, ocasionalmente a 60 m y usualmente entre 1 m y 25 m, bien iluminadas y cercanas a las costas. Aunque también se encuentran en lagunas y zonas protegidas del arrecife, mayoritariamente se dan en zonas de fuertes corrientes.

Se distribuyen mayoritariamente en las costas occidentales del océano Atlántico central, en el Caribe y el Golfo de México. También en el Indo-Pacífico, en Chagos, Mauricio, Seychelles, Australia y Malasia.

4.7.3 – Morfología

Cualquier especie de *Acropora* puede variar su forma de crecimiento por su localización, condiciones ambientales, edad, tamaño, estación, etc. Así pues, la forma de crecimiento de la colonia de Acropora, puede tener o no una forma de crecimiento típica de la especie, lo cual hace muy difícil su identificación, que por lo general solo puede hacerse bajo el microscopio observando el esqueleto. El esqueleto es poroso y ligero.

En el caso que nos ocupa, la colonia crece en forma arborescente, con las ramas más espaciadas que en otras especies similares y las ramas secundarias creciendo en ángulo recto sobre la principal. Las ramas pueden medir, en condiciones ideales, entre 1 m y 2 m de largo y entre 4 cm y 7 cm de diámetro. Las colonias pueden medir varios metros. Son los corales de mayor crecimiento en los arrecifes, pueden crecer entre 10 cm y 20 cm al año, lo que se equilibra con el alto porcentaje de ramas rotas por las tormentas. Los pólipos[57] de Acropora son muy pequeños y presentan unas células urticantes denominadas nematocistos, empleadas en la caza de presas microscópicas de plancton. Esta especie en particular presenta los siguientes colores: amarillo y marrón.

4.7.4 – Alimentación

Los pólipos contienen algas simbióticas llamadas zooxantelas. Las algas realizan la fotosíntesis produciendo oxígeno y azúcares, que son aprovechados por los pólipos, y se alimentan de los catabolitos del coral (especialmente fósforo y nitrógeno). Esto les proporciona entre el 75 % y el 95 % de sus necesidades alimenticias. El resto lo obtienen atrapando plancton microscópico y materia orgánica disuelta en el agua.

4.7.5 – Reproducción - Generación

En general alcanzan la madurez sexual entre los 3 y 5 años, con un diámetro de sus "ramas" entre 4 cm y 7 cm. Se reproducen asexualmente mediante gemación y por fragmentación, siendo este último el modo dominante de reproducción, cuando las ramas de las colonias se rompen debido a los temporales, y sus fragmentos originan nuevas colonias.

57 En zoología, un pólipo (literalmente, "muchos pies") es una de las formas animales invertebradas del filo de los cnidarios, la otra es la medusa. Tienen forma de saco. En el extremo basal tienen una ventosa o disco pedal, por la que se fijan al sustrato, roca u objeto marino, como conchas de moluscos, mientras que, en el extremo opuesto, o apical, poseen un solo orificio, que hace de boca y de ano, rodeado generalmente de tentáculos, aunque algunos pólipos carecen de ellos. En la mayoría de los casos, los tentáculos poseen células urticantes, denominadas nematocistos, para atrapar a sus presas o defenderse de otros corales, en la lucha por el espacio estratégico en el arrecife.

Sexualmente son hermafroditas simultáneos, lo que quiere decir que las colonias generan gametos masculinos y femeninos, lanzando simultáneamente al exterior sus células sexuales, siendo por tanto la fecundación externa. Los huevos una vez en el exterior, permanecen a la deriva arrastrados por las corrientes varios días, más tarde se forma una larva plánula que, tras deambular por la columna de agua marina, y en un porcentaje de supervivencia que oscila entre el 18 y el 25 %, según estudios de biología marina, cae al fondo, se adhiere a él y comienza su vida sésil, secretando carbonato cálcico para conformar un esqueleto, o coralito. Posteriormente, forman la colonia mediante la división de los pólipos por gemación (figura 80).

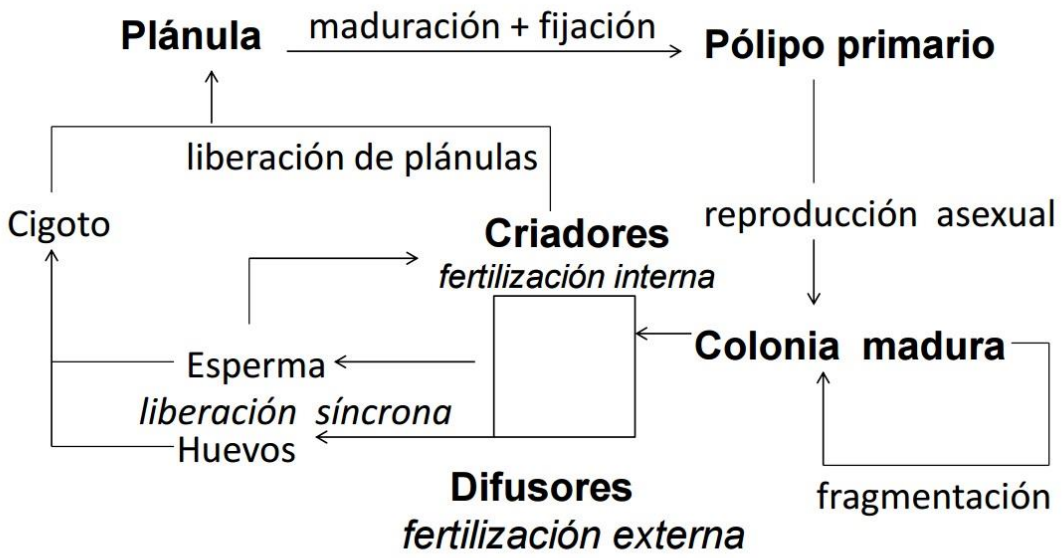


Figura 80. Los ciclos de vida y de generación de un coral Acropora.

Con el fin de investigar y producir un modelo en relación al crecimiento fractal y ramificado, es necesario conocer el proceso de generación y reproducción del coral, así como considerar datos cuantitativos en relación a los tamaños generales y particulares del sistema en sus diferentes edades, ya que diseñaremos con parámetros para simular este organismo. Para poder determinar un algoritmo de programación gráfico, necesitaremos simular al sol como un agente modificador que, según su movimiento en relación a un objeto adyacente, en este caso, el sistema de coral, sea capaz de modificar su generación y resultado geométrico físico. A través del uso de software como *Rhinoceros 5.0*, *Grasshopper* y los plugins complementarios *Anemone*, *Kangaroo* y *Cocoon*, determinaremos un conjunto de parámetros para poder controlar este sistema bio – computacional. Es necesario, en conjunto, crear un número de iteraciones de diseño que ayuden a la generación geométrica a partir de las cualidades anisotrópicas físicas y propiedades mecánicas del material a partir del cual está conformado este organismo.

Para poder determinar una fase inicial en este proceso de diseño, necesitaremos interpretar y entender ejemplos de estructuras fractales y de ramificación que se encuentran en la naturaleza, incluyendo árboles, anatomía humana, plantas y fractales en frutas y verduras. Por último, el estudio de estas características determinará e inspirará el proceso de diseño.

Por otra parte, un análisis matemático y físico es necesario para poder determinar inicialmente un proceso de diseño paramétrico, con el objetivo de generar un resultado geométrico que responda y sea variable en relación a parámetros que puedan ser modificados. El estudio de sistemas de ramificación en 2D y en 3D nos ayudará a establecer características y descripciones geométricas en la etapa inicial de diseño.

4.7.6 – Precedentes biológicos

La investigación de estructuras fractales y de ramificación en la naturaleza, es importante durante la primera etapa de diseño. Lo anterior incluye estructuras de árboles, como el análisis anisotrópico de las fuerzas que dan lugar a la modificación del tejido de la madera y a la forma final; la anatomía humana, por ejemplo, en los sistemas de ramificación de los bronquios de los pulmones, así como también el proceso de crecimiento de frutas y plantas bajo la influencia externa del sol y la fuerza de gravedad. Por lo anterior, el estudio de los corales es un ejemplo relevante, ya que involucra un sistema de crecimiento y conformación geométrica en ramificación, este sistema cumple la función de alimentación del organismo, así como los bronquios cumplen la función de distribuir la cantidad y el sentido del aire (derecho e izquierdo) en los pulmones humanos (figura 81) y por último, se encuentran expuestos y adaptados a condiciones externas, tales como la energía y la luz del sol, en conjunto con las fuerzas que producen las corrientes de agua en dónde se desarrollan (medio externo).

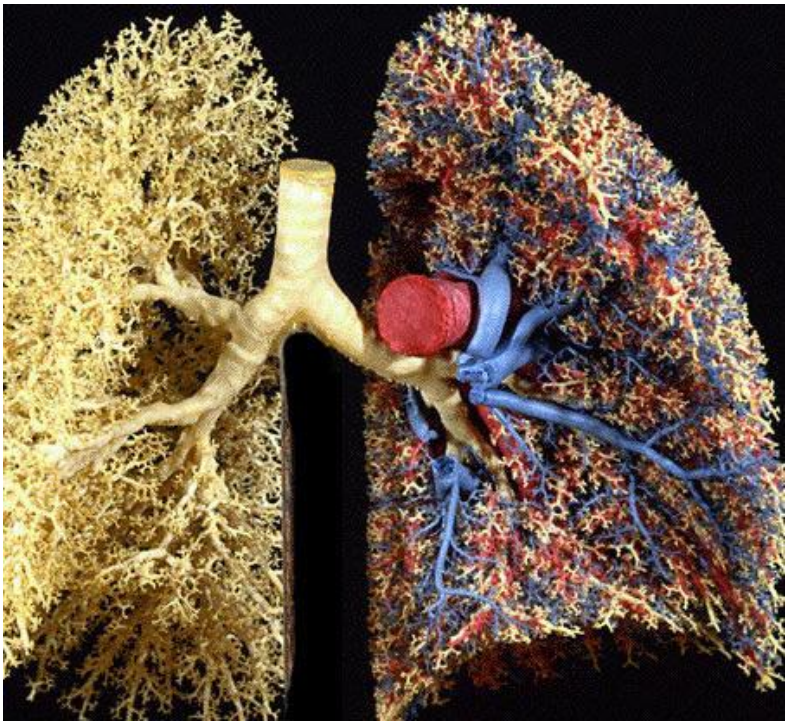


Figura 81. Patrones de crecimiento en ramificación encontrados en los bronquios de los pulmones humanos. La función principal de estos órganos es distribuir la cantidad total de aire aspirado a través de la tráquea hacia el pulmón derecho e izquierdo respectivamente, resolviendo así, un trabajo equitativo para cada pulmón, la función biológica que da lugar a la forma. Fotografía B. Sapoval.

Una cabeza de coral, es una colonia genéticamente idéntica de pólipos. Cada pólipo comúnmente es un animal invertebrado de unos pocos milímetros de diámetro y unos pocos centímetros de longitud. Un conjunto de tentáculos rodea el centro de cada apertura de las bocas de estos órganos del coral. Las cabezas de los pólipos crecen y se generan a través de reproducción asexual[58] de los pólipos. Un exoesqueleto es generado a partir de un proceso de síntesis de alimento (plancton microscópico) propio del organismo, que es conformado de carbonato cálcico; de este modo, durante muchas generaciones, se pueden crear colonias de exoesqueletos cada vez más grandes y a partir del mismo sistema de ramificación, es decir, en términos de generación matemáticos, es un crecimiento exponencial (1 – 2 – 4 – 16 – *n*); y de manera biológica, es el mismo sistema genético, ya que es inicio y fin de un solo organismo (figura 82).

Figura 82. Imagen realizada a través de un microscopio electrónico de una cabeza de coral, en dónde se aprecian los pólipos y las generaciones iniciales de ramificaciones de los exoesqueletos futuros del sistema conformados por carbonato cálcico, propiedad del organismo que le permite generar resistencia y anclaje a un medio ambiental en constante movimiento. Fotografía de Dave Fleetham.

58 La Reproducción Asexual consiste en que de un ser vivo ya desarrollado, se desprende una sola célula o trozos del cuerpo, los cuales por procesos mitóticos son capaces de formar un individuo completo, genéticamente idéntico al primero. Se lleva a cabo con un solo progenitor y sin la intervención de los núcleos de las células sexuales o gametos.

4.7.7 – Precedentes enfocados al proceso de diseño

Patrones de Fractales

La comprensión de patrones en fractales de crecimiento 2D y 3D es relevante, dado el mecanismo de generación de exoesqueletos ramificados anteriormente explicado, con el fin de establecer parámetros geométricos y numéricos que nos permitan generar, manipular y modificar formas unitarias específicas que afectarán al sistema general, produciendo complejidad geométrica. Estos patrones fractales de crecimiento van desde estructuras geométricas muy simples hasta muy complejas por medio de la repetición de la forma y la manipulación del tamaño y la escala de cada unidad. La reinterpretación y analogía a un algoritmo de programación es importante, ya que debemos de tener control en cada elemento geométrico que conformará el sistema final, y el coral, como tal, es una estructura que tiene principios idénticos a un fractal 3D en ramificación (figura 83).

Figura 83. Ejemplo de un patrón de crecimiento fractal 3D, presente en un *Romanescu*, una especie de coliflor. Fotografía de Gregory Bard.

Crecimiento Natural

El crecimiento es el principio elemental de generación de este sistema, el cual se encuentra influenciado por el metabolismo propio del organismo, así como por su sistema reproductivo, lo que buscamos es poder “estimular” a la geometría (patrón de crecimiento natural) por medio de la atracción del sol (agente modificador – generador) y el movimiento del agua, mediante la simulación y aplicación de las fuerzas físicas presentes en el agua a través de las corrientes marinas. Estos dos conceptos se traducen como los aspectos ambientales externos al organismo que podrán influir en el proceso de crecimiento y conclusión geométrica (figura 84).

Enfoque matemático - geométrico

Es necesario establecer un proceso matemático para poder determinar de manera inicial un método de diseño paramétrico. Con el objetivo, además, de generar una forma paramétrica, una estructura de ramificación es la más adecuada para poder describir la generación y resultado geométrico requerido para la representación de esta especie de coral.

- **Random 2D**: el primer método para tratar de imitar y representar un patrón de crecimiento y generación en la naturaleza tiene que ser *random*, ya que la capacidad de “creación” que la naturaleza lleva a cabo queda exenta de las capacidades de producción del ser humano. Este método random 2D será representado mediante líneas y sus respectivas subdivisiones. El patrón de crecimiento 2D nos permitirá controlar de manera satisfactoria el crecimiento en un sistema geométrico en dos dimensiones, que nos proporcionará “orden” y servirá de base del sistema completo, mediante la asignación de parámetros controlados por medio del número de divisiones de los segmentos que podría tener nuestro sistema inicial.

- **Ramificación 2D**: este sistema será desarrollado a partir del concepto de división de un componente (línea) en sub – componentes idénticos (sub – líneas), partiendo del estudio de los sistemas naturales de plantas, árboles, y en específico, de los corales. El desarrollo de un sistema basado en ramificaciones a partir de la expresión (XY / 4) de cada una de la líneas iniciales y generadoras de este sistema 2D, nos permitirá establecer un ciclo de crecimiento basado en una constante matemática exponencial 2D. Este ciclo puede ser expresado a través de una regla matemática, referida al número de ciclo e iteraciones producidas, 1 = 1, 2 = 4, 3 = 16, y de forma progresiva; dónde cada unidad representa un ciclo de crecimiento y los números generados, los sistemas de ramificación correspondientes a cada ciclo en el plano 2D (figura 84).

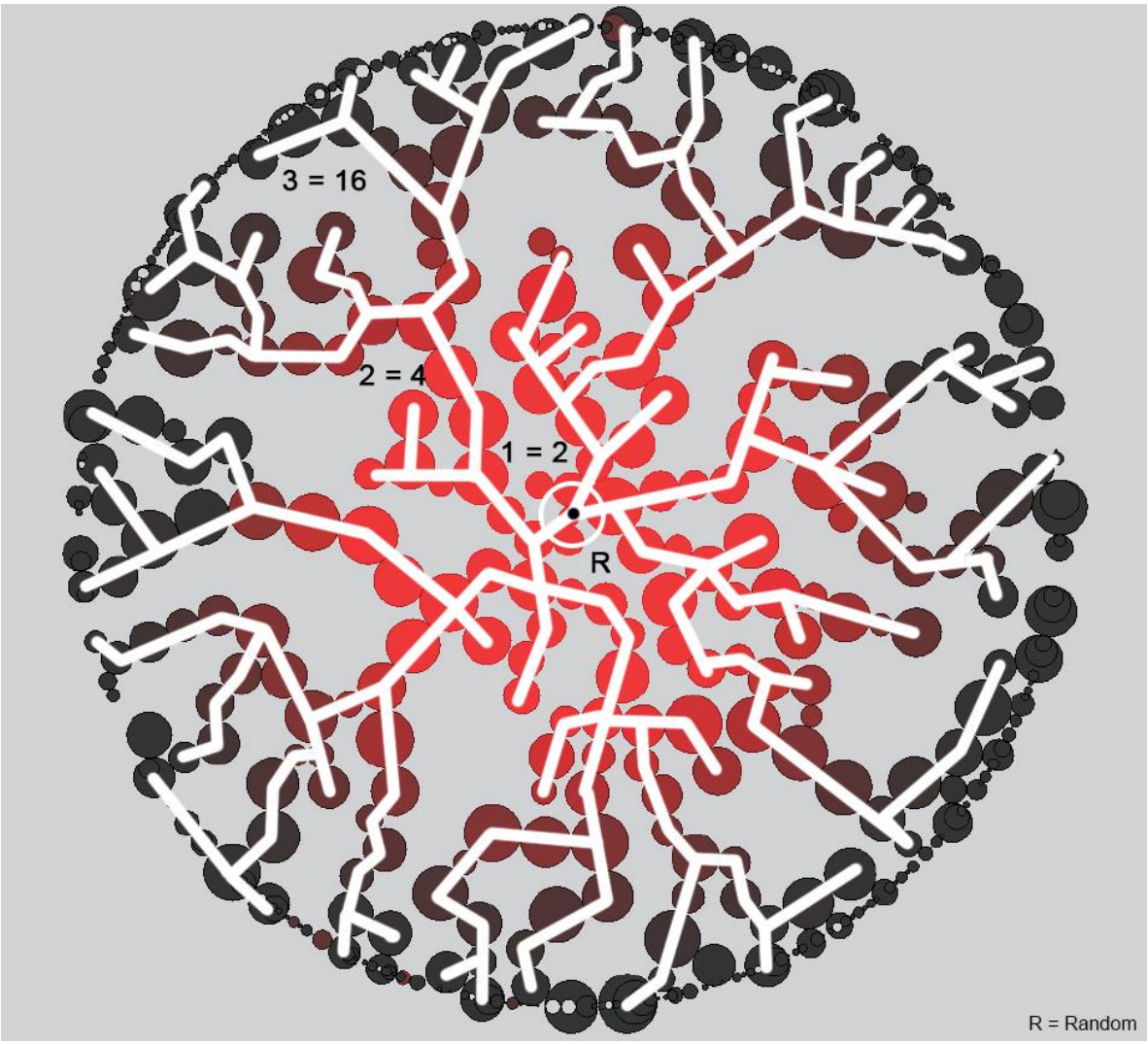


Figura 84. Ejemplo de patrón de crecimiento ramificado en 2D. Fuente: Andrea Rossi, con edición posterior de Alan Rodríguez.

- **Random 3D:** las estructuras encontradas en la naturaleza tienden a desarrollarse y crecer de manera aleatoria, afectadas y en respuesta al medio ambiente natural externo, como controlador. La tendencia a seguir un comportamiento de generación en específico se debe en parte a la procedencia de la luz y energía del sol o a la absorción y búsqueda de agua del organismo, por ejemplo, una planta. Por lo tanto, así como los seres humanos y las formas de vida de este planeta, estas fuentes de energía causan influencia directa en los dos. Esta interpretación deberá ser llevada a cabo por medio del rendimiento y respuesta de la simulación generativa, a través del movimiento del sol y las fuerzas presentes en el ambiente externo del sistema geométrico. El sol puede ser representado por medio de un punto atractor que puede modificar su posición en el espacio 3D de manera aleatoria o controlada, mientras que las fuerzas pueden ser reproducidas mediante componentes de gravedad y viento utilizando el plugin para Grasshopper Kangaroo. El movimiento del punto atractor (sol) nos dará la oportunidad de tener diferentes configuraciones espaciales y geométricas en el sistema y, por otra parte, las fuerzas brindarán la oportunidad

de obtener variabilidad geométrica en respuesta a la aplicación de fuerzas en el sistema en conjunto.

- **Ramificación 3D:** al tener un objeto en conjunto estático, y con soporte en un medio estático, como una roca o un cuerpo sin inercia, el empleo de un sistema de ramificación 3D nos ayudará a controlar y representar de mejor manera un resultado geométrico en respuesta a las constantes físicas, tales como la gravedad o el viento interpretadas mediante algoritmos y componentes de simulación física, tal como *Kangaroo*. A través de un sistema de ramificación en 3D, el patrón de crecimiento fractal y la geometría representarán nuestro organismo físico, que resolverá de manera satisfactoria la imitación (biomímesis) de nuestro sistema natural, además de poder controlarlo mediante datos y parámetros matemáticos (figura 85).

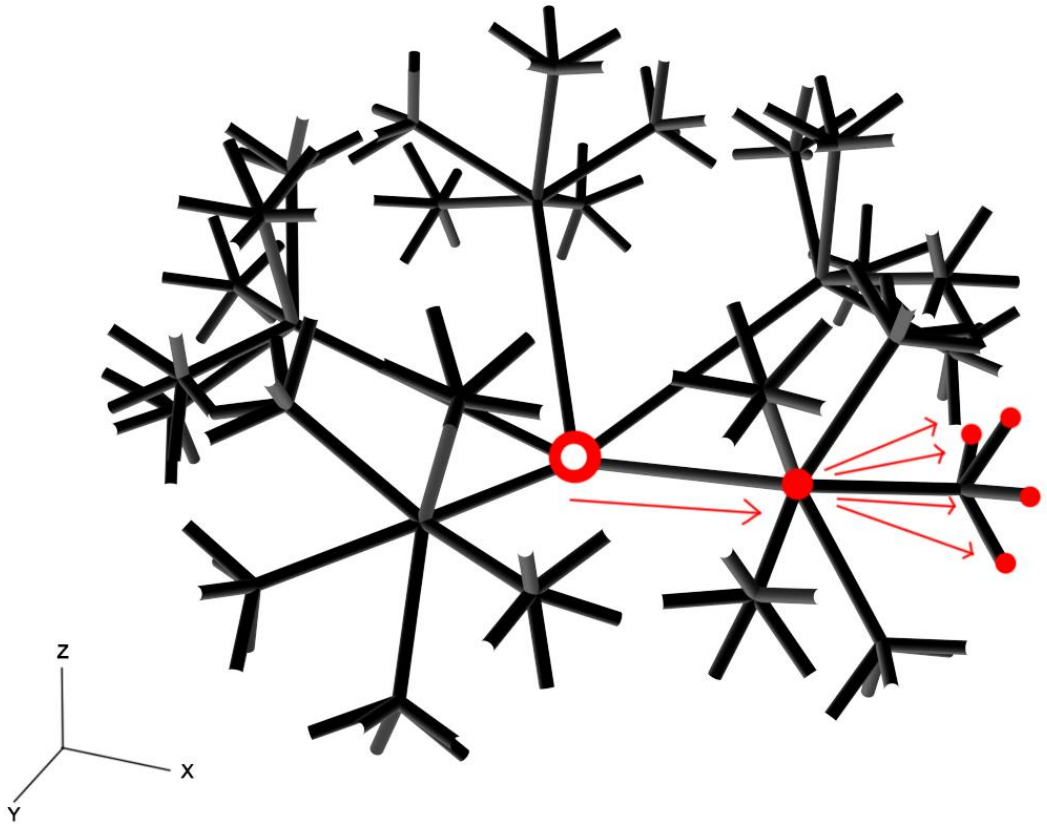


Figura 85. Ejemplo de patrón de crecimiento ramificado en 3D. Fuente: Alan Rodríguez Carrillo.

Enfoque Computacional

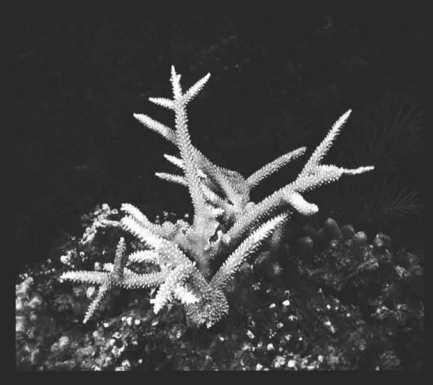
- ¿Cómo se puede lograr la simulación computacional de la materia y la energía, ligadas a la creación de la forma, sin crear primero la forma en sí? -

El camino más complejo para entender y ver en la naturaleza es el estudio de su comportamiento y generación, además de la imitación a partir de teorías científicas y numéricas que los humanos hemos desarrollado. La interpretación de parámetros ambientales y geométricos que han sido estudiados anteriormente nos ha dado un punto de inicio para generar un estudio basado en parámetros de programación gráfica computacional con el fin de representarlo en un modelo geométrico digital paramétrica. En consecuencia, cada elemento y concepto hasta aquí estudiado va a tener influencia en la interpretación aplicada al diseño mediante la programación computacional a través de conceptos físicos, geométricos y matemáticos que involucra la utilización de un software, en este caso particular, *Rhinoceros* y *Grasshopper*.

Enfoque y concepto de diseño

El concepto de diseño abordará controles de manipulación a través de parámetros en patrones de crecimiento ramificado y fractal de corales, generados mediante un algoritmo de programación gráfica; con el objetivo de representar un sistema orgánico natural mediante una imitación artificial representada a través de geometría digital. A partir de aquí, la tarea recaerá en cómo poder manipular y controlar este sistema producido de manera artificial.

Con el fin de representar de mejor manera lo antes descrito, se desarrolló un esquema conceptual a aplicar durante este proceso de diseño en particular, tomando como referencia los temas y elementos desarrollados a lo largo de este capítulo, con el fin de exponer los conceptos de manera más sencilla, relacionando a la biología, la computación y el diseño (figura 86).

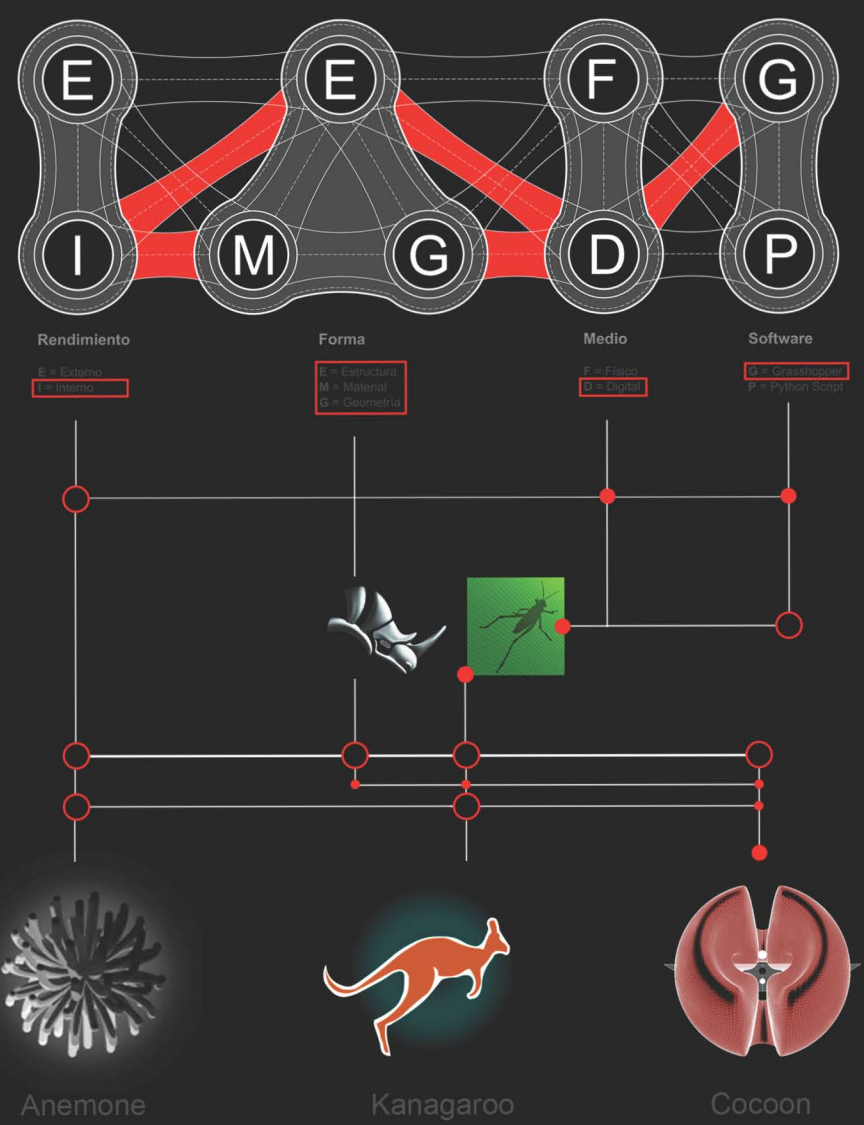


Acropora Cervicornis
[Coral Cuerno de Ciervo]

Reino:	*Animalia*
Filo:	*Cridaria*
Clase:	*Anthozoa*
Subclase	*Hexacorallia*
Orden:	*Scleractinia*
Familia:	*Acroporidae*
Género	*Acropora*
Especie:	*A. Cervicornis*

LAMARCK, 1816

En peligro crítico de extinción



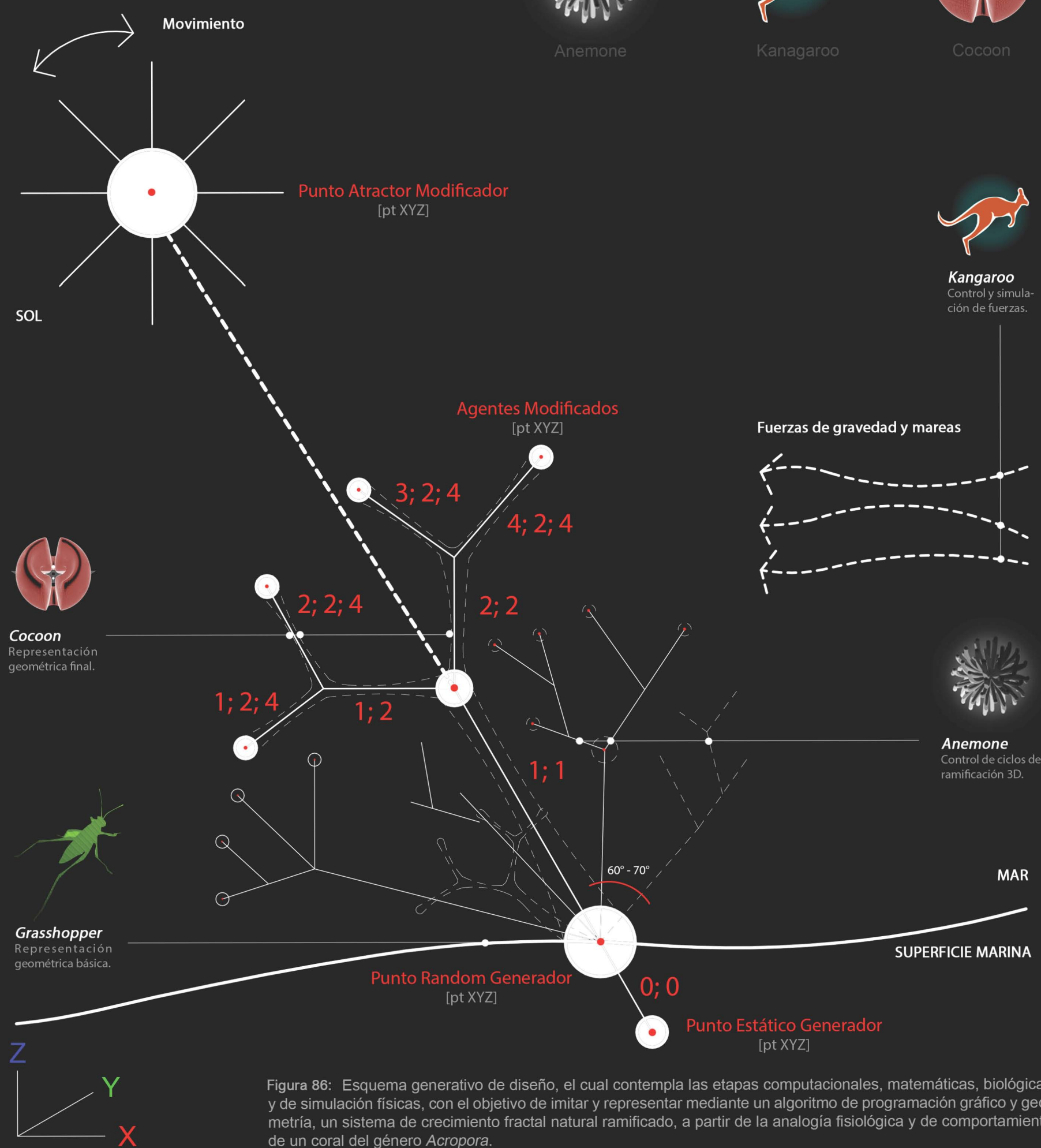


Figura 86: Esquema generativo de diseño, el cual contempla las etapas computacionales, matemáticas, biológicas y de simulación físicas, con el objetivo de imitar y representar mediante un algoritmo de programación gráfico y geometría, un sistema de crecimiento fractal natural ramificado, a partir de la analogía fisiológica y de comportamiento de un coral del género *Acropora*.

4.8 – Enfoques aplicados al proceso de diseño

El anterior esquema (figura 86), muestra los distintos enfoques que se desarrollarán y aplicarán a un proceso de diseño basado en la biomimética y la computación. En relación al proceso de diseño, es necesario realizar y limitar aquellas metodologías que nos ayudarán a desarrollar un modelo paramétrico de manera satisfactoria; para ello, tendremos que establecer enfoques de diseño computacionales y etapas de diseño, con el fin de obtener un producto geométrico representativo y adecuado del organismo biológico. Este tipo de enfoque estará basado en las ramas del conocimiento descritas a lo largo de este documento y su relación directa entre ellas. Es por eso que delimitaremos las etapas de diseño a cada uno de estos enfoques a aplicar mediante la programación logarítmica gráfica.

1. **Etapa – Enfoque:** Biología + Programación + Matemática + Física + Geometría
2. **Etapa – Enfoque:** Física + Programación + Matemática + Geometría
3. **Etapa – Enfoque:** Física + Programación + Geometría
4. **Etapa – Enfoque:** Biología + Programación + Geometría
5. **Etapa – Enfoque:** Biomimética + Geometría.

La aplicación y delimitación de etapas - enfoques durante este proceso de diseño, nos ayudará a desarrollar los conceptos e ideas de una manera sistematizada, con objeto de representar un organismo biológico, propio del aspecto cognoscitivo del ser humano en su deseo de la comprensión lógica de la naturaleza. La relación de las disciplinas en cada etapa de diseño, por otra parte, nos proporcionará las bases teóricas y técnicas a aplicar durante esta metodología, a través de la programación computacional gráfica. Otro aspecto importante, está relacionado con la experimentación e imitación del organismo a estudiar, ya que por medio de parámetros matemáticos y físicos, seremos capaces de relacionar estos conceptos con la geometría resultante a partir de la modificación de números y variables presentes a lo largo del desarrollo de la definición en la Gráfica Acíclica Dirigida (*DAG*), que nos proporcionará un control “artificial” de generación geométrica que esté relacionado con el aspecto físico y de comportamiento interno de un organismo biológico.

Es necesario realizar una analogía a las fases del proceso de diseño desde el punto de vista de generación geométrica computacional, y de manera más específica, al software que será la herramienta de generación, en este caso, *Rhinoceros 5.0* y *Grasshopper*. Por lo anterior, la analogía se representa a través de una serie de pasos que serán llevados a cabo con el software antes descrito:

1. Geometría de soporte a partir de una superficie 3D – ***Rhino + Grasshopper***
2. Generación de ciclos de ramificación – ***Grasshopper + Anemone***
3. Presión del agua y velocidad de corrientes marinas – ***Grasshopper + Kangaroo***
4. Modificador de crecimiento [Sol] – ***Rhino + Grasshopper***
5. Representación de geometría afectada – ***Grasshopper***
6. Variación en geometría de crecimiento – ***Rhino + Grasshopper***
7. Exclusión geométrica final – ***Grasshopper***
8. Representación geométrica final – ***Grasshopper + Cocoon***

La primera fase de este caso de estudio contemplará las etapas ***1 - 1*** y ***2 – 2.***

4.8.1 – [1] _ Etapa – Enfoque: [1 – 1] y [2 – 2]

La aplicación del primer enfoque de diseño contempla la relación directa de las disciplinas complementarias, de manera paramétrica, es decir, un cambio en una variable matemática, tendrá impacto y modificación directa en un componente geométrico, pero a su vez, estará regido por un límite, por ejemplo, el grado de apertura máximo de las ramificaciones fractales iniciales generativas del organismo, relacionado con el movimiento del sol y las corrientes marinas. Por lo anterior, este primer enfoque contempla a la biología y en específico, a la biomimética, ya que tendremos que establecer aquellos límites iniciales de control de nuestro organismo, que responde al medio ambiente externo (agua, movimiento, posición del sol), y lo traduciremos a un lenguaje de programación lógico gráfico; este tipo de metodología es relevante al contemplar a la biomimética como disciplina limitante de generación formal, ya que a partir del estudio de las propiedades anatómicas del organismo, podremos identificar las limitantes vectoriales y numéricas que representen de manera satisfactoria al espécimen biológico natural.

Como primer paso en este enfoque de diseño, tendremos que identificar los parámetros variables y constantes que rigen la generación y forma final del organismo a programar. El esquema (figura 88), muestra las características físicas y generativas más importantes de este organismo desde el punto de vista geométrico y físico – matemático; una de esas características es que posee un grado de ramificación limitado por la separación angular entre sus componentes, un punto de inicio y generación a partir de la superficie marina y un límite de ramas o elementos genéticamente idénticos regidos por la energía y movimiento del sol y las corrientes marinas. Tales descripciones y representaciones a través del dibujo, son el punto de inicio a contemplar durante este enfoque. Para proporcionar una mejor idea de lo descrito y dibujado, analizaremos lo anterior mediante una fotografía (figura 87) de dicho organismo.

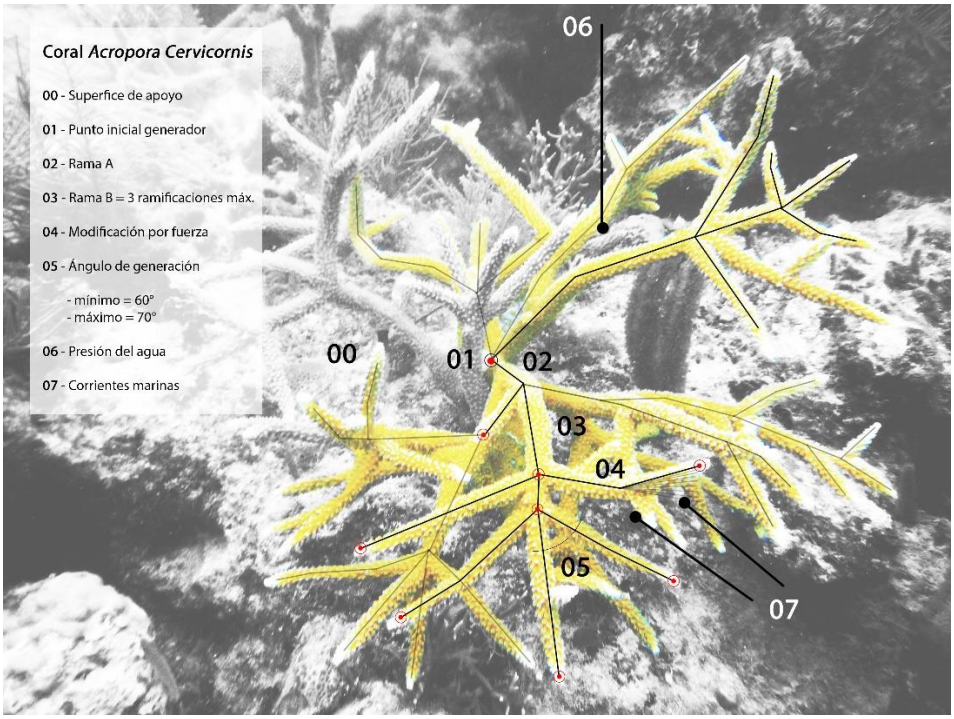


Figura 87. Configuración de generación de forma, comportamiento y respuesta al medio natural externo de un coral de mar del género Acropora. Fuente: Fotografía de biólogo y ecólogo Dr. Diego Lirman. Rosenstiel School of Marine and Atmospheric Science, Miami. 2012. EUA.

Como principio de análisis, el organismo posee un soporte inicial, que representa la superficie marina, así como el inicio de la ramificación de soporte y generación principal de todo el sistema. A partir de esta ramificación surgen las subyacentes, que poseen un número máximo de divisiones restringido a tres ramas. Por otra parte, cada una de las ramificaciones que conforman a todo el sistema, se encuentran expuestas a un entorno externo (mar) y por consiguiente, a fuerzas que modifican su crecimiento y propagación a lo largo de la vida del organismo; las ramas, además, están restringidas por ángulos de generación ramificados a 60° y 70° respectivamente, que son modificados por las fuerzas que interactúan de manera directa [06 – 07] con ellas.

A partir de un análisis analógico (biomimética) del organismo natural (figura 87) y en relación al esquema lógico de generación de la forma a través de la programación lógica gráfica computacional (figura 86), podremos determinar la relación directa que existe entre la lógica natural y la lógica artificial cognoscitiva humana, con objeto de realizar un proceso de diseño basado en un agente biológico natural y representar cuantas más características sean posibles, a través de un logaritmo (*DAG*) y geometría.

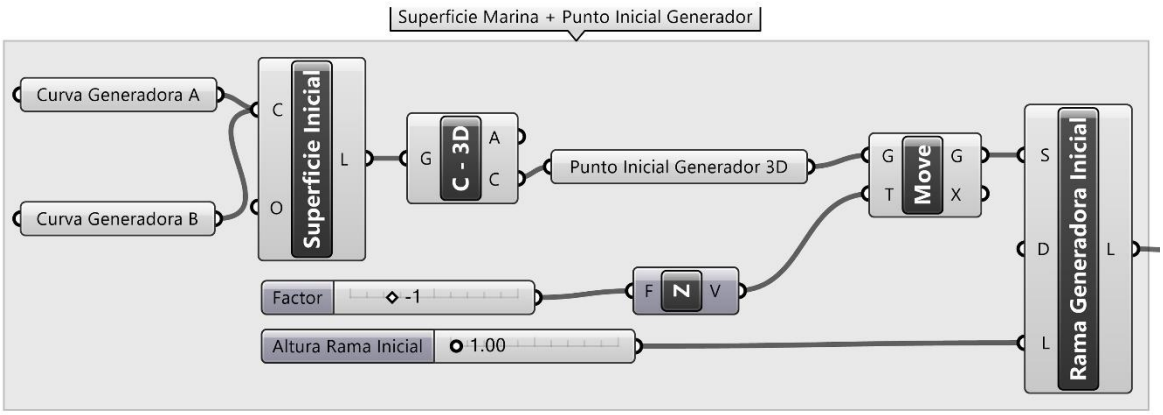
En relación al estudio y representación del proceso de diseño específico, tomando como referencia los esquemas analógicos anteriores, cada "*etapa – enfoque*" desarrollada en este capítulo será dividida en grupos de parámetros y componentes que representen cada una de las fases de generación computacional; los análisis, geometría y simulaciones serán desarrollados en el entorno de modelado de *Rhinoceros 5.0* y el entorno de programación de logaritmos gráficos *Grasshopper 0.9.0076*, por lo que los grupos representarán la generación matemática, geométrica y física de las etapas de diseño, además de ser los componentes a partir de los cuáles se cuantificará el nivel de complejidad ciclomática, latencia y modificación en la definición del logaritmo.

[A] – Superficie Marina + Punto Inicial Generador = Soporte del Sistema

La generación de grupos a través de una gráfica de programación gráfica es un aspecto relevante, ya que contempla todos los parámetros y componentes para la producción de un resultado geométrico paramétrico, por lo anterior, debemos de establecer jerarquías para cada y en cada grupo del algoritmo inicial, además de definir una terminología que nos permita distinguir cada proceso que llevará a cabo el algoritmo, que calculará la computadora y que será representado a través de la geometría. Los grupos se distinguirán por medio de letras del alfabeto griego incrementales, que representarán cada una de las fases específicas del proceso de diseño.

Grupo A – Superficie marina + Punto Inicial Generador

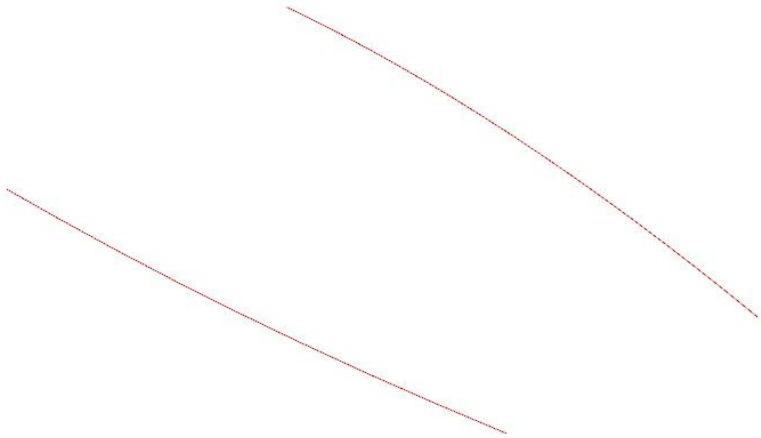
Para poder distinguir y comprender cada una de las fases de generación, se colocará el grupo final de tal fase, y se explicará a detalle cada uno de sus componentes y parámetros en relación al diseño paramétrico generativo.



Grupo A: Esquema específico ejemplo de una gráfica DAG, que muestra la fase inicial de generación a través de una superficie geométrica y un punto centroide 3D a partir de ella.

La fase inicial contempla la imitación del soporte del coral Acropora, es decir, el medio de sustento físico a partir del cual se genera. Este soporte puede ser interpretado a partir del análisis y en base a una referencia analógica de la superficie o rocas marinas en dónde se desarrolla el organismo; en este caso de estudio, una superficie marina será contemplada, ya que el organismo se desarrollará en un entorno de modelado computacional 3D y no existen organismos "digitales" acompañantes en esta bio – simulación, lo que implica que no hay un arrecife o rocas formadas que le puedan dar origen.

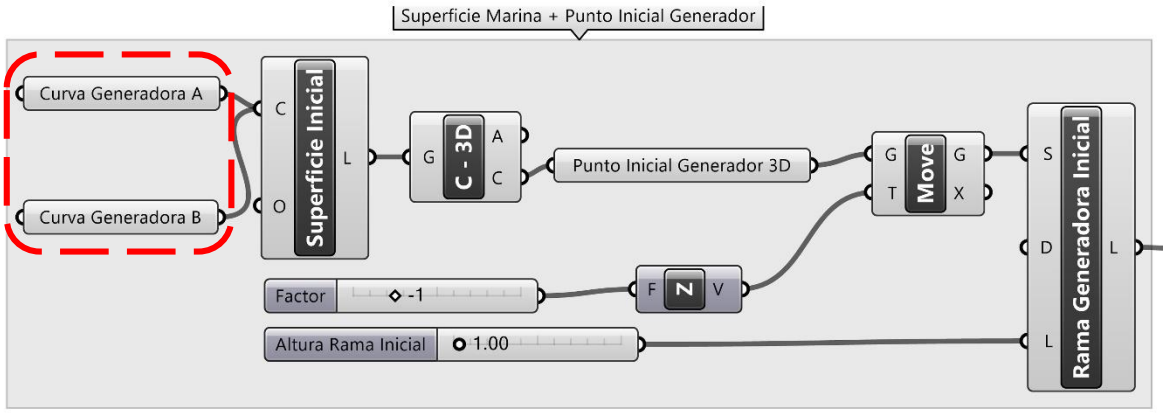
La primera premisa es generar dos curvas a partir de tres puntos de control, para poder modificar la altura de los puntos en el eje "Z" y obtener una variación de altura que nos permita imitar a una superficie marina irregular por medio de una superficie geométrica 3D. Para la generación de las dos curvas se utilizó el componente *"curve from points"* de *Rhinoceros 5.0* y el modificador de desplazamiento "*move*" para incrementar la altura de los puntos de las dos curvas a lo largo del eje "Z", y a 2 m de distancia en el eje "Y" (crecimiento máximo del organismo) y modificar los puntos a 1 m (altura máxima del organismo 1.50 m). Como resultado obtenemos dos curvas que deben de ser unidas en relación vectorial y lineal - geométrica, es decir, conformar una superficie 3D.



01 – Generación de dos curvas de control primario en Rhinoceros 5.0

El siguiente paso es generar una superficie entre las dos curvas ilustradas que sirva de soporte para nuestro sistema de ramificación fractal inicial. La generación de un algoritmo gráfico que nos permita realizar lo antes mencionado será desarrollado a través de *Grasshopper*, por lo que tendremos que hacer una referencia de geometría *"física"* presente en el entorno de modelado 3D de *Rhinoceros 5.0* a un componente de referencia geométrica de *Grasshopper*, es decir, vamos a traducir geometría visual a geometría de datos a través de un componente de curvas que nos brinda este entorno de programación.

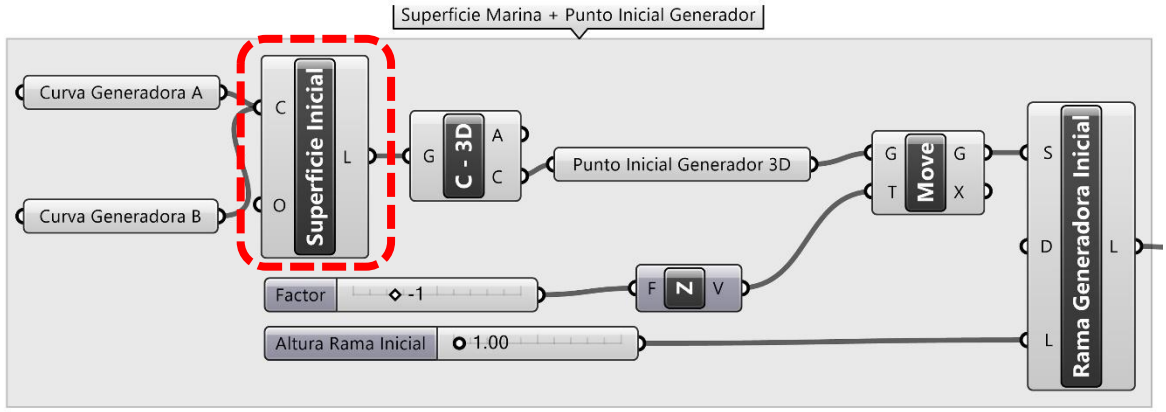
El procedimiento anterior descrito es importante de explicar a detalle y comprender, ya que después de esta fase, de cambio de geometría a datos, el proceso de diseño se relacionará de manera directa con la manipulación de datos y parámetros en el entorno de programación gráfica. A continuación, interpretaremos la geometría física a datos mediante el siguiente esquema:



01A – Traducción de geometría *"física"* a datos mediante una referencia al componente *curve* de Grasshopper.

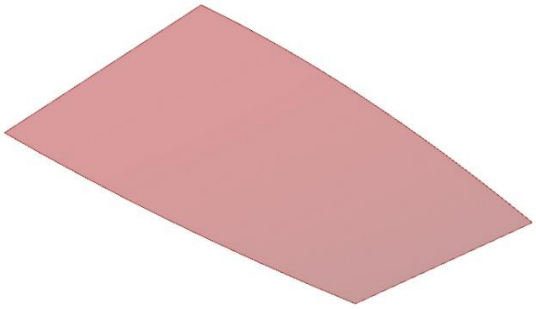
Al haber realizado la referencia y relación geométrica de datos, cualquier modificación que se aplique a las curvas *"físicas"*, serán interpretadas por el componente de *Grasshopper* que modificará todos elementos posteriores durante el proceso de diseño, cumpliendo una de las premisas iniciales de diseño paramétrico.

Las referencias de datos, en este caso de curvas, nos permitirán continuar con la generación de la superficie de soporte inicial:



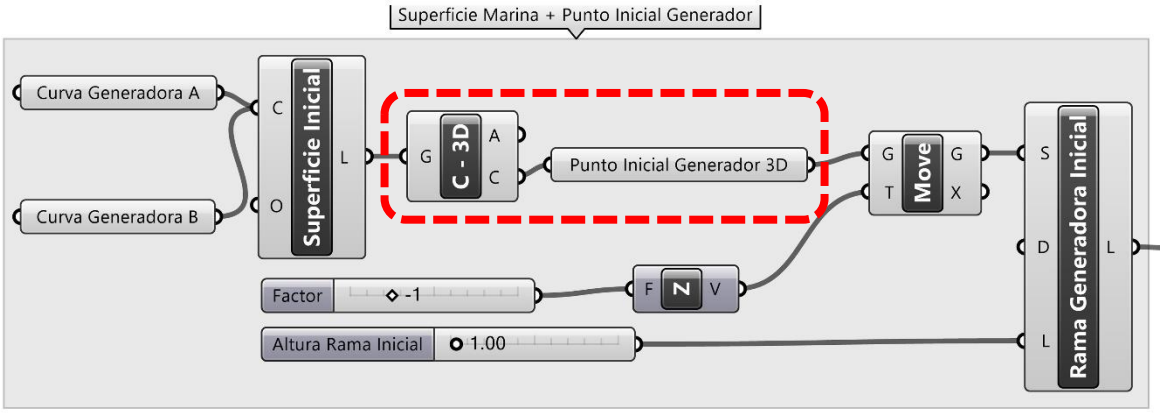
02 – Relación de curvas A y B al componente *loft* de Grasshopper para generar una superficie 3D.

En este caso la relación de curvas con el componente *loft* de *Grasshopper* nos brindará la oportunidad de generar una superficie controlada y variable por medio de los puntos de control de las dos curvas. Esta superficie ahora generada y representada en el entorno de trabajo de *Rhinoceros 5.0*, será el elemento geométrico inicial de generación y simulación del organismo a programar y representar.



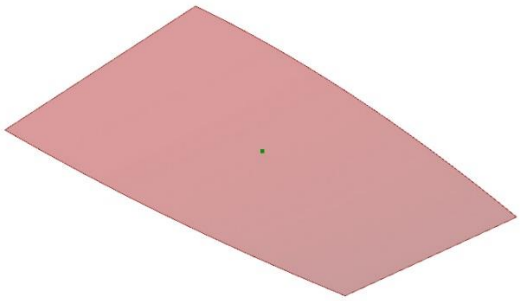
02A – Superficie de generación inicial.

Un ciclo de ramificación puede ser interpretado como un inicio, y de manera geométrica, el inicio de todo cuerpo geométrico es un punto, es por lo anterior que necesitaremos localizar un punto en relación a la superficie 3D generada:



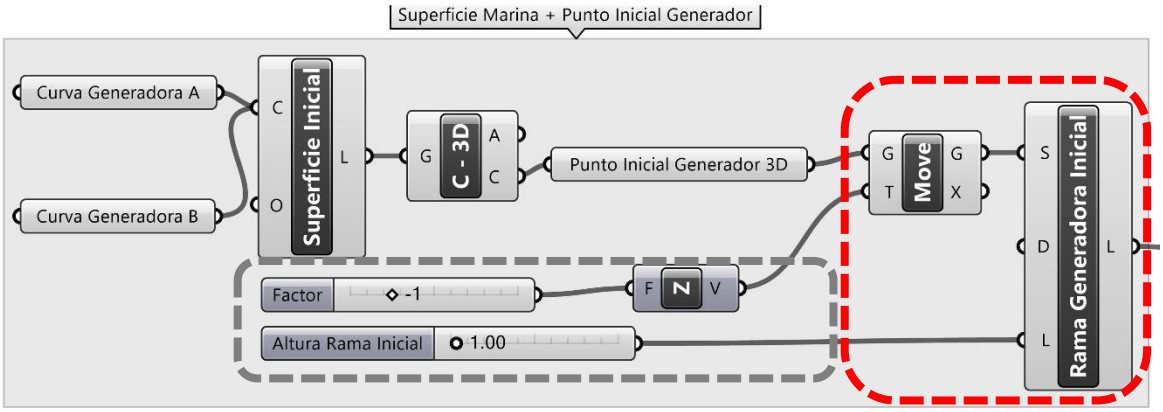
03 – Localización del centroide de la superficie 3D por medio del componente *mesh area 3D* y *point.*

El componente *mesh area 3D* (C – 3D), localizará y generará un punto central 3D en la superficie generada, si la superficie es modificada, el punto cambiará de posición, adaptándose al medio geométrico generador inicial.

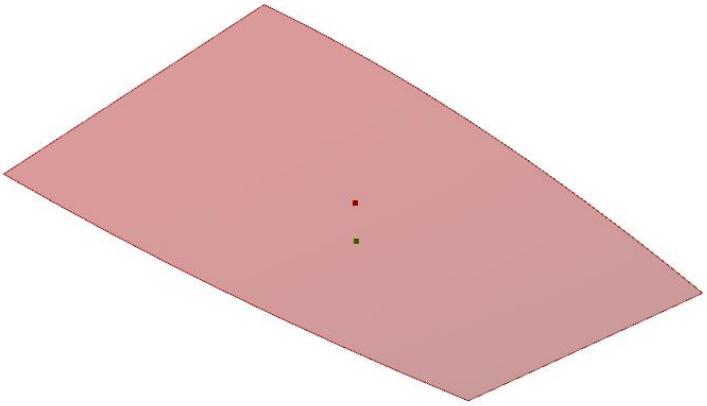


03A – Centroide de generación inicial.

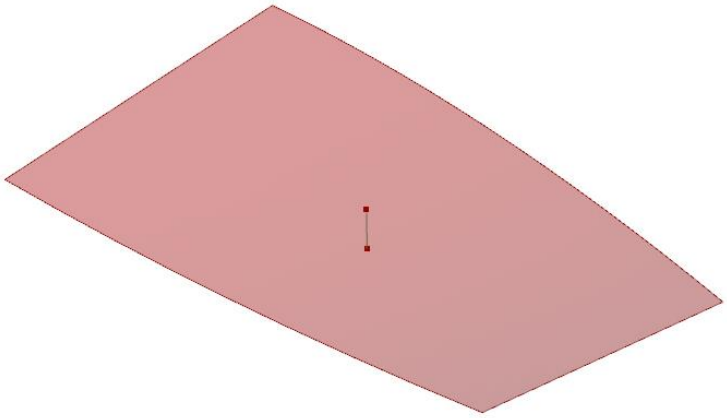
El inicio del sistema de ramificación estará determinado por una primera iteración lineal, que será generada a partir del punto centroide de la superficie. Este punto tendrá que ser desplazado hacia la parte inferior de la superficie para generar la rama inicial, esto implica tener un control vectorial en relación, que pueda generar una línea en dirección al eje “Z”, (figura 89), dónde el sistema inicia a partir de la superficie y no en una rama inicial:



04 – Desplazamiento geométrico y vectorial del punto centroide en el eje Z a través del componente *move* y representación vectorial – lineal por medio del componente *line SDL (Rama Generadora 0)*.



04A – Centroide en desplazamiento -1 m en el eje “Z”.

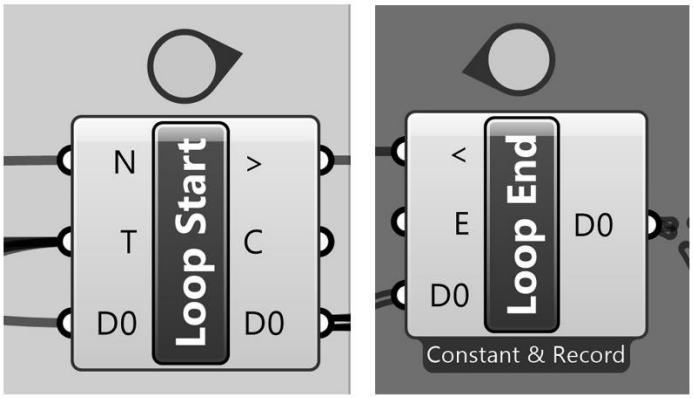


04B – Línea vectorial de generación inicial.

La representación y generación de una rama inicial como parte del sistema, nos permitirá generar un diseño basado en ciclos de ramificación, por lo que es necesario tener una línea con un componente vectorial implícito.

[B] – Ciclo Exponencial Inicial

El componente externo *Anemone*[60] (*plugin*) nos ayudará a generar los ciclos de ramificación controlados basados en geometría y control vectorial. Por lo anterior, tendremos un ciclo inicial y final de generación.

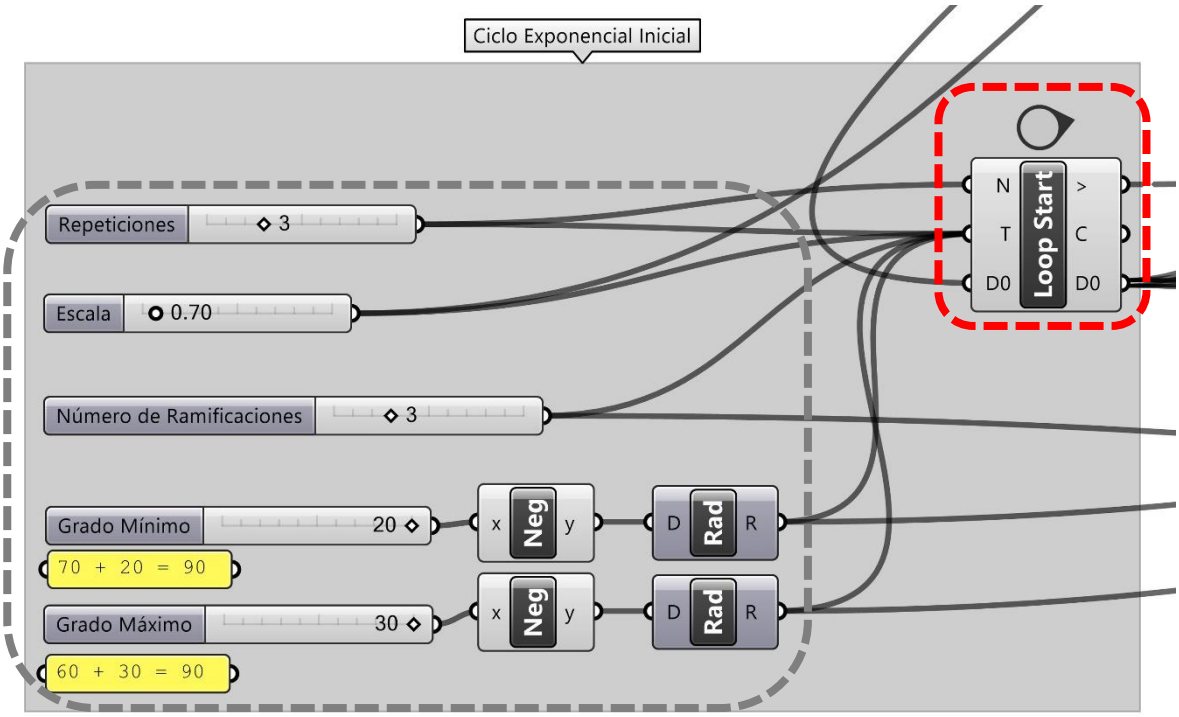


A – Ciclo Generador Inicial **B – Ciclo Generador Final**

Grupo B – Ciclo Exponencial Inicial

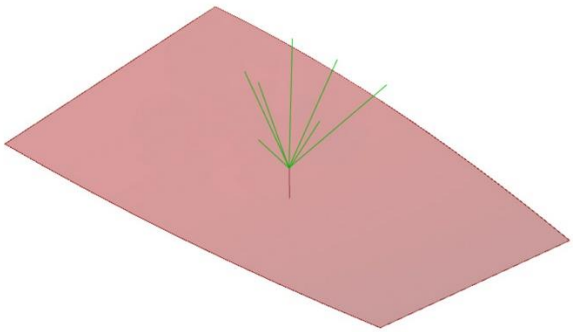
Un ciclo de ramificación desarrollado con este componente, producirá un resultado geométrico – vectorial, que necesitará ser controlado a partir de una serie de parámetros estudiados del organismo. Estos parámetros o límites – variables, determinan los ciclos de generación de manera controlada y en relación directa con la conformación física del espécimen, además de su comportamiento de crecimiento inicial, por ser un ciclo inicial y final que almacena los datos introducidos al inicio del ciclo y los interpreta en el último.

59 El plugin *Anemone* brinda la posibilidad de producir geometría lineal – vectorial basada en ciclos de generación exponencial y progresiva, es decir, por medio de parámetros numéricos se pueden producir sistemas complejos fractales, lineales y de ramificación controlados por una variable, pero con un resultado que contempla muchos elementos pertenecientes a un mismo conjunto dependiente. Este componente es desarrollado actualmente por Mateusz Zwierzycki y se puede encontrar más información en: http://www.grasshopper3d.com/group/anemone.

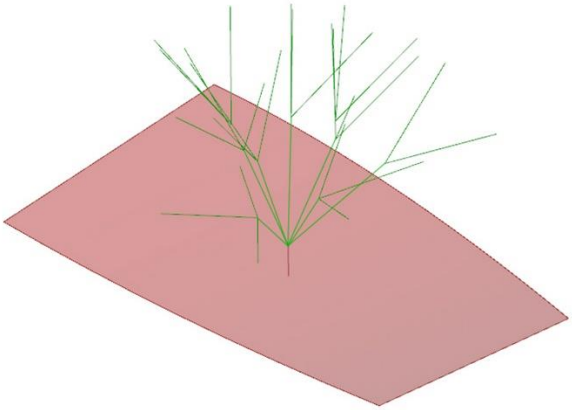


01 – A través del uso del componente *Loop Start* de *Anemone*, definiremos los parámetros límites y en relación al objeto biológico natural real, con el fin de tener un control de nuestro sistema.

A partir de los parámetros numéricos límites (figura 89), un ciclo de generación lineal puede ser llevado a cabo; tales parámetros se refieren al número de repeticiones máximas del organismo en ramificaciones, al número de ramificaciones o iteraciones secundarias lineales, un grado mínimo y máximo de apertura en las ramas, y por último, una relación de escala adecuada para la visualización en el entorno de modelado. Como límites establecidos tenemos la relación lineal – vectorial del grupo A, a través de la conexión D0, que representa los datos a ser generados en ciclos de repeticiones progresivas de acuerdo a los parámetros numéricos establecidos en los *sliders*, su relación y conexión con T, que limitará la propagación del número de repeticiones N, por medio de los grados radianes y el número de ramas a generar en el ciclo final. Con esto obtenemos el primer ciclo de generación.



01A – Iteración 00 3 [N] + 3 [T] = 6R

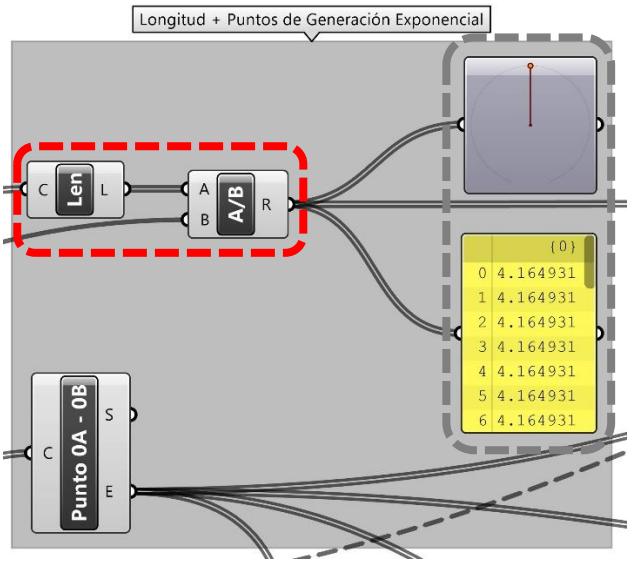


01B – Iteración 01 – 6R * 3R

[C] – Puntos de generación + longitud determinante

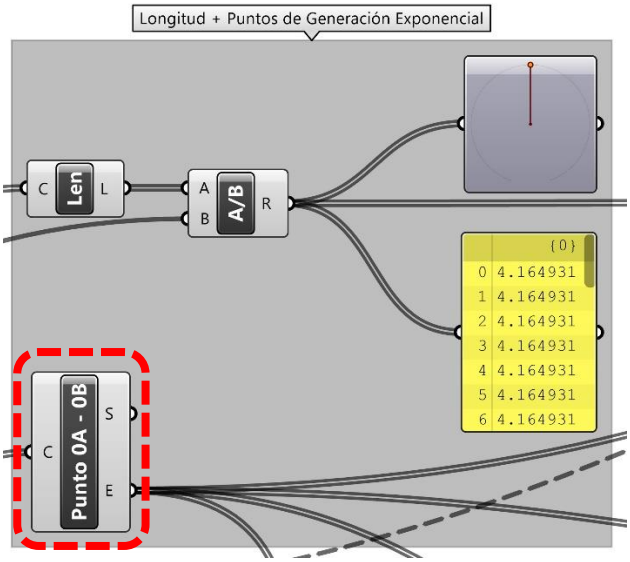
Grupo C – Longitud + Puntos de generación exponencial

Al obtener la primera iteración por medio de la generación cíclica inicial [6R * 3R], la incógnita a resolver es la distancia que existe entre las últimas ramificaciones [3R] en relación a su último punto de origen y al último particular, es decir, necesitamos obtener la distancia en longitud de cada una de ellas.



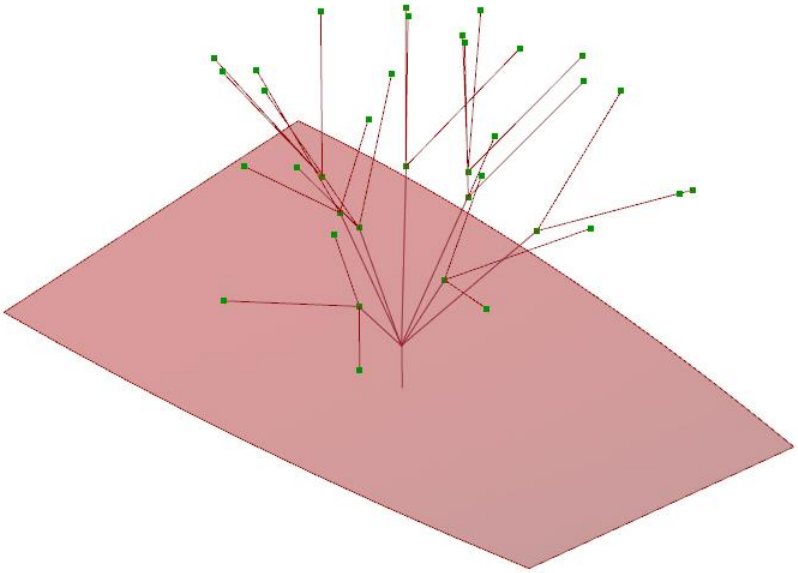
01 – El componente *Length* (*Curve)* y *Division*, en relación directa con el objeto de conocer las longitudes de la iteración R3.

La operación realizada nos brindará el resultado numérico a establecer en las longitudes del ciclo o *Loop* final. Este número *R* será el parámetro de control en el ciclo final de generación de líneas vectoriales que represente el sistema de manera conjunta.



02 – El componente *End Points* relacionado con la iteración *R3* y con el fin de obtener los puntos finales de esta última iteración.

A través de los puntos finales localizados en la última iteración [*R3*], se producirán los puntos medios de generación de círculos, que contemplan el siguiente grupo.



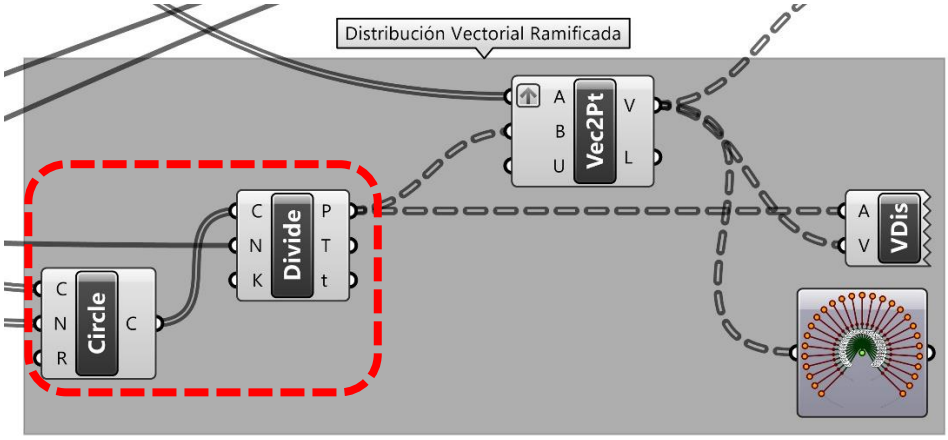
02A – Localización de puntos finales para producir los centroides de círculos.

[D] – Distribución vectorial

La última iteración producida [*R3*] posee un parámetro límite de 3 ramas, en relación al organismo, que se repite en la siguiente iteración (figura 89), por tal motivo, necesitamos localizar los últimos puntos de *R3*, y hacer una relación a través de círculos.

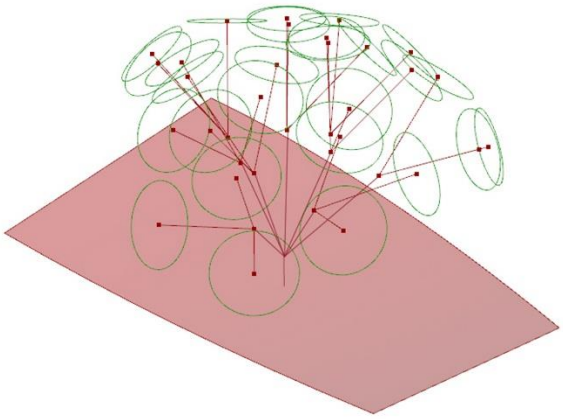
Grupo D – Distribución vectorial ramificada

Al utilizar círculos que inician en los puntos finales de la iteración *R3*, tendremos la oportunidad de establecer un valor mínimo y máximo de división a través de puntos en la circunferencia, que establecerá el límite en cuanto al número de iteración siguiente [*R4*].

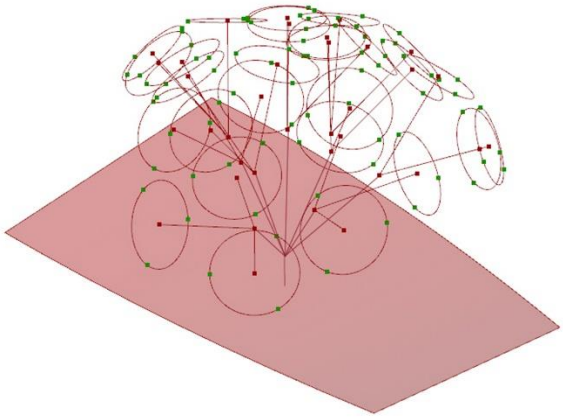


01 – El componente *circle* y *divide* para producir el inicio de la siguiente iteración.

El parámetro numérico mínimo límite es 1 y el máximo límite de la siguiente iteración o ramificación es de 3, por lo que los círculos podrán estar divididos hasta 3 puntos a lo largo de la circunferencia.

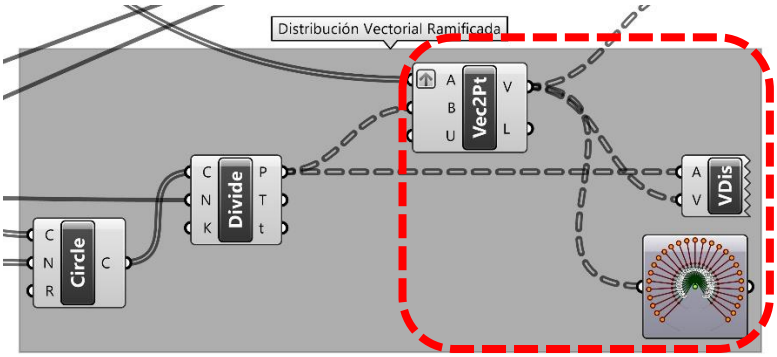


01A – Círculos a partir de puntos finales *R3*



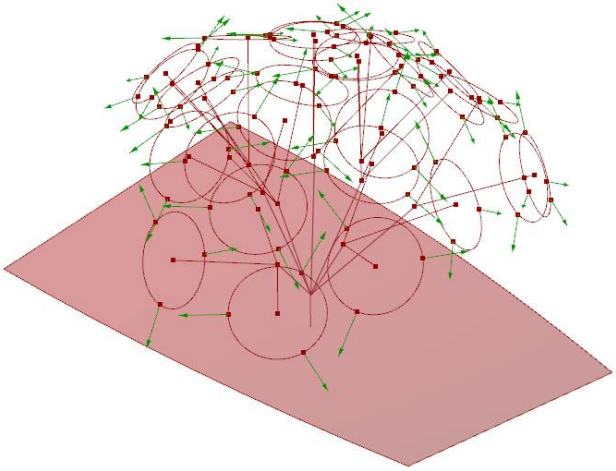
01B – División en 3 puntos = Dirección *R4*

La división de los círculos en tres puntos condicionará la dirección de inicio de la iteración *R4*, es por lo anterior que un vector es necesario, ya que proporcionará la dirección y magnitud de generación de la iteración *R4*.



02 – *Vector 2 Points* para convertir la posición *XY* de los puntos en dirección.

La incorporación de un vector proporcionará una dirección de inicio a través de la división de los círculos de la iteración *R3*, así como una magnitud en relación al grupo *B*.



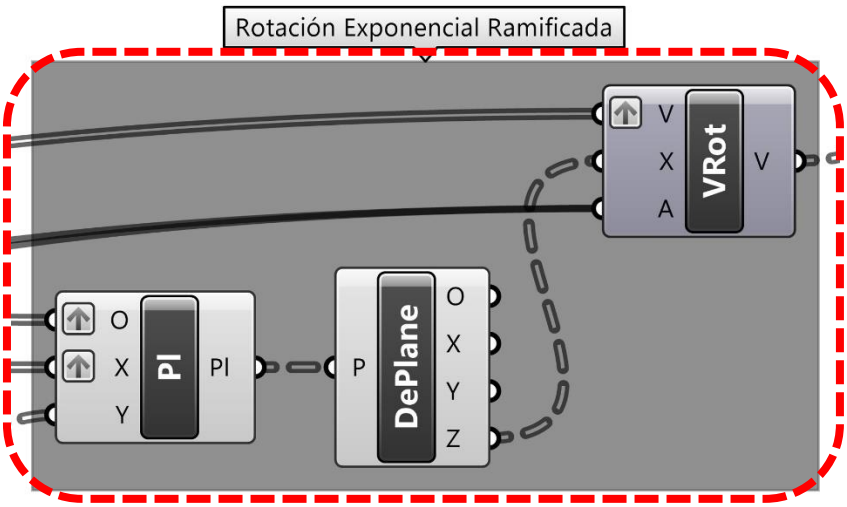
02A – Vectores de dirección generadores de la iteración *R4*.

[E] – Rotaciones unitarias exponenciales

A partir de los puntos finales de la iteración *R3*, y la relación vectorial y de magnitud en el ciclo inicial, necesitamos colocar un plano, a partir del cual generar geometría lineal en base a los ejes que producen cada uno de los vectores iniciales de la iteración *R4*.

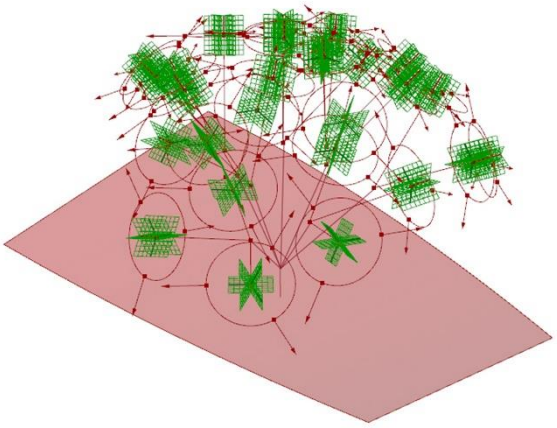
Grupo E – Rotaciones exponenciales ramificadas

Con el fin de colocar un plano en cada uno de los puntos finales *R3* e iniciales *R4*, y generar a partir de ellos líneas, el uso del componente *construct plane* es imprescindible, ya que podemos localizar los puntos base [*O*] correspondientes al primer ciclo de iteración y combinar una magnitud de dirección vectorial de los puntos que dividen a los círculos de la iteración *R*3 referidos al eje *Z* de desplazamiento con una dirección *X* en rotación (componente *Rotate Vector*) a establecer en la iteración *R4*.

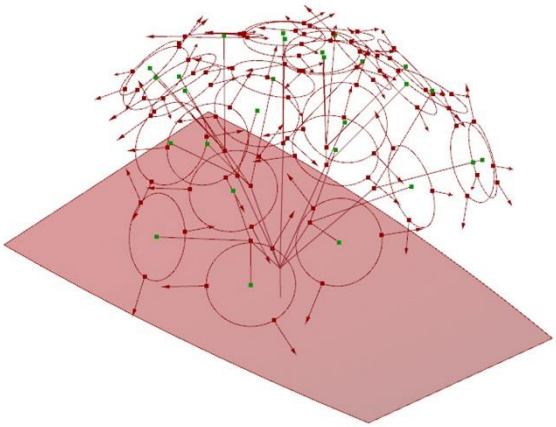


01 – Componentes base de generación geométrica lineal – vectorial a través de *construct plane*, *deconstruct* plane y *rotate vector.*

Con esta combinación de componentes obtendremos un resultado basado en geometría de ciclos, parámetros de división y vectores, que servirán para establecer la dirección vectorial del ciclo final a través de un componente lineal – vectorial principal del ciclo final.



01A – Planos de generación geométricos.



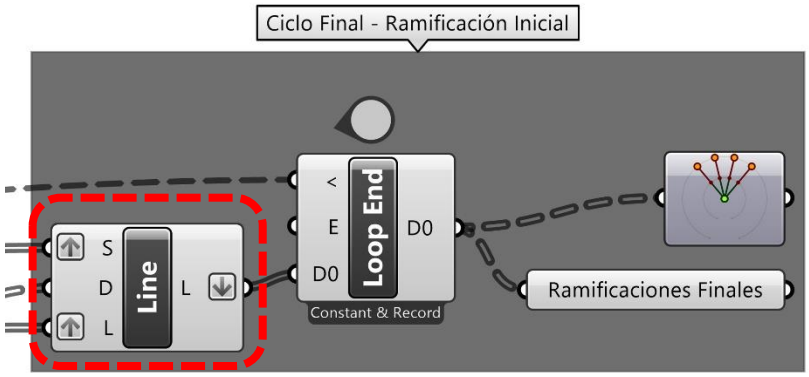
01B – Puntos generadores Geo – Vectoriales.

[F] – Ciclo final vectorial y geométrico lineal

La representación geométrica lineal a partir de los puntos iniciales de la iteración *R4* resulta algo complicada de entender, pero de manera sencilla, hemos convertido los puntos finales de la iteración *R3* no sólo en elementos geométricos, sino que hemos añadido características vectoriales y de magnitud basadas en el primer ciclo de generación, por lo que cualquier cambio en los parámetros iniciales del ciclo inicial, afectará la magnitud y dirección de cada uno de estos puntos.

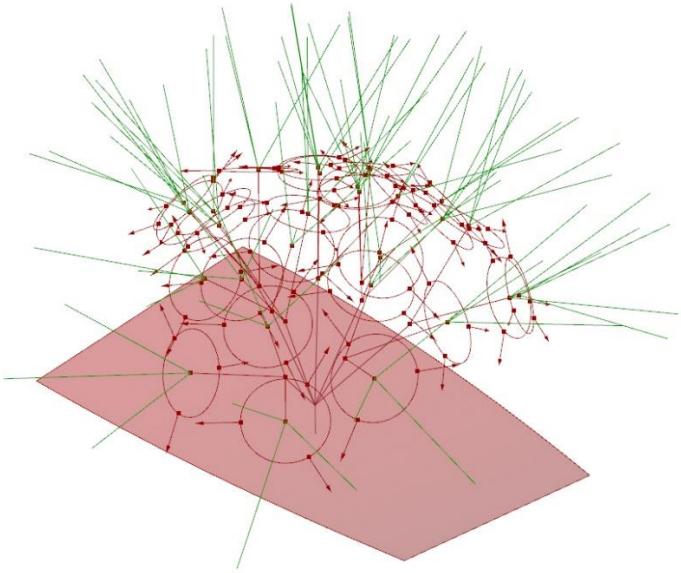
Grupo F – Ciclo final – Ramificación inicial

Hemos convertido entonces, geometría simple a geometría vectorial, para poder generar la siguiente iteración, necesitaremos un componente que nos permita generar las líneas (ramas) de la iteración *R4* en base a vectores, geometría y magnitudes, en este caso la magnitud referida a la longitud y a la división de ramificaciones de las de la iteración *R3*. El componente que nos ayudará a realizar lo anterior es *line SDL*.



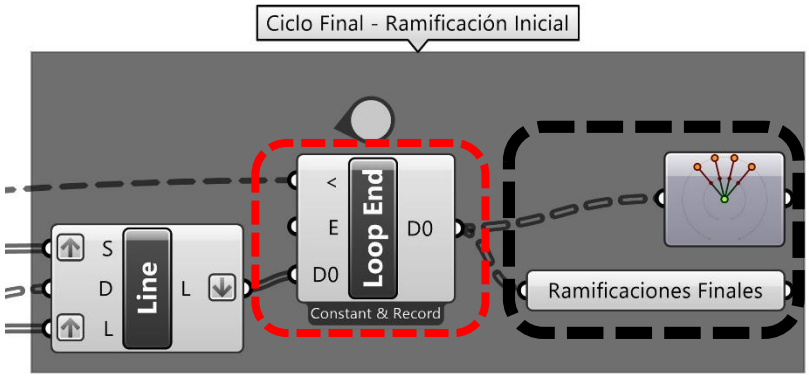
01 – *Line SDL* para interpretar datos y alojarlos en el *Loop End.*

Al representar todos los datos geométricos, de magnitud y vectoriales a través de line SDL, obtendremos la representación particular de la iteración *R4*, pero exenta de los ciclos.



01A – Iteración *R4* independiente del sistema.

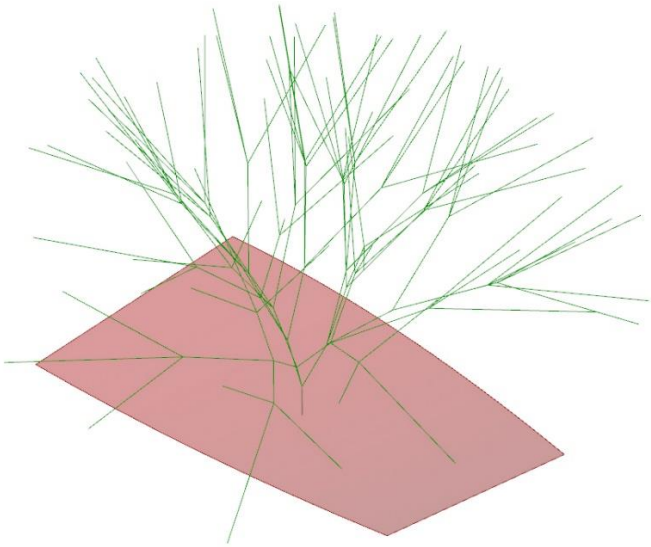
Como lo muestra la anterior imagen, la iteración *R4* ha quedado exenta de la generación cíclica, pero dependiente de ella en parámetros. Para poder llevar a cabo la iteración final de manera cíclica y paramétrica tendremos que explotar las variables de entrada del componente line SDL [*Graft* - representado con un cuadro que contiene una flecha hacía arriba] para, posteriormente condesar esas variables y datos [*Flatten* - representado con un cuadro que contiene una flecha hacía abajo] y alojarlos en el parámetro de entrada del ciclo final [*D0*] para establecer la relación unitaria en conjunto de todo el sistema de generación fractal ramificada.



01 – *Loop End* representando el ciclo completo del sistema.

Al relacionar la salida del parámetro del primer ciclo (***Loop Start >***) con el parámetro de entrada (***Loop End <***), establecemos una relación cíclica directa con los las variables y números iniciales de control de nuestro sistema de generación ramificado, además de contemplar los datos en conjunto del componente *line SDL*, referidos a geometría, vectores y parámetros de modificación 3D. El componente *curve* [Ramificaciones Finales] representará la geometría resultante de tal ciclo de manera unitaria, con el fin de realizar las simulaciones físicas que afectan a este sistema biológico – computacional en conjunto.

Por otra parte, el elemento que tiene ramificaciones dispuestas de manera radial en dos círculos, pertenece al componente *Param Viewer*, y cada uno de los puntos representa a cada una de las ramificaciones del ciclo completo de manera unitaria, es decir, cada punto representa la iteración ***R1 + R2 + R3 + R4***, en un mismo sistema paramétrico.



01A – Sistema Fractal Ramificado Lineal Final.

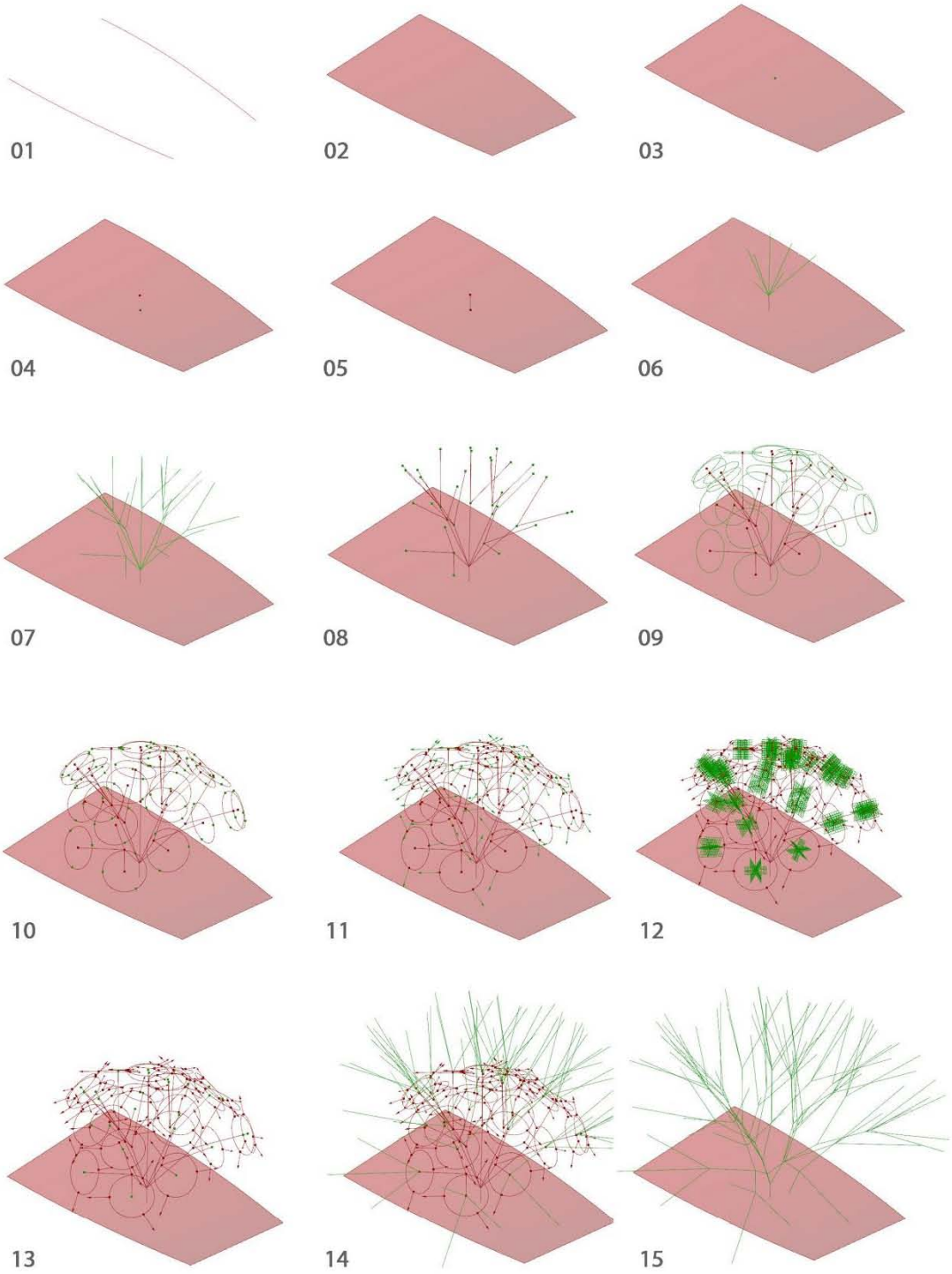


Figura 88. Etapas del proceso de generación cíclica ramificada del sistema geométrico paramétrico.

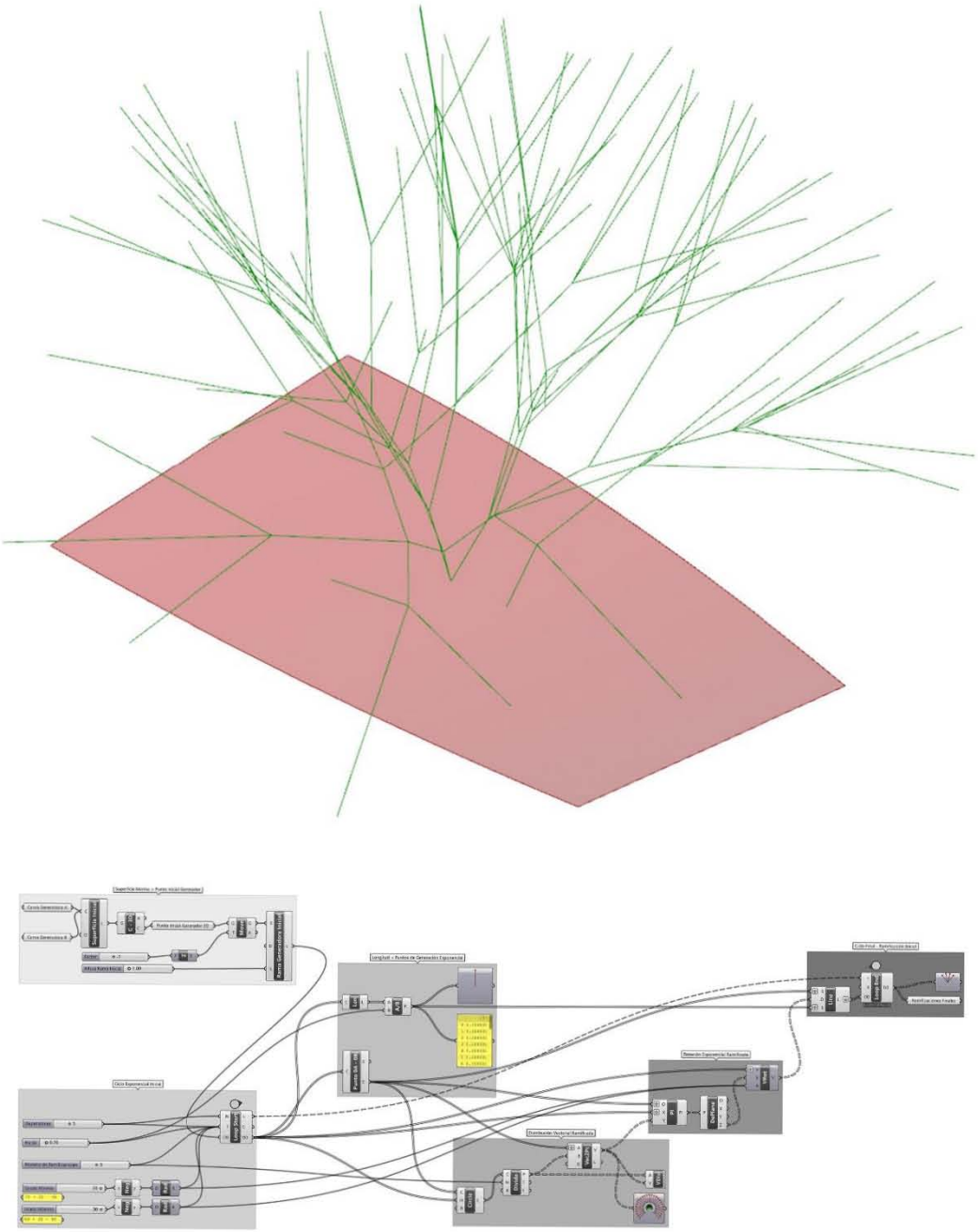


Figura 89. Conformación geométrica lineal final del ciclo fractal de ramificación, basado en un algoritmo de programación gráfica de *Grasshopper* y representativo de la *[1] _ Etapa – Enfoque: [1 – 1] y [2 – 2].*

4.8.2 – [2] _ Etapa – Enfoque: [2 – 3] y [3 – 3]

En esta etapa del proceso de diseño se desarrollan conceptos relacionados con los enfoques descritos al inicio de esta metodología. Estos enfoques incluyen la relación de la física, las matemáticas, la programación, la representación de resultados y modificaciones a través de la geometría, en conjunto con la simulación de fuerzas, como la presión del agua y la velocidad de las corrientes marinas que afectarán a nuestro prototipo artificial por medio del uso de plugin *Kangaroo* para *Grasshopper*.

En el anterior enfoque de diseño aplicado a través de la programación logarítmica gráfica, desarrollamos el ciclo de ramificaciones fractales iniciales y finales, el resultado de tal proceso nos proporcionó la representación de este ciclo de datos y parámetros mediante líneas que representan a cada uno de estos ciclos, y por consiguiente, cada grupo de ramificaciones. Esto es representado a través de un “árbol de datos”, dónde cada iteración de “ramas” representa un ciclo de generación basado en la analogía del organismo natural.

01 – El componente *Param Viewer* y las 4 ramificaciones finales de ciclo: 1R + 2R + 3R + 4R.

Con lo anterior, hemos logrado hacer un conjunto unitario, que se divide en 4 partes o ramas. Para poder llevar a cabo una simulación física es necesario acceder a cada una de las ramificaciones y descomponerlas en curvas y puntos, con el objetivo de aplicar fuerzas a cada uno de los elementos, reunir el comportamiento y volver a realizar un conjunto para poder representar el resultado de tal simulación en un objeto descrito por la geometría.

Es por eso que hemos utilizado el componente *curve* (Ramificaciones Finales), con el fin de enviar todos los datos del *Loop End* (ciclo final) a un componente que representará a esos datos en geometría y que nos permitirá desarrollar la siguiente etapa. En este caso, vamos a retirar el componente “Ramificaciones Finales” del grupo referido al ciclo final.

01A – Componente *curve* (Ramificaciones Finales) excluido del grupo de generación cíclica.

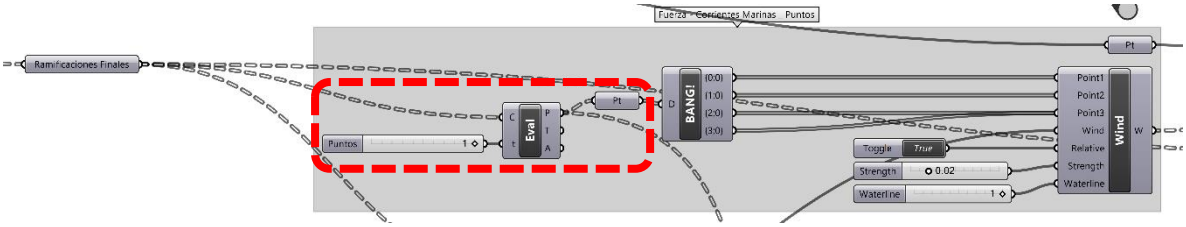
Cabe señalar que, así como hemos empleado grupos para explicar cada proceso de diseño y su respectiva representación, en este apartado también utilizaremos grupos para explicar el proceso de diseño referido a la simulación física y matemática.

[A] – Aplicación de fuerzas a elementos unitarios del ciclo final

Como se ha comentado anteriormente, el primer paso para poder aplicar fuerzas a nuestro ciclo de ramificaciones es descomponer esta geometría en sus partes elementales, es decir, necesitamos obtener cada uno de los puntos de inicio y final de las ramas, así como las líneas. Esto nos permitirá aplicar las fuerzas referidas a la presión del agua y a las corrientes marinas que afectan al organismo, para representar al final de la simulación, nuevamente la geometría unitaria en conjunto.

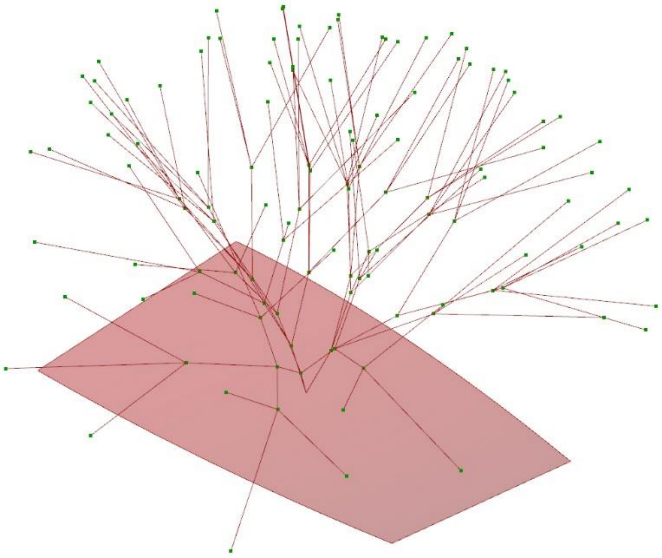
Grupo A – Fuerza - Corrientes Marinas _ Puntos

Con el fin de obtener y aplicar fuerzas a cada uno de los componentes de este sistema geométrico, vamos a descomponer todo este sistema. El primer paso es dividir los elementos en los puntos, los cuales nos servirán para aplicar las fuerzas de las corrientes marinas a través del componente *wind* de *Kangaroo*. Es de resaltar la importancia que tiene este componente como generador del primer ciclo de modificación o cambio, ya que actuará sobre cada uno de los puntos que generar las líneas de la ramificación secundaria, a la que llamaremos R [B].



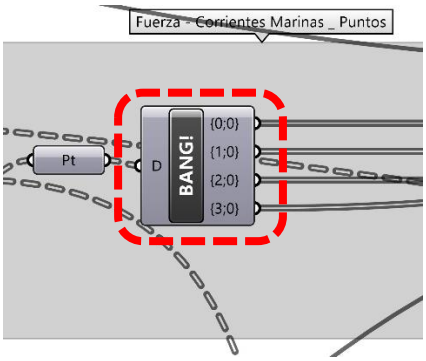
01 – El componente *Evaluate Curve* y *point* para representar los puntos en conjunto.

En este caso, el componente *Evaluate Curve*, nos permitirá hacer una selección unitaria de todos los puntos que se encuentran conformando a la totalidad del sistema de ramificación, el parámetro 1 en unión con *t* determina que la selección es de todos los puntos. Con esto obtenemos todos los puntos que conforman al sistema de manera unitaria.



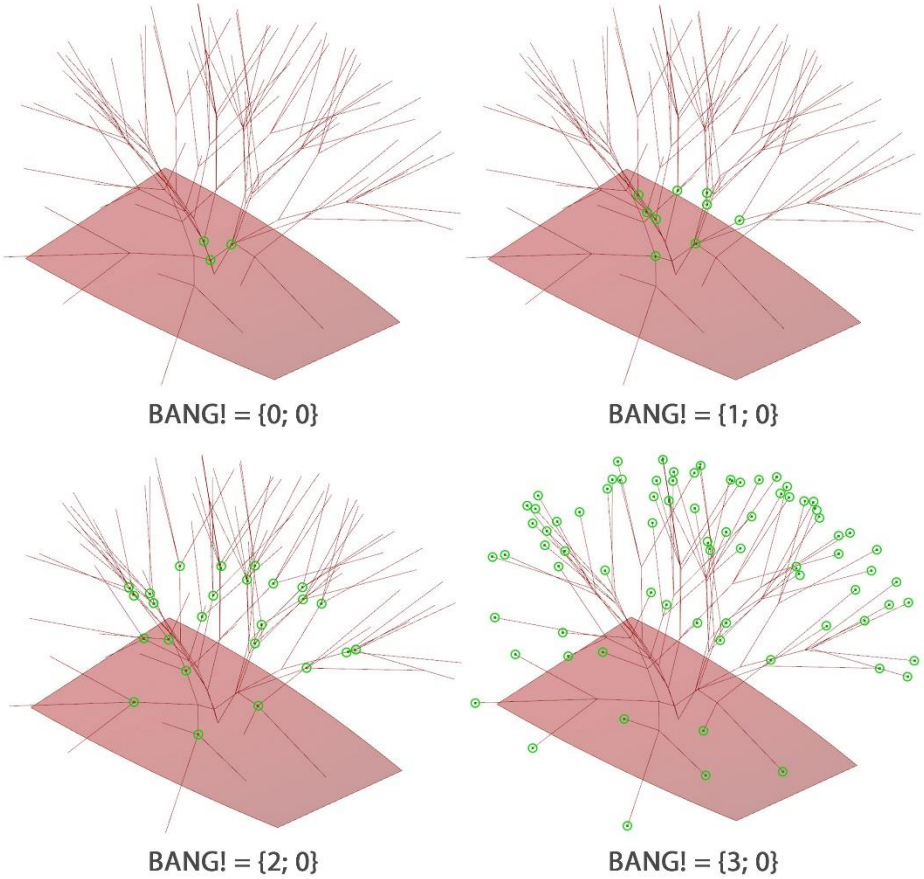
01A – Selección de todos los puntos que dan lugar al sistema de manera unitaria.

Al tratarse de un sistema unitario, es necesario acceder a cada uno de los ciclos de generación (*R1 + R2 + R3 + R4*) representados a través de los puntos. Esto nos ayudará a definir cuales puntos son menos y más afectados por las fuerzas a aplicar en cada ciclo, tal y como sucede en el espécimen natural, ya que las ramas superiores (más jóvenes) son afectadas en mayor proporción debido a la relación directa con el entorno (agua). Para realizar lo anterior, utilizaremos en componente *Explode Tree* (*BANG!*), el cual nos permitirá acceder a cada sistema de puntos, definido por una correspondiente ramificación ({A; B}), dónde *A* representa el ciclo de ramificación y *B* el sistema completo geométrico.



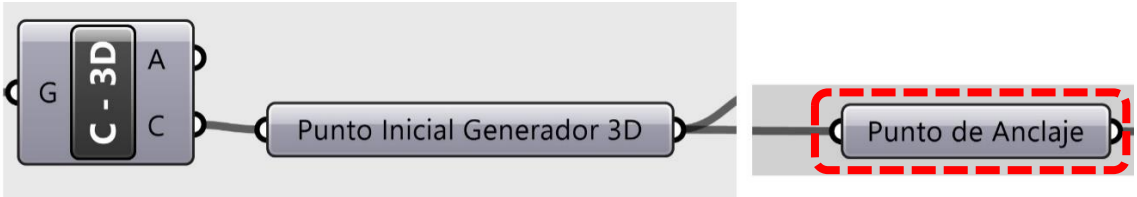
02 – El componente *BANG!* para obtener cada iteración de generación a través de un sistema de puntos.

Al obtener cada iteración de generación, los puntos son divididos en grupos que dependen de cada ciclo de generación, lo que permitirá afectar a cada sistema mediante una fuerza.



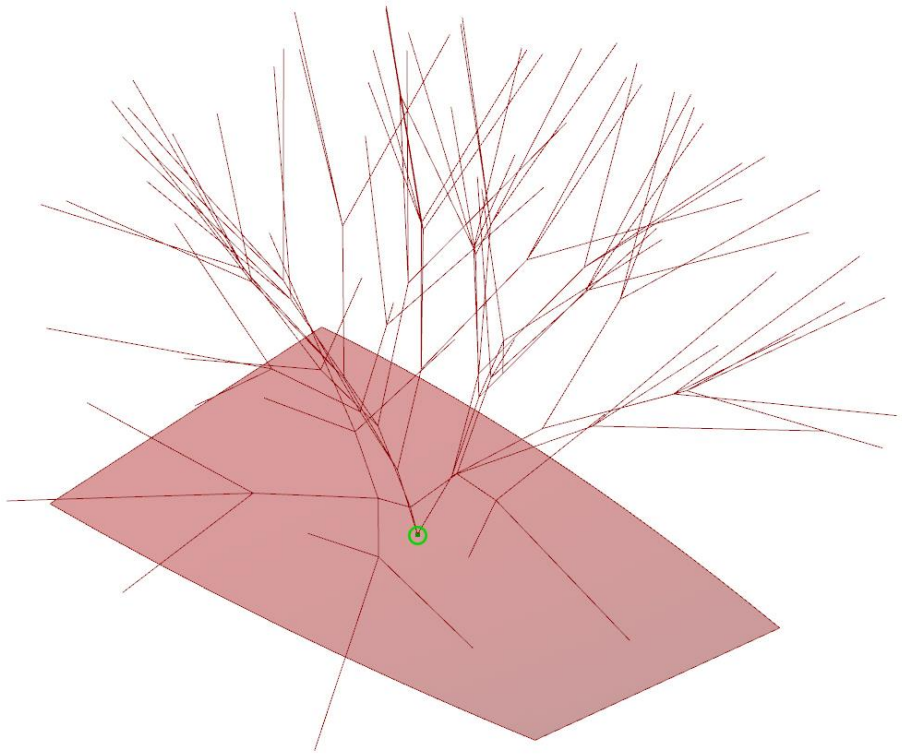
02A – Selección de puntos por iteración generativa en un sistema de ramificación fractal.

Cuando una magnitud de fuerza es aplicada a cada miembro de un sistema es necesario tener un punto de apoyo. En este caso lo vamos a nombrar como punto de anclaje (*Anchor Point*), el cual es un elemento necesario para poder aplicar una fuerza a un sistema, de lo contrario, todo el sistema se desplazaría a causa de la magnitud aplicada. Por lo anterior, necesitamos incorporar el punto centroide generador inicial como punto de anclaje, ya que el sistema digital se genera a partir de él.



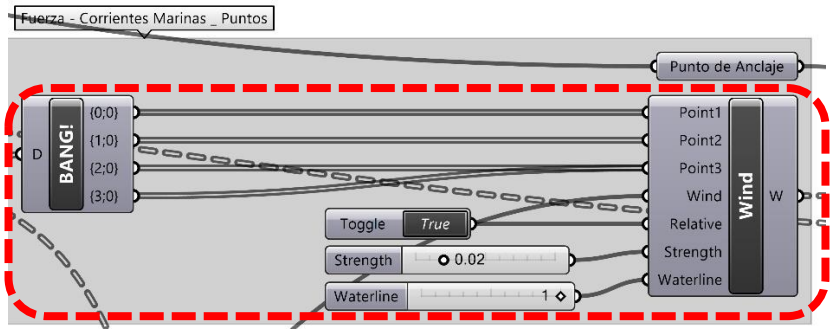
03 – Relación directa de Punto Inicial Generador con Punto de Anclaje para simulación de fuerzas

Al haber realizado la relación directa del punto generador con el punto de anclaje necesario para la simulación de fuerzas, hemos determinado de manera paramétrica tanto el inicio de la generación del sistema, como su modificación geométrica a causa de la aplicación de fuerzas que dependen del componente inicial de generación (Punto Inicial Generador 3D), que será soporte principal de los componentes secundarios generados y afectados.



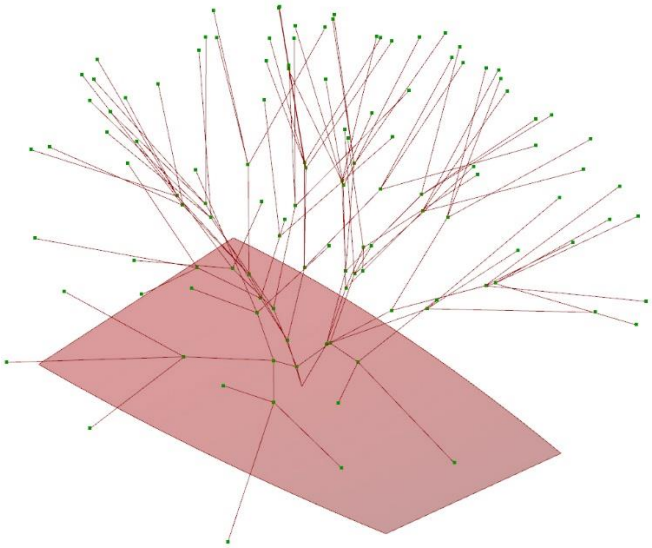
03A – Punto de Anclaje necesario para llevar a cabo la simulación de fuerzas en todo el sistema.

Los puntos necesarios a los cuales aplicar la fuerza (*wind*), están determinados por las iteraciones según cada ciclo de generación, esto quiere decir que la iteración menor *{0; 0}* representa a los miembros con más edad del sistema natural – computacional, y la iteración *{3; 0}* a los miembros más jóvenes. La simulación de la fuerza de las corrientes marinas se llevará a cabo con el componente *Wind* de *Grasshopper*, a partir de una relación con los puntos de iteraciones obtenidos, que representan los elementos a afectar.



04 – Relación de puntos de iteración con entradas de puntos a afectar mediante la fuerza.

Al tener este tipo de relación, la iteración de puntos *{0; 0}* es enlazada con la entrada *Point1*, esto significa que los miembros de este conjunto serán afectados con una menor magnitud de fuerza; en la iteración *{1; 0}*, estos miembros serán afectados por una magnitud de fuerza mayor que la anterior iteración y, las iteraciones *{2; 0}* y *{3; 0}* serán las más afectadas por las fuerzas aplicadas, dado que son los miembros más jóvenes de todo el sistema, capaces de resistir más que los de mayor edad, como sucede en la naturaleza.



04A – Miembros del sistema completos a ser afectados por la fuerza de las corrientes marinas.

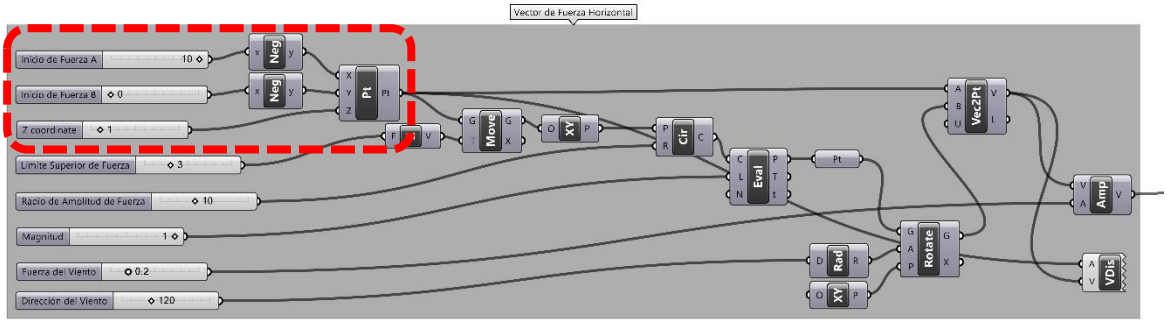
Por otra parte, el componente Wind nos solicita un parámetro referido a *Relative*, en este caso el valor booleano es verdadero (*True*), ya que la fuerza a aplicar varía en cada iteración. El parámetro de entrada referido a *Strength* (intensidad) es de 0.02, que representa una magnitud adecuada para un sistema orgánico que se encuentra debajo del mar, referido en el parámetro de entrada *Waterline*, en el cual, el número 1 representa que es un sistema bajo la influencia del agua (mar) y el 0 que no hay influencia de este elemento. El parámetro de entrada necesario para llevar a cabo la simulación, referido a *Wind* (fuerza vectorial) será explicado en el siguiente grupo.

[B] – Generación de fuerza vectorial de viento – corrientes marinas

En relación al componente de la fuerza del viento (corrientes marinas), es necesario considerar un tensor vectorial, el cual contemple una magnitud y una dirección variable. Para ello, es importante trabajar con este componente como un elemento ajeno al sistema a modificar, pero que sea el parámetro que afectará y modificará la posición y configuración del sistema final a través de la fuerza del viento, es decir, será su controlador.

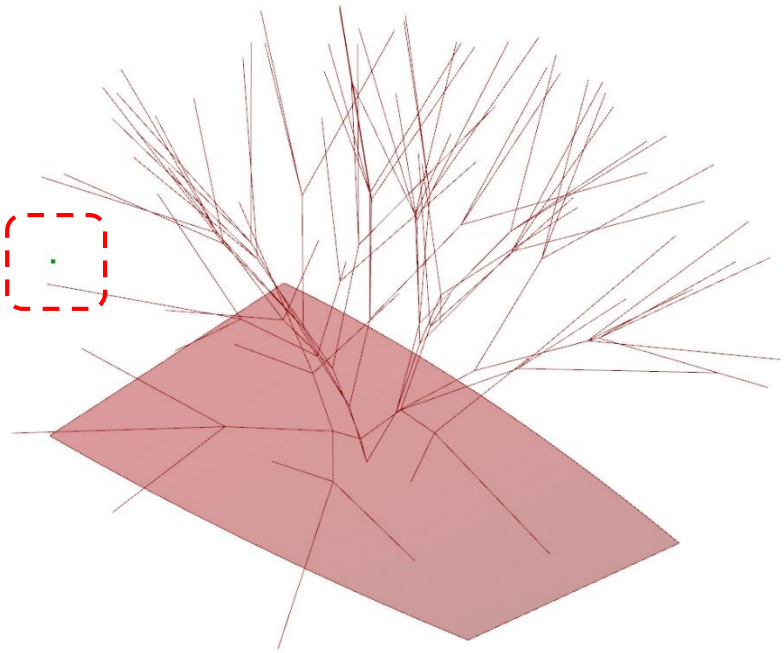
Grupo B – Vector de Fuerza Horizontal

Para poder determinar un vector de modificación, empezaremos con la incorporación de un punto ajeno y exterior al sistema



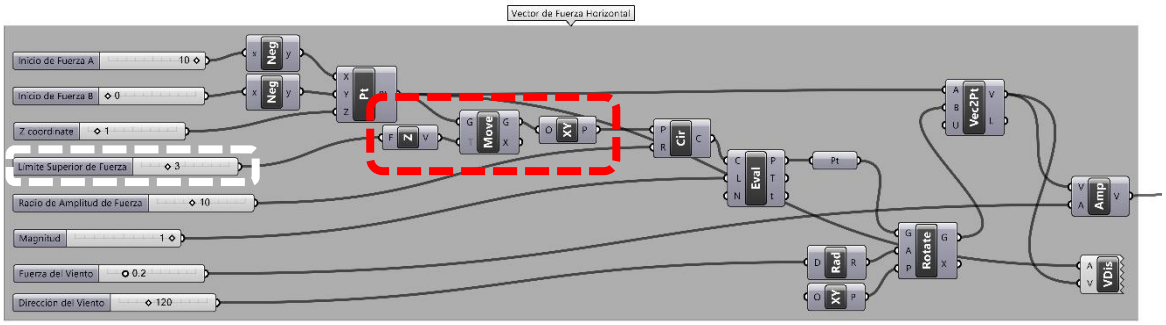
01 – El componente *Construct Point* (Pt), elemento inicial generador del sistema vectorial unitario.

El punto generador deberá estar ajeno al sistema de ramificación fractal inicial, es por ello que se encuentra referido en los parámetros de posición (*sliders*) a -10 unidades en el eje *X*, en relación al eje *Y* - 0 con el fin de aplicar la fuerza de manera lineal y a 1 unidad en el eje *Z*, ya que no puede estar a 0 unidades, porque afectaría de manera axial a la superficie inicial generadora y no al sistema. Este punto será el inicio del vector generador de fuerza.



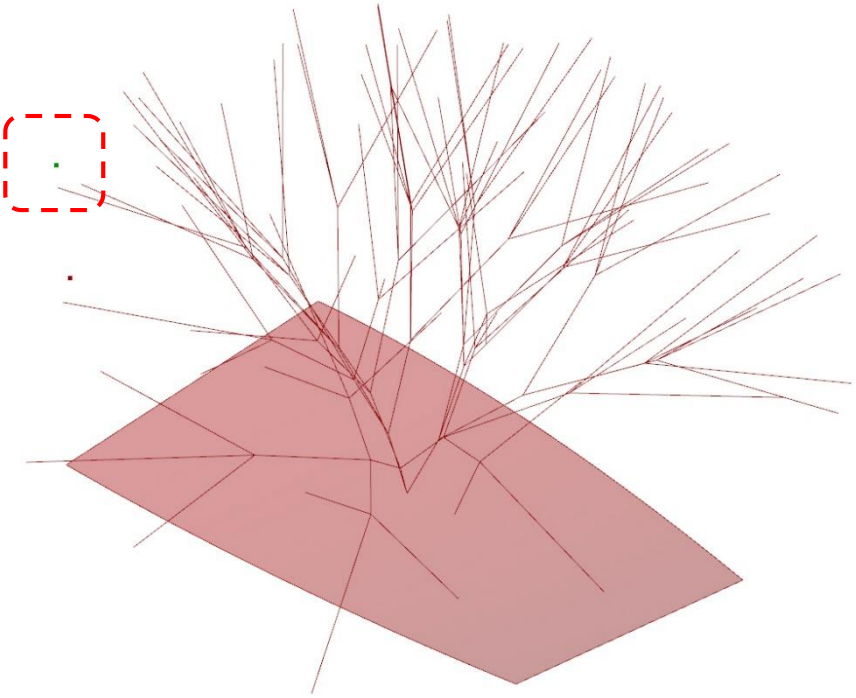
01A – Punto generador vectorial externo al sistema de ramificación.

Al tener el punto inicial para incorporar un vector de fuerza, la referencia en el eje *Z* nos dará la altura a partir de la cual esta fuerza afectará al sistema. Necesitamos incorporar un nuevo punto referido a la altura, el cual limite la aplicación de la fuerza según la altura del organismo digital, además de definir un plano de trabajo que defina la dirección de la interacción del organismo con la fuerza aplicada por medio de un elemento radial.



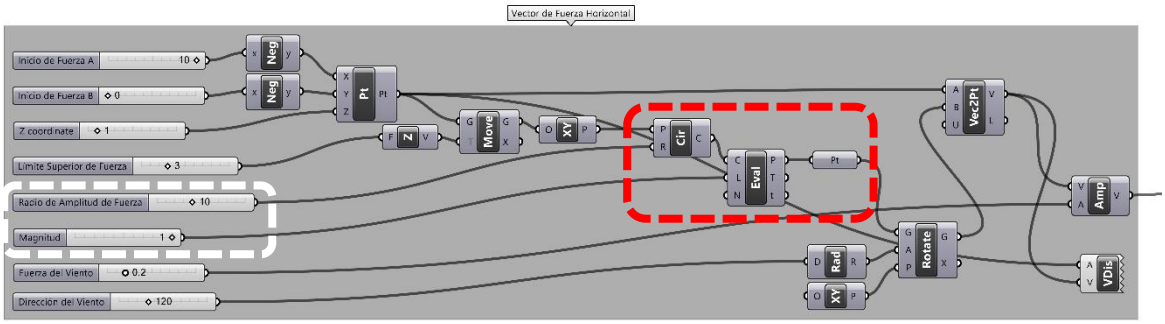
02 – El componente *move, vector Z* y *XY plane*, límites en altura de la fuerza a aplicar en el sistema.

La incorporación y relación del componente *move* (movimiento) con el punto de generación vectorial inicial nos permitirá limitar la amplitud del vector referido a la altura que establece la magnitud de *3* del componente vector *Z* (altura máxima que puede alcanzar un sistema de corales de muchos años), y establecer un plano de trabajo en los ejes *XY* para la adición de un elemento geométrico que defina una fuerza y una dirección de manera radial.



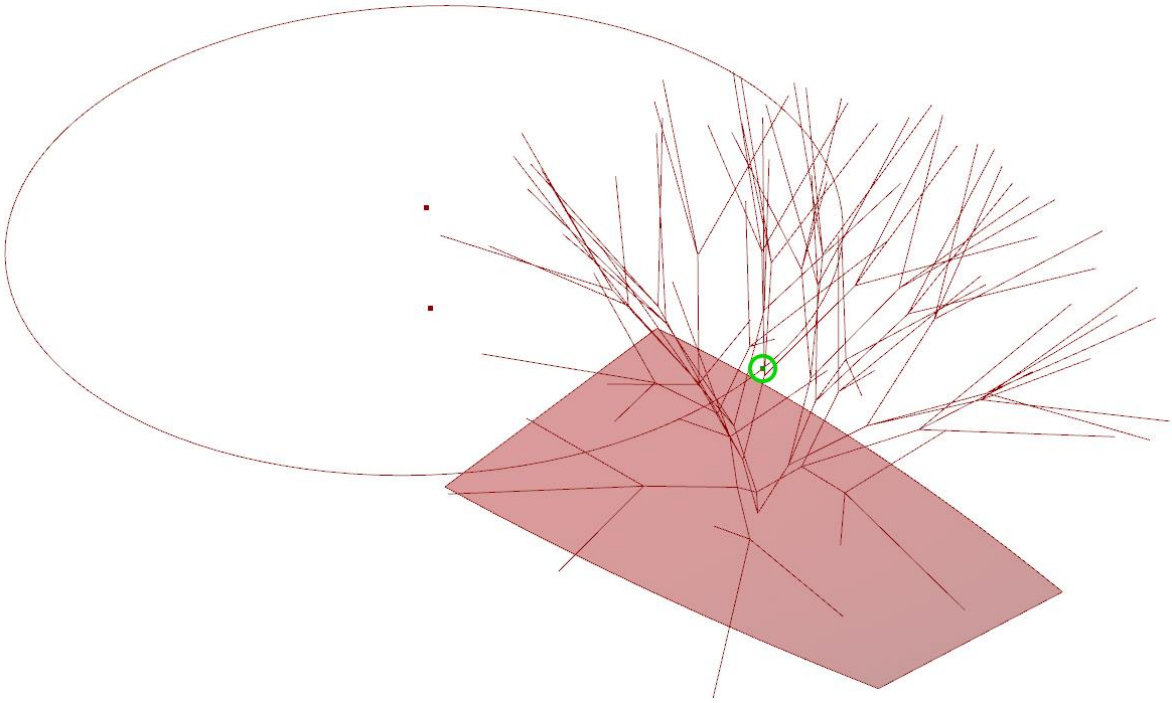
02A – Punto límite de magnitud referido a la altura máxima del sistema.

Para definir un elemento de control de dirección y magnitud del vector de fuerza a incorporar, el uso del componente *circle* es imprescindible, ya que podemos, en base a un radio de amplitud y un punto referido a su circunferencia, relacionar el punto de inicio y el punto final con magnitud y dirección en base a un control de rotación. Este proceso nos servirá para que la fuerza de las corrientes marinas afecte al sistema en cualquier dirección.



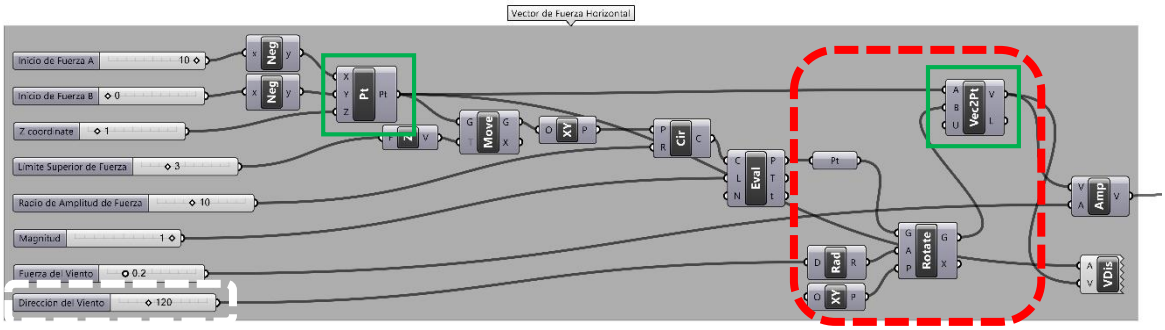
03 – El componente *circle, Evaluate curve* y *point*, para establecer una dirección al vector de fuerza.

El uso del componente *circle* definirá la dirección del vector de fuerza, en base a un comportamiento radial. Por otra parte, el componente *Evaluate curve* definirá el punto de dirección final del vector, tomando como referencia la división en un punto (componente *point*) que interactúa con la totalidad de la circunferencia mediante la asignación del valor numérico *1*. A partir de este proceso de asignación de dirección a un tensor vectorial, seremos capaces de variar la relación del punto inicial con el final, únicamente mediante la adición de un componente de rotación, que nos permitirá definir una dirección y magnitud de fuerza a aplicar en el sistema algorítmico computacional – natural.



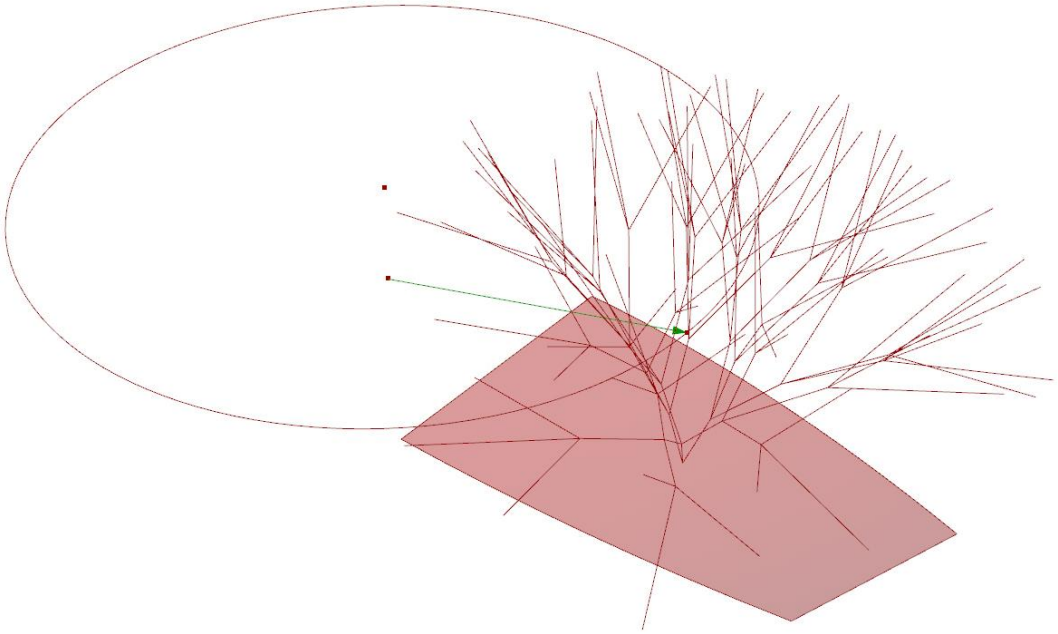
03A – Punto vectorial final que define la magnitud y dirección de la fuerza a aplicar (*wind*).

Una vez generado el punto inicial (fuerza) y el punto final (dirección), necesitamos añadir un componente relacionado con la rotación en un plano horizontal, esto quiere decir que necesitamos un componente que incluya la fuerza y posición del punto inicial y la dirección del punto final. El componente *rotate* (rotación) es el indicado para controlar la dirección del punto final y el punto a partir del cual va a rotar es el inicial.



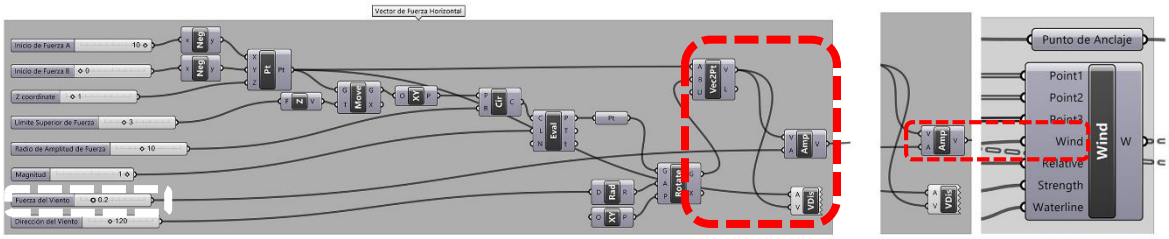
04 – En verde los puntos generadores vectoriales controlados a través del componente *rotate*.

Al incorporar el componente *rotate*, necesitamos referir la geometría (entrada *G*) que vamos a rotar, en este caso, el punto final generado a partir del componente *Evaluate curve* y representado por *point*. El uso de un componente que nos permita definir grados se interpreta a través de un valor de rotación radial. En este caso, *120 Radianes* equivalen a 360°, es por eso que el *slider* con el valor *120* se relaciona con *Rad* y con *A*, que representa el ángulo de rotación. Al tener un comportamiento de rotación, representado por la circunferencia, es de considerar su plano de origen, es por eso que definimos el componente *XY plane* como plano de rotación, ya que es el mismo plano a partir del cual generamos la circunferencia de control. Una vez obtenidos todos los componentes y controles de modificación geométrica, necesitamos representar al vector resultante. Esto se obtiene a partir de la incorporación del componente *vector from two points*; el punto inicial representa el inicio de la fuerza y el punto final, la dirección y en conjunto. Al unir ambos puntos con las entradas de datos de componente *A – B*, hemos representado un vector que proporcionará la fuerza de las mareas del agua (mar) afectando al sistema en conjunto.



04A – Vector resultante que representa la fuerza de las corrientes marinas a aplicar en el sistema.

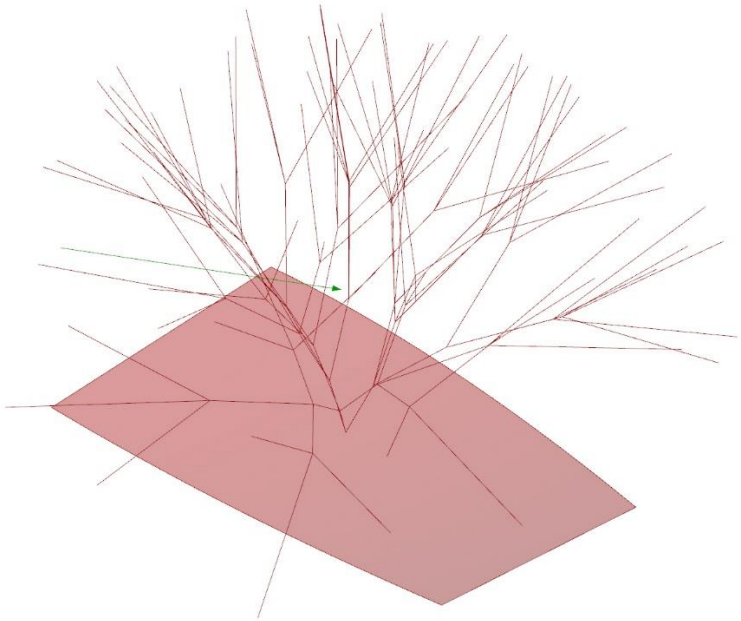
Al obtener una representación vectorial de la fuerza y dirección de las corrientes marinas que afectarán y modificarán el sistema de ramificación, existe un problema referido a los datos que interpretará y calculará el componente *Kangaroo - Wind*. Por lo anterior, necesitamos *"traducir"* datos vectoriales a cantidades referidas a magnitud y dirección de la fuerza. El componente *amplitude* nos ayudará a llevar a cabo tal tarea.



05 – El componente *amplitude* (vector – cantidad) y la relación con el componente *wind* de fuerza.

Al relacionar los datos de salida vectoriales del componente *vector2Pt* con el parámetro de entrada *V* del componente *amplitude*, se lleva a cabo una interpretación de parámetros vectoriales a numéricos que representan una magnitud de fuerza física, como tal, estos parámetros necesitan tener un valor de amplitud, referido a la entrada *A* del componente y que representará la magnitud de la fuerza final que genera la definición de control vectorial.

Una vez obtenida esta magnitud física de fuerzas, el resultado (salida *V* de *amplitude*) se relaciona con el parámetro de entrada *Wind* del componente del mismo nombre, estableciendo una relación geométrica (puntos afectados del sistema de ramificación), una relación de control, magnitud - dirección vectorial, y una relación referida a una fuerza controlada por un valor numérico de amplitud. Con esto y a partir de los parámetros geométricos, numéricos y booleanos, hemos definido la fuerza de las corrientes marinas que afectarán al sistema computacional - natural en conjunto. Este aspecto es importante, ya que este componente (*wind*) será solo una fuerza que deberá ser simulada por medio de un componente que traduzca los parámetros de control en una simulación computacional – física que afecte al sistema geométrico de ramificaciones.



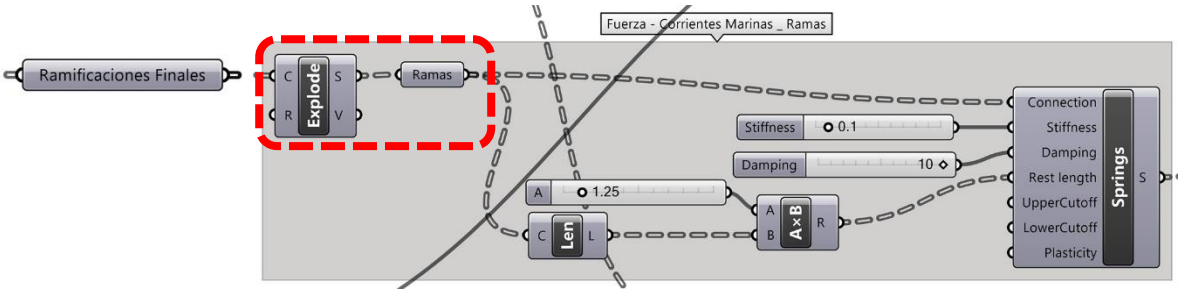
05A – Fuerza resultante a aplicar en las iteraciones del sistema computacional ramificado.

[C] – Generación de fuerza lineal – elástica en ramificaciones.

En el anterior grupo definimos la fuerza de las corrientes marinas a través de *Kangaroo-Wind* en los puntos de cada una de las iteraciones de generación en las ramificaciones. El siguiente paso es definir la misma fuerza, pero en las ramas, es decir, en las líneas que conforman a este sistema. Para eso, deberemos trabajar únicamente con las líneas y excluir los puntos.

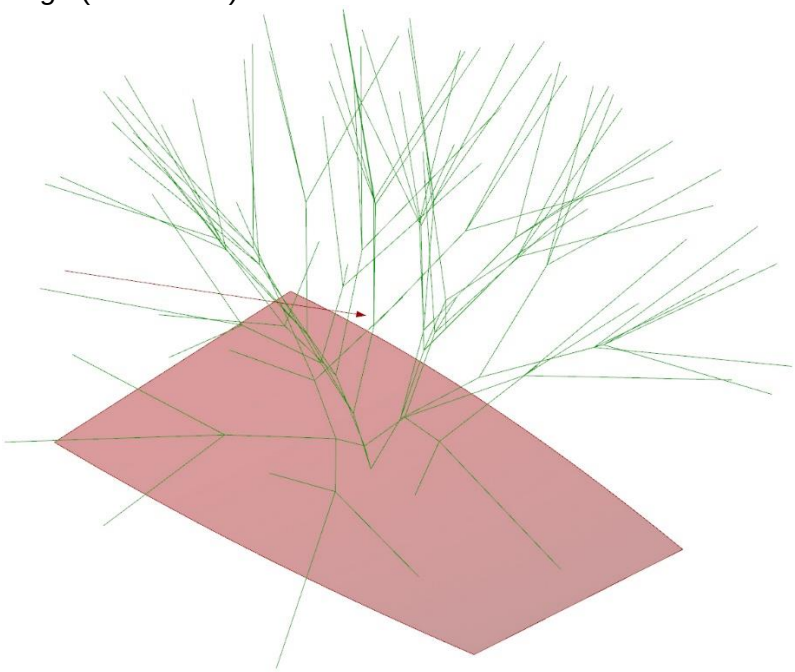
Grupo C – Fuerza - Corrientes Marinas _ Ramas

Es importante considerar que, al trabajar con líneas, este sistema está conformado por puntos que dan lugar a las líneas. En este caso necesitaremos utilizar un componente que nos ayude a excluir líneas y puntos denominado *Explode curve* (explotar curva).



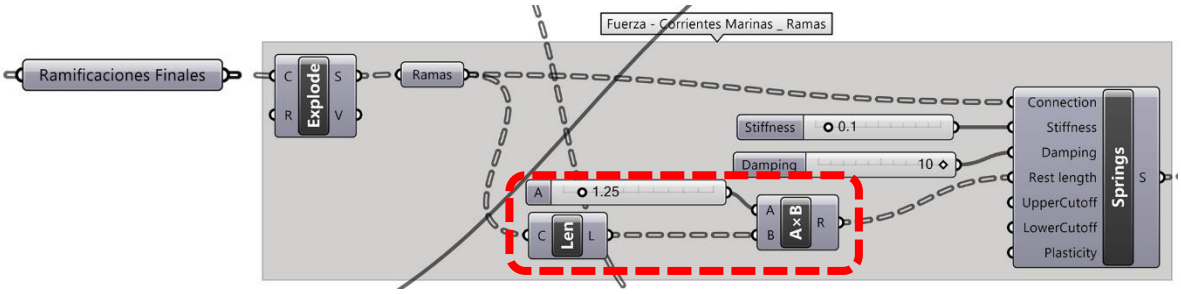
01 – El componente *Explode curve* + *curve,* para excluir líneas y puntos del sistema de ramificación.

Al utilizar el componente *Explode curve* a partir de las ramificaciones del ciclo de generación final (Ramificaciones Finales), seremos capaces de dividir el sistema geométrico en vértices [*V*] (puntos) y segmentos [*S*] (líneas) de este organismo digital. En esta ocasión, añadiremos el componente *curve* para unir únicamente todas las líneas de este sistema, al cual, posteriormente aplicaremos la fuerza de corrientes marinas, a través del componente *Kangaroo – Springs* (elasticidad).



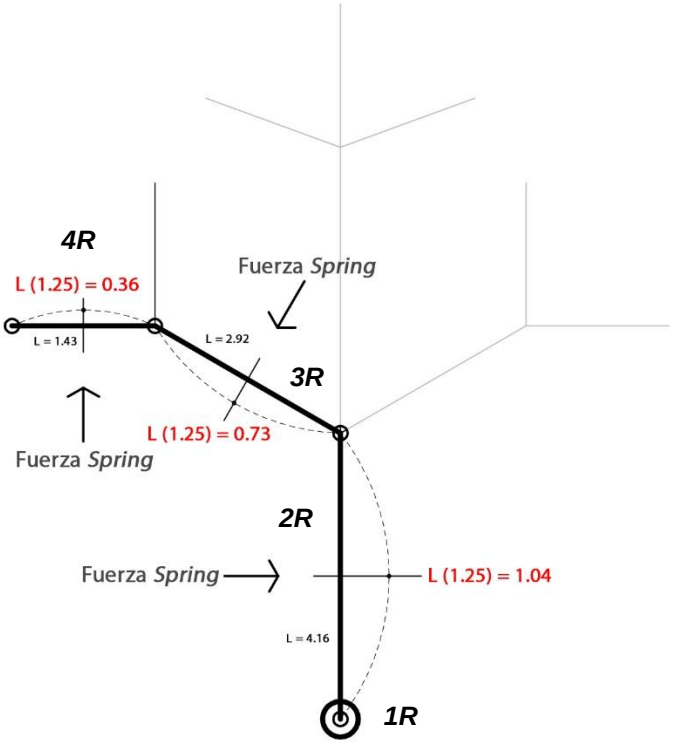
01A – Exclusión de puntos; sistema conformado únicamente por líneas, elementos a aplicar la fuerza.

Al haber realizado la exclusión de los puntos y dejar únicamente las líneas, necesitamos conocer la dimensión de cada uno de los ciclos de ramificaciones con el fin de establecer un límite máximo de fuerza de elasticidad a aplicar mediante un parámetro de multiplicación matemático y un número que permita asignar la magnitud límite correspondiente.



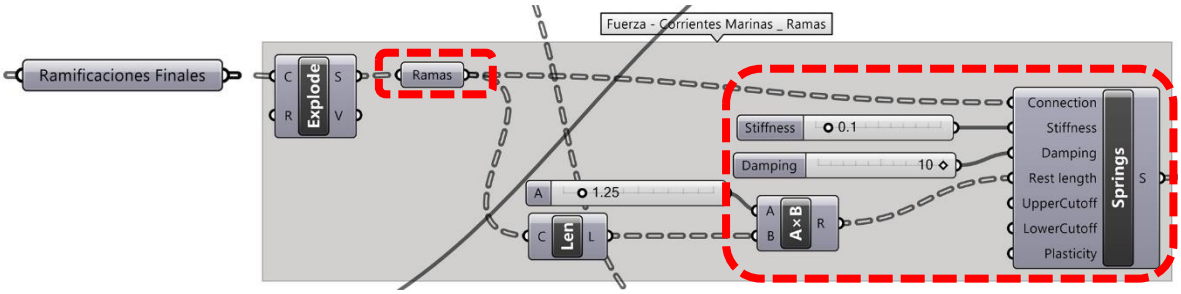
02 – El componente *Length curve* y *multiplication* definirán el límite de elasticidad de cada línea.

El componente *Length curve* nos ayudará a conocer la dimensión de cada iteración de líneas de generación, incorporando a su vez un panel para ver los datos (no aparece en la definición), el panel calcula la dimensión de cada iteración de líneas (*1R + 2R + 3R +4R*); es de considerar que 1R es interpretado como el punto de generación inicial, por lo que las dimensiones de cada iteración se representan en el siguiente esquema:



02A – Longitud máxima de transformación en cada iteración generativa a través de las líneas (nR).

Los resultados de cada iteración corresponden a *{1R = 0}*, *{2R = 4.16}*, *{3R = 2.92}* y *{4R = 1.43}*, que al ser multiplicados por el componente *multiplication* y al añadir un parámetro numérico de *1.25*, obtendremos la longitud máxima (en rojo) de transformación de cada línea al aplicar la fuerza de elasticidad definida mediante el componente *Kangaroo–Springs*.



03 – El componente *Kangaroo – Springs,* fuerza a simular mediante el componente *Kangaroo Physics*.

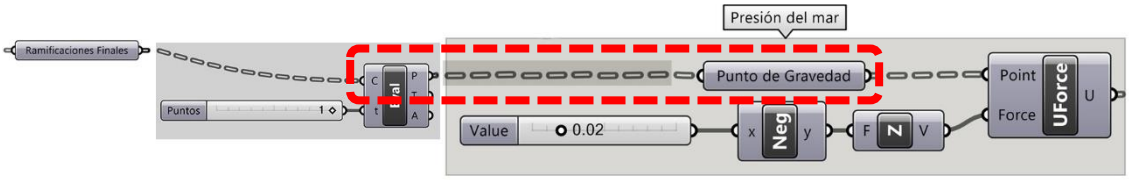
El último paso a realizar es conectar los objetos a modificar mediante la aplicación de la fuerza correspondiente (las líneas del sistema independientes de los puntos), lo cual propicia una relación de salida del componente *curve* (Ramas) y la entrada del componente *Springs - Connection*, con esto tenemos la fuerza a aplicar y al objeto al que se le aplicará tal fuerza. Además, hemos definido *Rest length* (longitud máxima de modificación en cada línea según la iteración generadora). Tenemos, ahora que definir la fuerza de rigidez que tendrán las líneas, es decir, la fuerza contraria a la elasticidad, por ello incorporamos un *slider* y un parámetro numérico de *0.1*, con el fin de que el objeto no sea tan rígido, de lo contrario, la fuerza de rigidez no permitiría la transformación elástica de las líneas. Por otra parte, la entrada *Damping – Springs* (amortiguación) será controlada mediante un *slider* con un parámetro numérico de *10*, valor que nos permitirá disminuir la fuerza a aplicar en los objetos (líneas), mientras el número sea menor, la fuerza tendrá un mayor impacto en todo el sistema y no habría un control paramétrico computacional en relación a la escala del organismo digital en cambio geométrico – lineal.

[D] – Generación de fuerza de presión en el agua

Con objeto de poder simular la aplicación de una fuerza de presión en el agua, es necesario conocer como la gravedad ejerce presión en la superficie terrestre. El valor constante de la fuerza de gravedad es de *9.81 m/s*^{2}, que se define según la aceleración que tiene un objeto al caer y la fuerza que ejerce el aire sobre ese objeto en movimiento. En el caso del agua, esta fuerza es variable según la profundidad del agua, no es lo mismo estar debajo a una profundidad de 1 metro que a una profundidad de 300 metros. Por lo anterior, se utilizará la analogía (aceleración) de la gravedad para la constante de presión, dónde el valor numérico de presión a ejercer en el organismo será baja, dado que el organismo natural crece en condiciones bastas de iluminación y energía del sol, y a una profundidad entre 5 m y 25 m.

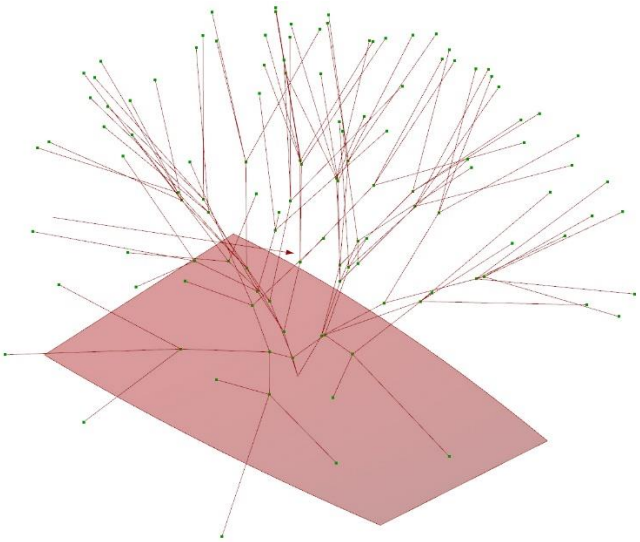
Grupo D – Presión del mar

Dado que el sistema actual final está definido a partir de un componente de generación cíclica (*Anemone*), necesitamos utilizar la geometría generada a partir de este ciclo, es decir, las curvas finales (Ramificaciones Finales). A partir de la selección de esta geometría representada a través de líneas, debemos evaluar aquellos elementos a los cuales aplicar la fuerza. El componente que nos ayudará a realizar lo anterior es *Evaluate curve*, para localizar todos los puntos generadores del sistema lineal y elementos a aplicar la fuerza de presión correspondiente mediante el componente *Kangaroo – UnaryForce*; y que al ser geometría de generación elemental (punto), serán capaces de modificar a las líneas finales, elementos y resultado de una generación secundaria.



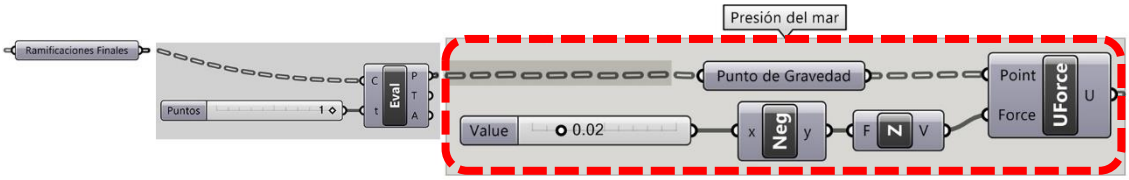
01 – El componente *Evaluate curve* y los *puntos de Gravedad* que serán modificados.

A partir de la relación entre el componente *curve* (Ramificaciones Finales) y el componente *Evaluate curve*, el control de entrada *t* y el control del sistema de selección a través de un *slider* con un valor de *1*, podremos excluir todos los puntos generadores del sistema de ramificación. Con ello obtenemos un segundo ciclo generador a través de puntos, al cual será aplicada una fuerza mediante el componente *Kangaroo – UnaryForce* que simulará la presión ejercida por el agua en el organismo digital – natural.



01A – Puntos a los cuales será aplicada la fuerza de presión mediante *Kangaroo – UnaryForce.*

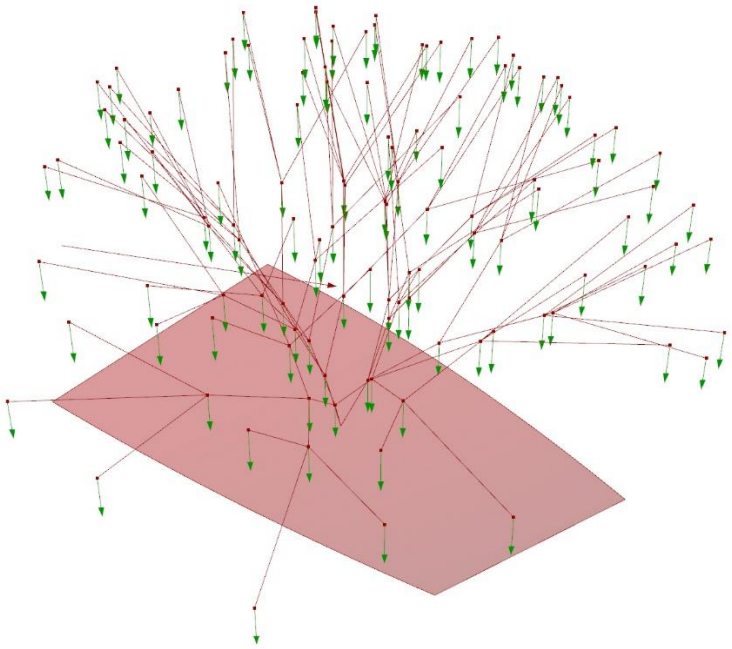
Al obtener todos los puntos secundarios a modificar, necesitamos definir la fuerza y la magnitud a aplicar en cada uno de ellos que simule la presión ejercida por el mar.



02 – El componente *Kangaroo – UnaryForce,* fuerza a aplicar sobre los puntos secundarios.

La relación del componente *point* (Puntos de Gravedad) y el componente *Kangaroo-UnaryForce* dará lugar a una fuerza que puede modificar la posición de objetos. Por otra parte, la entrada del componente *Force-UForce* necesita de la incorporación de una magnitud, que es definida a través del componente *slider* y un valor numérico de *0.02* que corresponde al valor de la fuerza a aplicar, y que cumple la condición de la poca presión ejercida a causa de la poca profundidad en el mar. En adición al anterior componente, se añadió una característica matemática de número negativo (*Neg)*, y un vector Z (*Z*), para determinar la dirección de la fuerza a aplicar sobre los puntos secundarios generadores. Es importante señalar que, en este caso de estudio, la simulación de la presión del agua en el

organismo solo contempla un eje de aplicación, ya que se ha aplicado con anterioridad la fuerza correspondiente a las corrientes marinas, que ejercen presión sobre el organismo. De lo contrario, este sistema computacional – natural, estaría expuesto a muchas fuerzas que no pueden ser controladas a partir de un logaritmo gráfico y, en parte también a los recursos de hardware limitados al momento de la simulación física computacional.

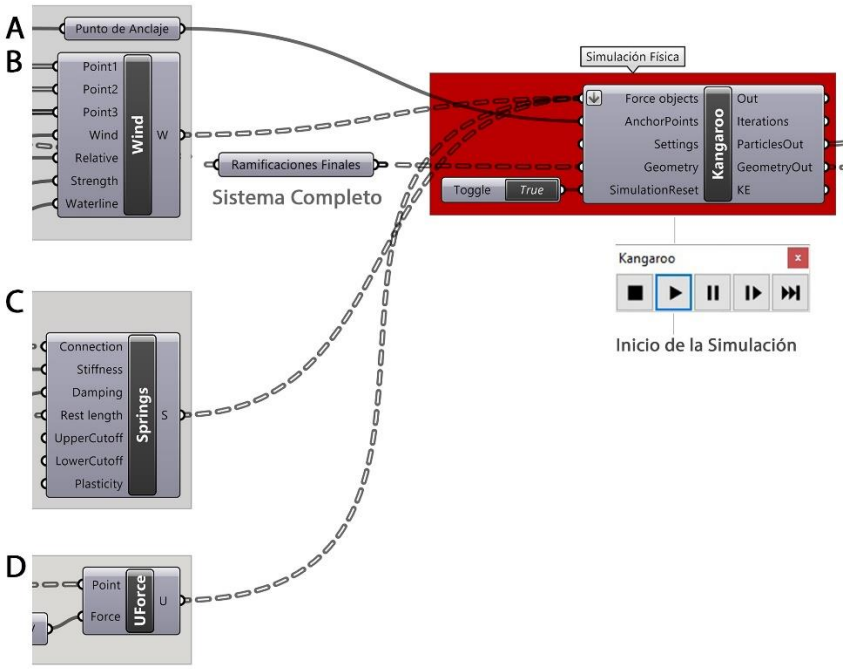


02A – *UnaryForce* y los vectores de aplicación de la fuerza de presión marina.

4.8.2.1 – Simulación física – *Kangaroo Physics*

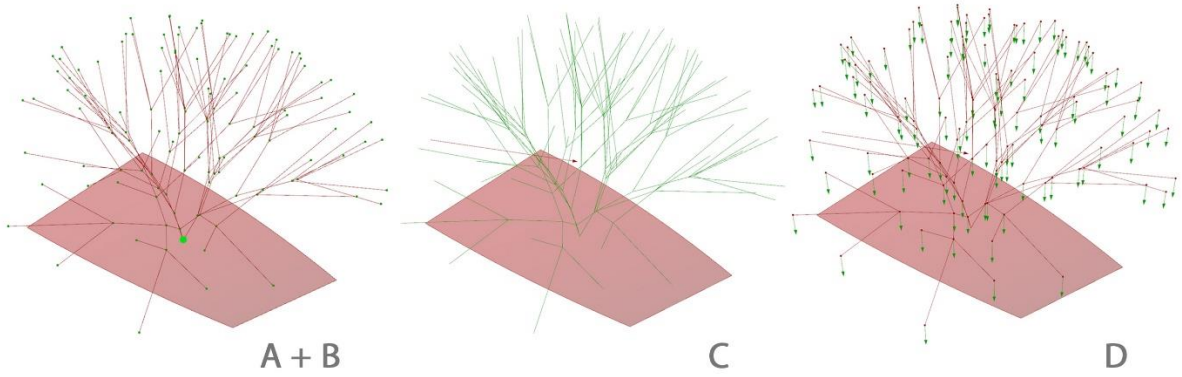
Realizar una simulación física – matemática a través de un logaritmo de programación gráfica, resulta en un proceso de diseño computacional muy completo, que abarca conceptos geométricos en cuanto a generación y modificación de componentes, además de la lógica intrínseca en cada relación. Por lo anterior, este tipo de simulaciones contemplan solo algunos de los aspectos relacionados a la producción de la forma, referidos a cada elemento que le da origen; en este caso, el análisis y el entendimiento de cada uno de los componentes resulta relevante, no solo por el resultado geométrico, sino por la relación directa con ciclos de generación paramétricos, referidos ahora a parámetros de modificación física. Este enfoque es necesario de comprender, dada la importancia que tiene al contemplar variaciones formales que ya no dependen de los miembros de generación inicial, sino de funciones que modificarán a elementos primarios con el fin de dar lugar a variaciones formales independientes, basadas en la aplicación de fuerzas de control generativas - paramétricas.

Este tipo de variación formal – física - matemática será llevada a cabo a través del plugin *Kangaroo Physics* en *Grasshopper*. Al contemplar cada una de las etapas de generación de fuerzas y elementos afectados, hemos sido capaces de distribuir estas fuerzas a cada miembro geométrico de la generación inicial, que será afectada a través de una simulación física que proporcionará resultados variables en forma. Así, el esquema de inicio de la simulación se resume en el siguiente esquema:



01 – Fuerzas a aplicar e inicio de la simulación en el sistema primario de ramificación.

Al haber obtenido los componentes de aplicación de fuerzas, por medio de un sistema ramificado inicial, el resumen (salidas *Punto de Anclaje* + *W* + *S* + *U*) debe incorporarse al a la entrada *Force Objects* (Objetos de fuerza) que serán modificados; este parámetro de entrada deberá calcular a los elementos de forma unitaria, es decir, en un solo elemento, es por eso que la condición de *Flatten* se encuentra activa (flecha hacia abajo). Por otra parte, el sistema geométrico a modificar es el ciclo de generación final representado a través del componente *curve* (Ramificaciones Finales), que nos proporcionará un resultado basado en puntos y líneas (*ParticlesOut* y *GeometryOut*). El componente referido a la letra *A* será el soporte de nuestro sistema, al ser el generador inicial de la geometría y ciclos de ramificación. La relación del botón de reinicio (*Toggle*) con la simulación, nos permitirá realizar el cálculo una y otra vez desde el inicio; cuando la condición booleana sea verdadera (*True*) la simulación estará funcionando, cuando sea falsa (*False*), no funcionará. Al hacer doble en el componente *Kangaroo* y presionar el botón *play*, la simulación será realizada en tiempo real, representada en el entorno 3D de trabajo *de Rhinoceros 5.0*.



01A – Geometría final – generadora, a modificar a través de la simulación de aplicación de fuerzas.

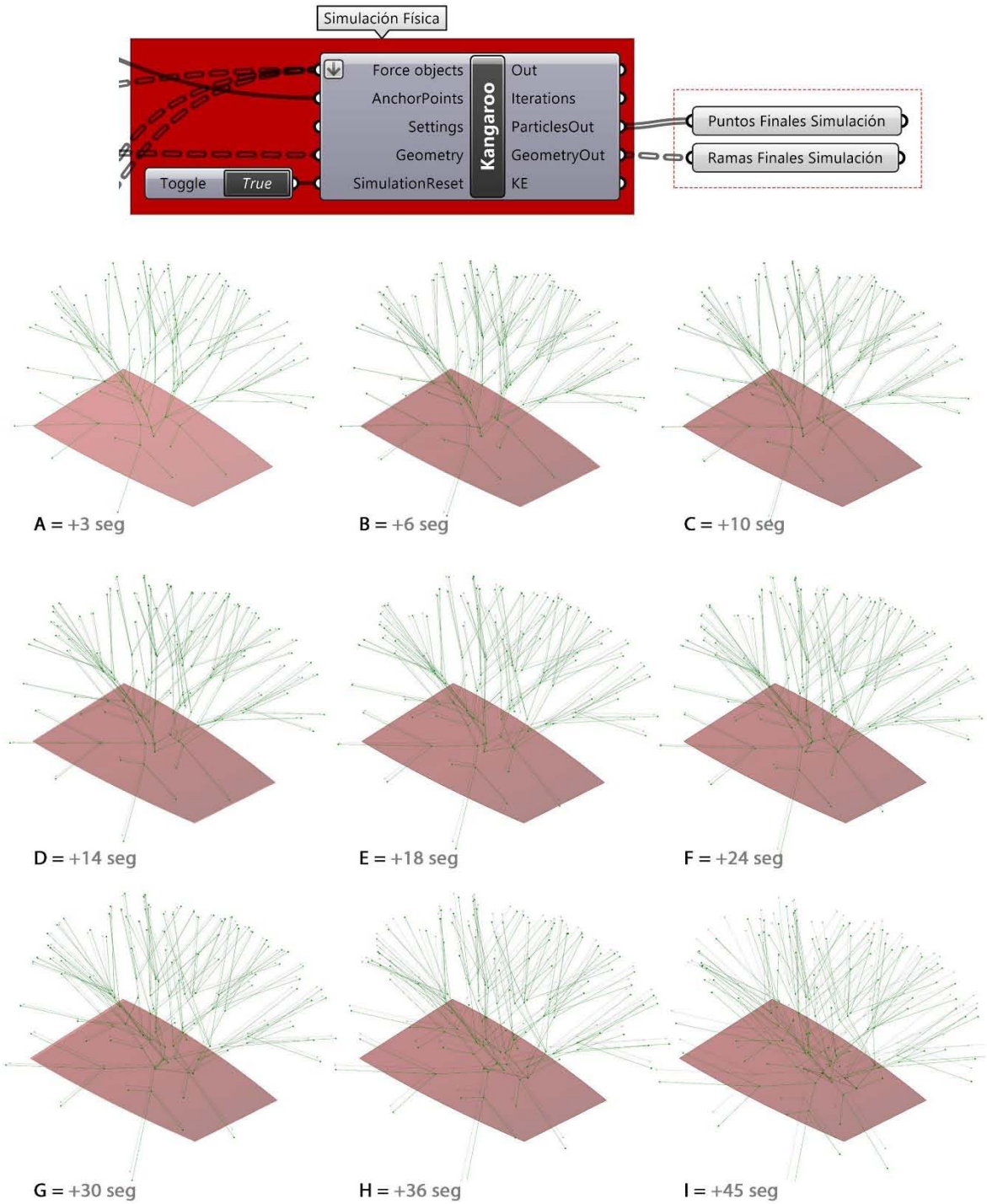


Figura 90. Variabilidad geométrica a partir de la simulación de fuerzas físicas, basado en un algoritmo de programación gráfica de *Grasshopper* y el uso del plugin *Kangaroo Physics,* representativo de la *[2] _ Etapa – Enfoque: [2 – 3] y [3 – 3].*

4.8.3 – [3] _ Etapa – Enfoque: [3 – 4] y [4 – 5 – 6]

En la anterior etapa, desarrollamos un proceso de simulación a través de la aplicación de fuerzas físicas a un determinado objeto, cuyo resultado es interpretado en geometría variable de puntos generadores y líneas. Una vez realizada la simulación en base a un parámetro de tiempo variable en segundos, seremos capaces de obtener el tercer ciclo de generación, el cual contempla las iteraciones iniciales y la geometría modificada por medio de *Kangaroo – Grasshopper*. Los resultados de esta simulación, aplicados a este enfoque, servirán de punto inicial para la generación de geometría que responda al comportamiento del sol, es decir, su movimiento a lo largo de la bóveda celeste, como medio de generación formal variable, basada en el principio de menor a mayor cantidad de materia según la cantidad de energía que absorbe el organismo.

Por lo anterior, en este enfoque se contemplan conceptos de física, programación gráfica, biología y geometría como medios de análisis y representación en este proceso de diseño. Es importante señalar que tendremos distintas variables de ramificaciones en relación a la aplicación de fuerzas, es por eso que vamos a seleccionar el esquema F (figura 90), que contempla la modificación geométrica basada en fuerzas en el segundo 24. A partir de esta selección, hemos establecido y limitado a un objeto por las fuerzas presentes en su entorno externo, y por otra parte, definimos geometría base para la siguiente etapa de diseño, por medio de un modificador generador variable según su movimiento [Sol].

Otro aspecto a contemplar, en relación a la variabilidad de escala en los objetos finales geométricos, se relaciona con método de representación inicial. En este enfoque, nos limitaremos únicamente a representar este cambio de escala y generación en base al desplazamiento de un modificador, por medio de esferas, con el propósito de visualizar el cambio en tiempo real en relación a la posición del sol. De otra manera, este proceso estaría limitado por los recursos del hardware en dónde se desarrolla el algoritmo gráfico, ya que se tendría que contemplar la simulación en tiempo real y la generación de mallas geométricas que representen de manera adecuada la forma del organismo. La optimización de mallas geométricas se realizará en el enfoque siguiente y final, a través del complemento *Cocoon* para *Grasshopper*, tomando como referencia los datos variables de escala y la generación de ramificaciones por medio de las respectivas iteraciones lineales.

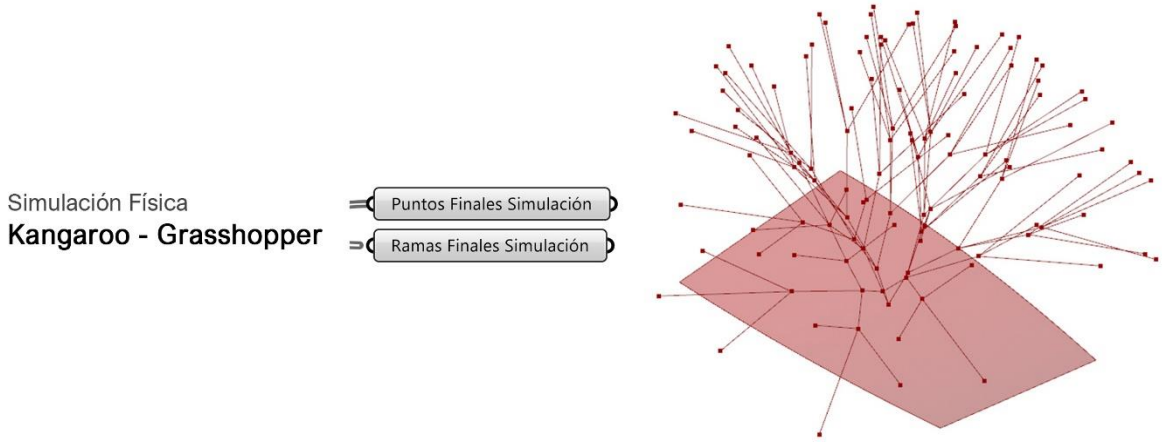


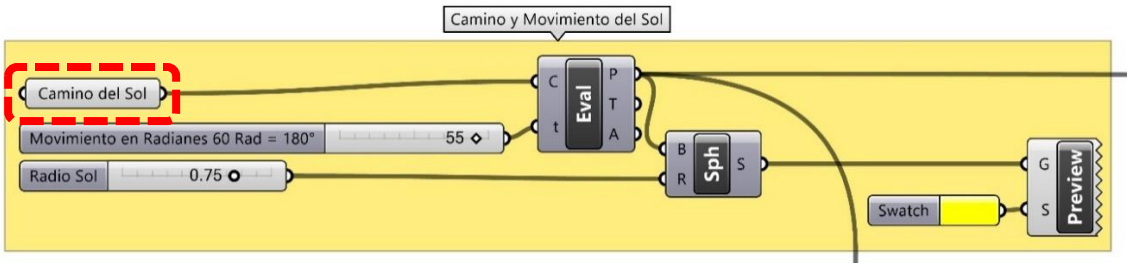
Figura 91. Geometría representativa del esquema F en el segundo 24 (figura 90). Punto de inicio para la generación de datos y geometría variable según un modificador en movimiento [Sol – Punto Atractor].

[A] – Punto atractor modificador y generador de geometría

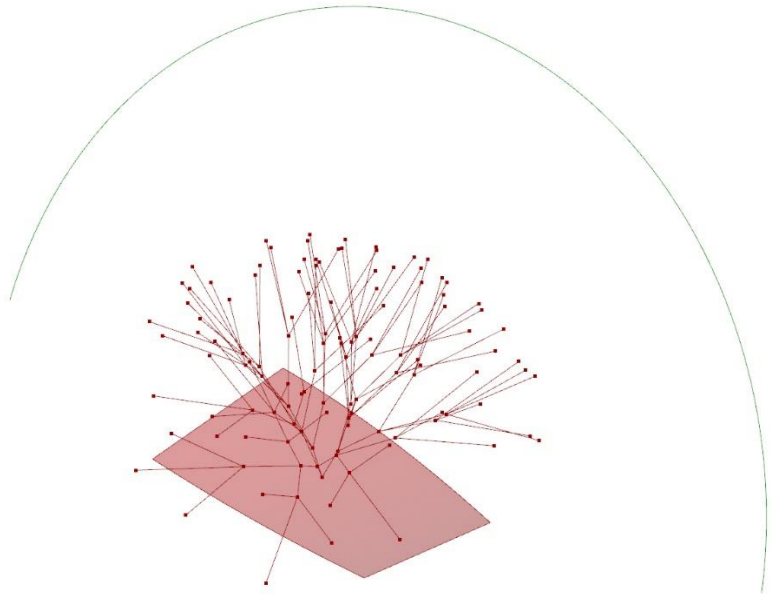
En relación a la generación de la forma, a través de la materia del organismo [Carbonato Cálcico], existen variables que limitan la generación de materia y que dan lugar a la forma final, una de estas variables es el movimiento y la cantidad de energía que irradia el sol y es absorbida por el organismo. Este concepto de generación es relevante, ya que limita y establece la cantidad de materia que tendrá el organismo, así como la cantidad de ramificaciones, es decir, a menor energía, menos cantidad de materia, y a mayor energía, más cantidad de materia. En este primer grupo, nos limitaremos a establecer un punto atractor que recorrerá una circunferencia, que representa al sol y el camino que recorre, además de establecer una relación con la geometría final de la simulación física.

Grupo A – Camino y Movimiento del Sol

Para poder generar geometría en base a un objeto atractor, es necesario primero, establecer un límite y camino de movimiento, en este caso, el desplazamiento que sigue el sol en relación a la bóveda celeste. Este tipo de comportamiento lo representaremos con una curva a 180° de generación (medio círculo), la cual realizaremos por medio de la herramienta *arc* (*start, end, angle*) de *Rhinoceros 5.0*. Una vez incorporado el arco, haremos la relación de la geometría física a la geometría de datos por medio del componente curve de *Grasshopper* (Camino del Sol). Este arco deberá envolver a la totalidad del organismo digital, con el fin de establecer una correcta relación del componente modificador (sol), y los elementos afectados (puntos – líneas).

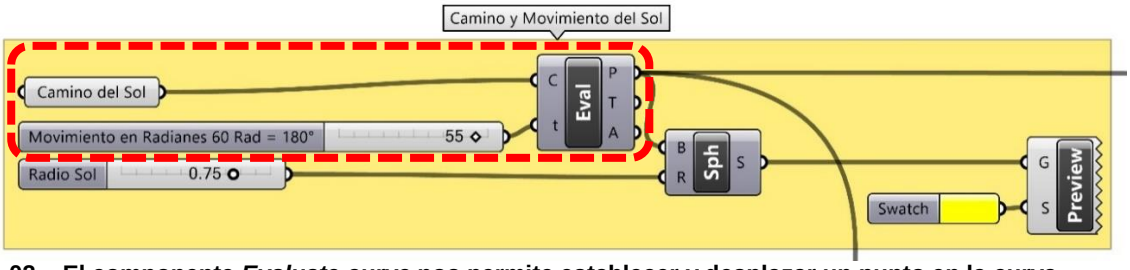


01 – Componente *curve* que representa datos del camino del sol, definido en *Rhinoceros 5.0*



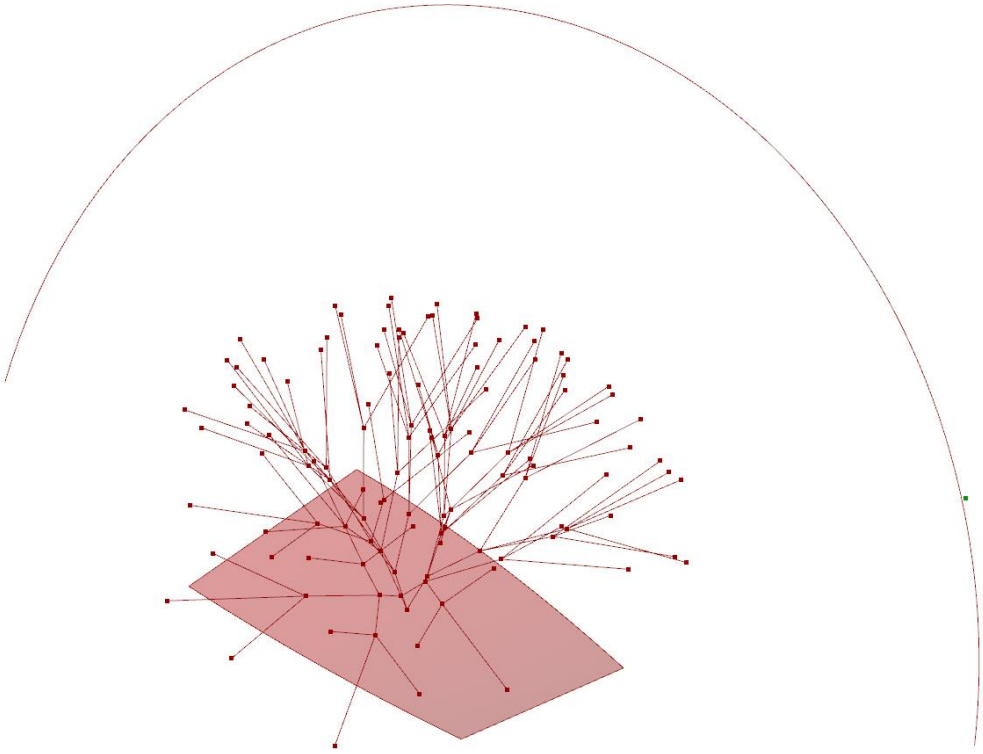
01A – Camino del sol que envuelve a la totalidad del organismo geométrico simulado.

Una vez definida la trayectoria del sol, necesitamos localizar un punto en su circunferencia, el cual nos permitirá establecer el elemento inicial generador del sol. Cabe señalar que, en este caso, el camino del sol se definió arbitrariamente dado que no hemos establecido un origen geográfico basado en coordenadas, dado que el organismo se encuentra en las costas del mar y puede tener localizaciones variables. Por lo anterior, se pueden establecer coordenadas y su respectiva montea solar, y el medio de producción del algoritmo sería, en general, el mismo, ya que dependería de una curva, así como en este proceso.



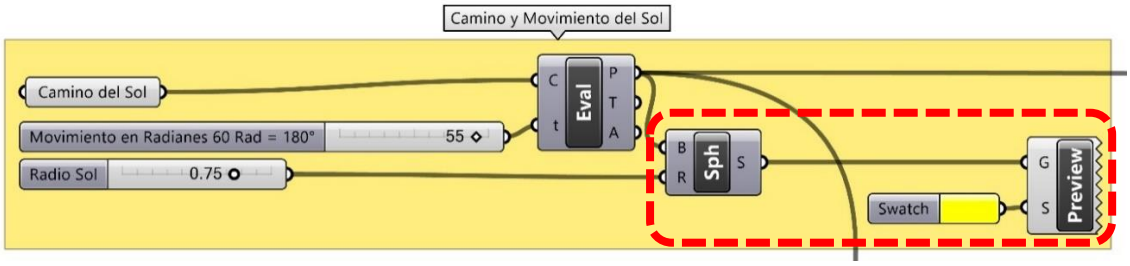
02 – El componente *Evaluate curve* nos permite establecer y desplazar un punto en la curva.

Para localizar un punto en la curva o trayectoria del sol correspondiente, emplearemos el componente *Evaluate curve*, el cual nos permitirá establecer un punto siempre en relación a su medio de generación (en este caso, el Camino del Sol), además de controlar su posición en el elemento geométrico generador, a través de la entrada *t,* en esta ocasión, hemos establecido una relación matemática en radianes, utilizando un *slider* con un límite numérico de 60, que equivale a 180° de rotación a lo largo de la trayectoria del sol. Otro aspecto importante, es que dependiendo del cambio numérico que controla la posición, la salida del componente *P,* será el resultado final de tal desplazamiento, que tendremos que relacionar con el sistema geométrico final (puntos y líneas de ramificaciones), con el fin de controlar la cantidad de materia generadora de la forma final.



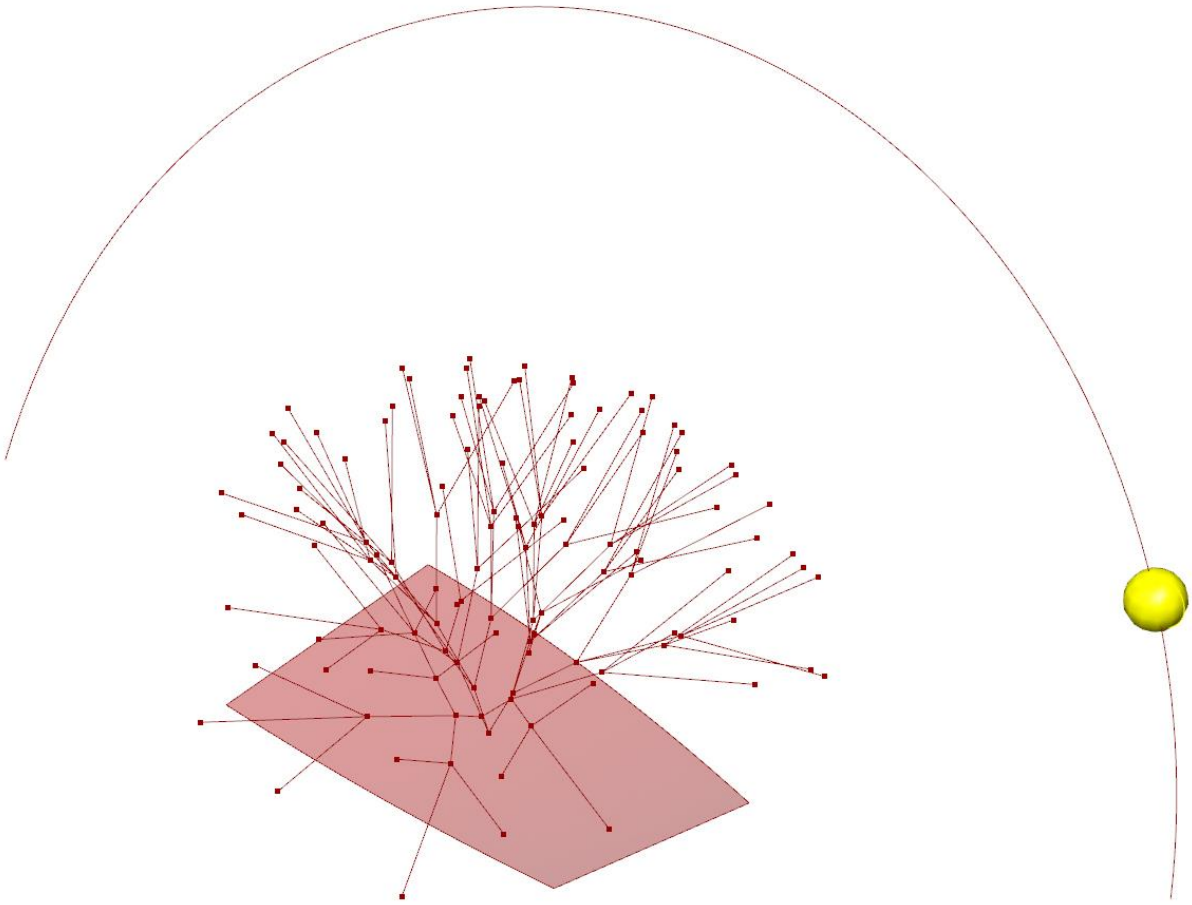
02A – Punto atractor y generador – modificador externo al sistema de ramificaciones simuladas.

Para representar el sol, generador a partir del punto en movimiento en la curva, utilizaremos el componente *Sphere* de *Grasshopper* para representar tal relación de mejor manera.



03 – El componente *Sphere* nos permite representar de mejor manera al Sol.

La relación entre el punto de salida *P*, que representa el desplazamiento del sol y su trayectoria, y la entrada *B* del componente *Sphere*, nos ayuda a representar de una manera gráfica más clara al sol, elemento externo al sistema de ramificaciones que limitará la generación de materia (mallas geométricas) según su movimiento. Por otra parte, el componente *Custom Preview* es usado para representar al sol, en base a un parámetro de color por medio del componente *Color Swatch - Amarillo*, con el fin de poder identificar y tener un mayor control del sol en el entorno de modelado 3D en *Rhinoceros 5.0*.



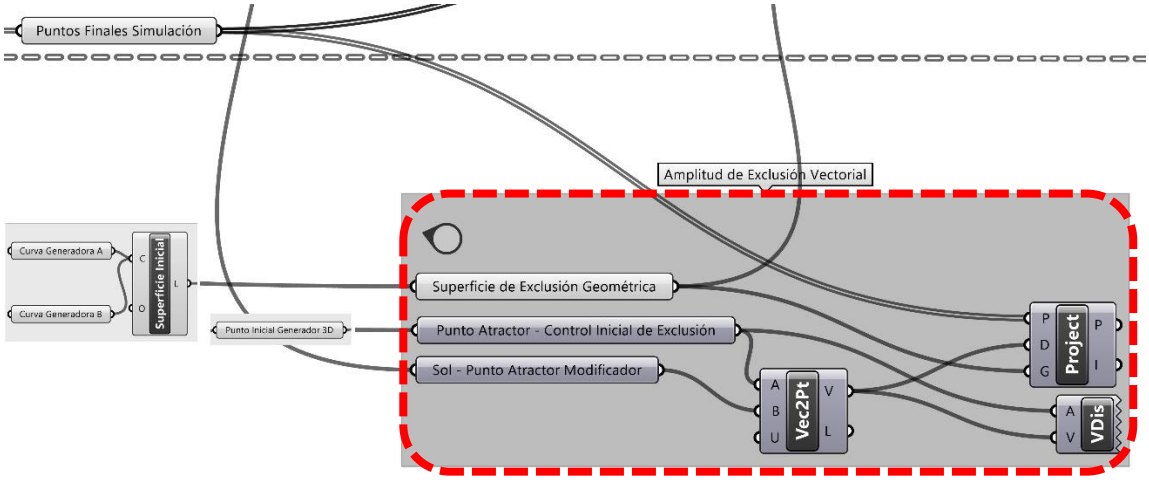
03A – Representación del sol, elemento externo generador y modificador de geometría

[B] – Relación y no-generación de materia en base a un punto atractor [Sol]

Un aspecto importante relacionado con la generación de materia es la no – generación sobre los elementos de soporte, es decir, en el punto generador y en la superficie marina desarrollada al inicio de este proceso de diseño. Por lo anterior, es necesario excluir aquellos elementos geométricos que no "desarrollaran" materia (mallas geométricas), así como la relación entre el sol y los elementos anteriores, con el fin de que no reaccionen en generación, debido al movimiento del punto atractor modificador (sol), pero que tengan relación con él, ya que son los medios generadores iniciales del sistema de ramificaciones.

Grupo B – Amplitud de Exclusión Vectorial

Para poder realizar una exclusión de datos, la cual contemple la relación vectorial, pero no la geométrica de modificación, necesitamos incorporar los elementos generadores del sistema de ramificación. Es por eso que la superficie y punto inicial generador deberán de ser clonados, con el propósito de manipular esa relación geométrica no-geométrica, pero sin afectar los ciclos de generación iniciales. Los *Puntos Finales de Simulación* obtenidos mediante la ejecución y aplicación de fuerzas deberán estar relacionados, de igual manera, con esta exclusión, ya que de ellos depende la posterior incorporación de geometría final variable en escala en función del desplazamiento del sol.



01 – Relación vectorial de elementos generadores iniciales y agente modificador por medio de los componentes *Vector 2 Points* y *Project Points*.

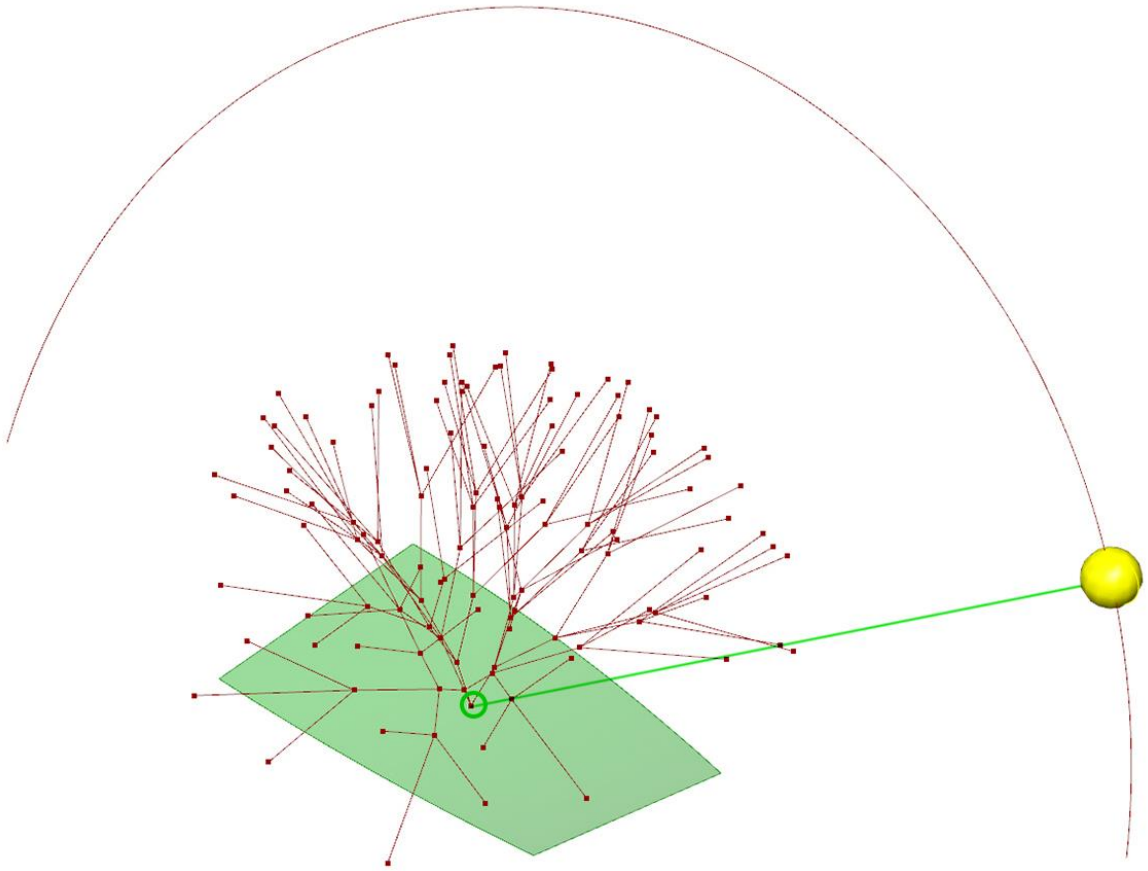
El componente *curve* (Superficie de Exclusión Geométrica) es una copia de la superficie generadora inicial, en la cual eliminaremos cualquier proceso de producción de geometría. El punto atractor, es una instancia del *Punto Inicial Generador 3D*, el cual nos permitirá hacer una relación vectorial, en función del *Sol – Punto Atractor Modificador*, con el fin de establecer siempre un cambio en función del desplazamiento del sol, pero sin afectar a la superficie, es por eso que se incorpora el componente *Project Point*, para proyectar los datos vectoriales en la superficie, en función de los puntos finales de la simulación, pero sin tener impacto en la generación de geometría en la superficie a causa de la modificación en la posición del sol.

A partir de este grupo seremos capaces de relacionar datos de posición vectoriales en función del control y desplazamiento de un elemento atractor, pero generador de geometría al mismo tiempo, como lo es el sol. Sin embargo, este cambio de posición no involucra la

adición extra de elementos geométricos en la superficie, ya que la hemos relacionado con un vector, pero no con algún componente que pueda generar objetos variables en relación al movimiento del sol.

Es importante tener en cuenta que, durante este procesamiento, todos los puntos generadores de las ramificaciones, se encuentran en la misma posición que el punto generador inicial 3D, debido a la relación establecida en el componente *Vector 2 points*, por lo que el desplazamiento del sol modificará sus componentes vectoriales, pero en la misma posición. Con esto hemos obtenido la relación de todos los puntos generadores en función del movimiento del sol, en cuanto a sus propiedades vectoriales. En este caso, el siguiente paso será regresar estos puntos que dependen del movimiento del sol a su posición inicial que contempla el resultado a partir de la aplicación de fuerzas con *Kangaroo - Grasshopper*, con el fin de generar las ramificaciones del sistema digital – natural.

Otro aspecto a tomar en cuenta es la no-generación de geometría en la superficie generadora inicial a causa del desplazamiento del sol. Sin embargo, la superficie siempre mantendrá una característica vectorial – no geométrica debido a la relación de los puntos generadores con el punto generador inicial, que depende de la misma superficie. Esta relación aún no es representada por medio de los componentes adecuados, ya que solo es una representación de datos en la salida del componente *Surface* (Superficie de Exclusión Geométrica) que será llevada a cabo explícitamente en el siguiente grupo.



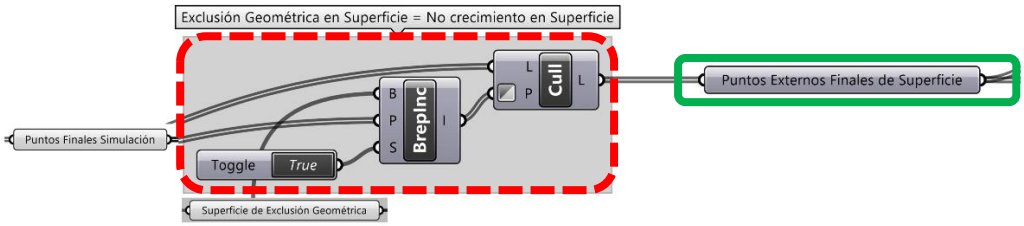
01A – Representación Geométrica - Vectorial de elementos generadores iniciales – simulados y agente modificador [Sol].

[C] – Eliminación de geometría vectorial no-generadora

Con el propósito de relacionar los datos vectoriales del anterior grupo y los componentes que nos permitan excluir aquellos elementos que producirán y que no producirán nueva geometría en función del desplazamiento de sol, utilizaremos el componente *Point in Brep* y el componente *Cull* de *Grasshopper.*

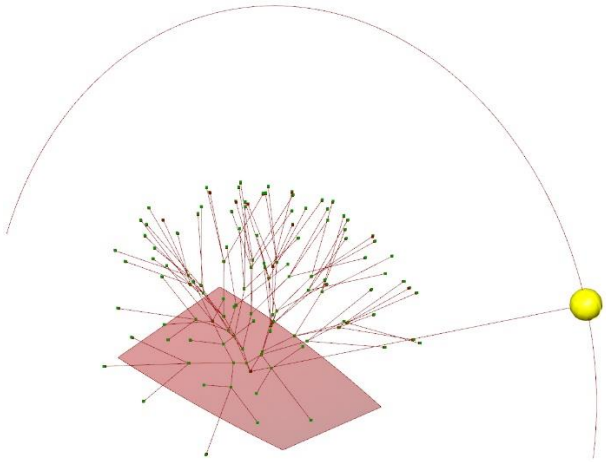
Grupo C – Exclusión Geométrica en Superficie = No crecimiento en Superficie

A partir de la relación de los *Puntos Finales de la Simulación* realizada con *Kangaroo-Grasshopper*, y la *Superficie de Exclusión Geométrica*, determinaremos aquellos elementos afectados por el movimiento del sol en cuanto a resultados vectoriales y geométricos.



01 – El componente Point *in Brep* y *Cull* nos ayudarán a seleccionar elementos de generación.

Al enlazar la salida de datos de los *Puntos Finales de Simulación* con la entrada de datos *P* del componente *Point in Brep* y la *Superficie de Exclusión Geométrica* con la entrada *B* respectivamente, hemos proyectado los puntos modificados por las fuerzas en la superficie de exclusión geométrica, pero que cuenta con datos vectoriales en relación a la posición del sol (Grupo B; la entrada *S* (*Strict*) y el valor booleano de *True* enfatizan está relación de manera unitaria. Tenemos aquí una gráfica *DAG* que relaciona posición (*Puntos Finales Simulación*) respecto a un objeto que no puede generar geometría (*Superficie de Exclusión Geométrica*), es decir, hemos restringido la siguiente producción de geometría a elementos estáticos generativos y a un objeto de soporte no generativo. El componente *Cull* (desechar) seleccionará aquellos elementos que generan o no generan geometría en relación con los puntos proyectados en la superficie. Es por eso que la entrada *P* se encuentra activa con un cuadro con una línea diagonal en blanco y negro (*Invert*), para seleccionar los puntos simulados que conservan la misma posición y generan geometría al exterior de la superficie (*Puntos Externos Finales de Superficie*), pero que no podrán generar geometría al estar en el interior de la superficie en función de la modificación de las ramificaciones a causa de la aplicación de fuerzas y el desplazamiento del sol.



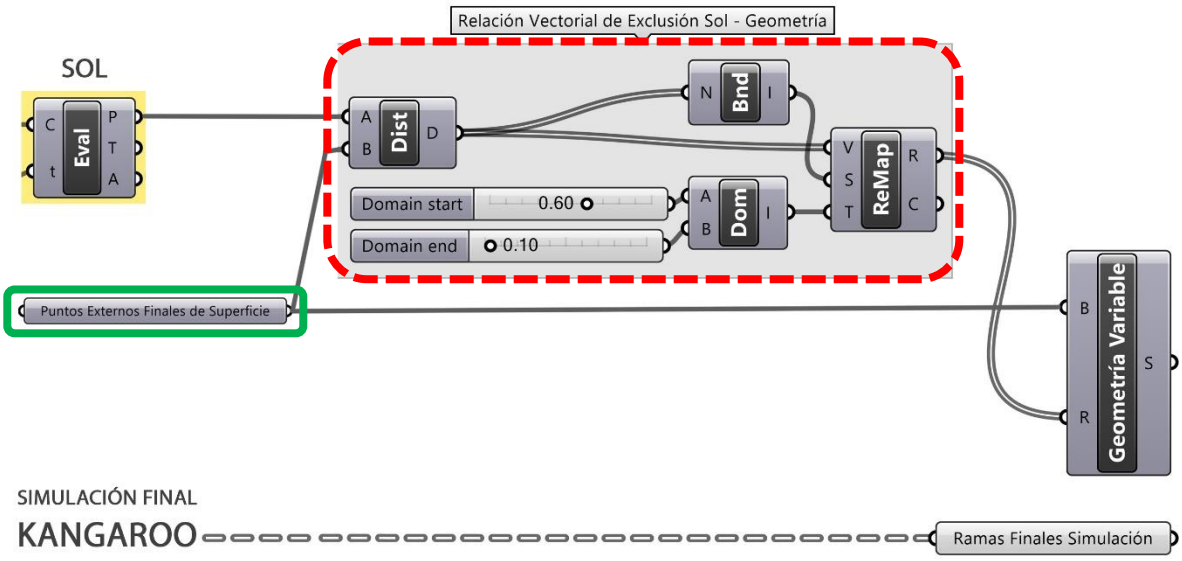
01A – Puntos excluidos generadores finales.

[D] – Generación de datos numéricos variables para producción de geometría

Hemos definido los *Puntos Externos Finales de Superficie* como aquellos elementos que podrán generar geometría en escala variable dependiendo del desplazamiento y la posición del sol, además de estar o no en la *Superficie de Exclusión de Geometría*. El siguiente paso es asignar un dominio numérico que pueda controlar la variación de escala en función de las características antes descritas y que nos permita representar y visualizar tales variaciones en un elemento de geometría muy simple como una esfera.

Grupo D – Relación Vectorial de Exclusión Sol - Geometría

Para desarrollar un dominio numérico variable de control de escala, es necesario establecer una relación entre el sol (*Eval*) y los *Puntos Externos Finales de Superficie*, a partir del conocimiento de la distancia que existe entre ellos, y que determinará el cambio de una escala mayor, cuando el sol se encuentre más próximo al sistema de ramificación generador, y una escala menor cuando el sol se encuentre más lejos.

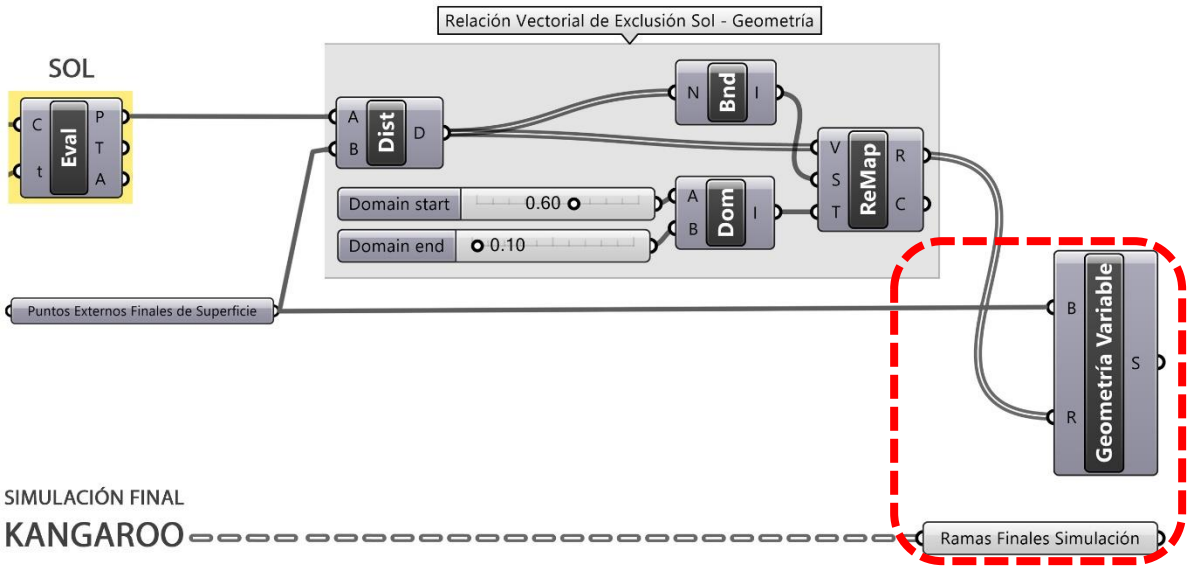


01 – Los componentes *Dist, Bounds, Dom* y *ReMap*, generadores de la variación numérica en escala.

El componente *Distance* nos permitirá hacer una relación de distancia entre el punto atractor modificador (*Sol)* y los *Puntos Externos Finales de Superficie*, con esto limitaremos la variabilidad de escala en función de que tan próximo se encuentre el elemento modificador. Por otra parte, esta relación será constante, por eso necesitamos emplear el componente *Bounds*, con el fin de tener un límite de distancia que varia de acuerdo al parámetro numérico de salida de *Dist* (D). El componente *Dom* (Dominio) nos permitirá establecer los límites máximos de escala a través de parámetros numéricos utilizando los sliders; el dominio inicial (*Domain Start*) es de 0.60, que se refiere al valor máximo que tendrá la geometría como parámetro de control de escala, y el dominio final (*Domain End*) correspondiente a 0.10 es el valor mínimo para el control de escala. Por último, dado que tenemos un parámetro de generación de distancia (*Dist*), un elemento que restrige los límites de la distancia de un objeto a otro (Sol – Sistema de Ramificaciones _ *Bounds*) y un dominio de control de escala mínima y máxima (*Dom*), incorporamos el componente *ReMap*

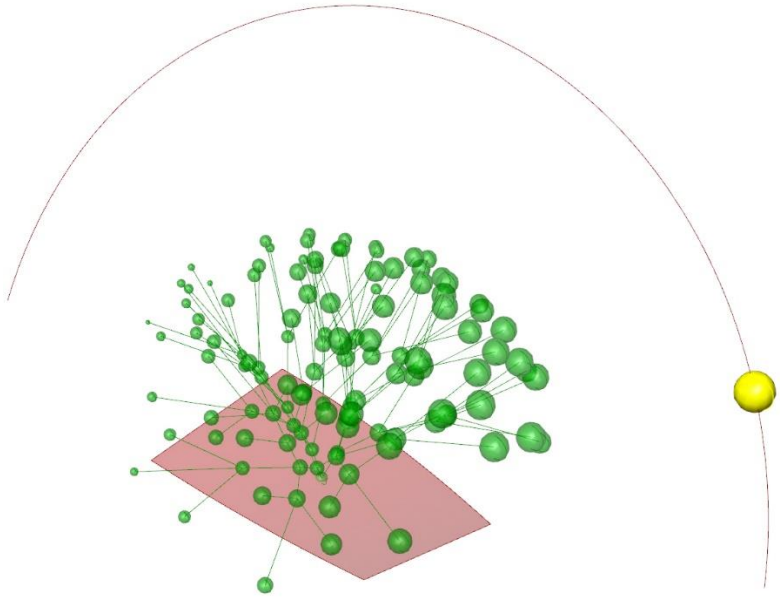
para generar un nuevo dominio numérico que pueda ser interpretado como parámetro de control numérico en cualquier componente referido a la generación de geometría.

El valor numérico de la salida del componente *R* de *ReMap*, contiene todas las variables antes descritas, que proporcionará un valor de control numérico en conjunto dependiente de la relación del sol con el sistema y su generación a partir del resultado de fuerzas físicas aplicadas en elementos geométricos generadores y variables en escala.



02 – Los componentes *Dist, Bounds, Dom* y *ReMap*, generadores de la variación numérica en escala.

Para representar la variabilidad en escala de los elementos a partir de un parámetro de control numérico, se incorporó el componente de generación de geometría *Sphere* (Esfera), con el fin de visualizar el comportamiento y la representación formal variable final. Así como la conservación de las *Ramas Finales de Simulación*.



02A – Geometría final variable en función del desplazamiento y posición del sol.

Los dos componentes anteriores (*Geometría Variable* + *Ramas Finales Simulación*) serán los elementos iniciales generativos para la posterior incorporación y optimización de mallas geométricas (materia), en referencia de la variabilidad de escala *R* y las direcciones lineales de generación en las iteraciones del sistema de ramificación fractal digital – natural.

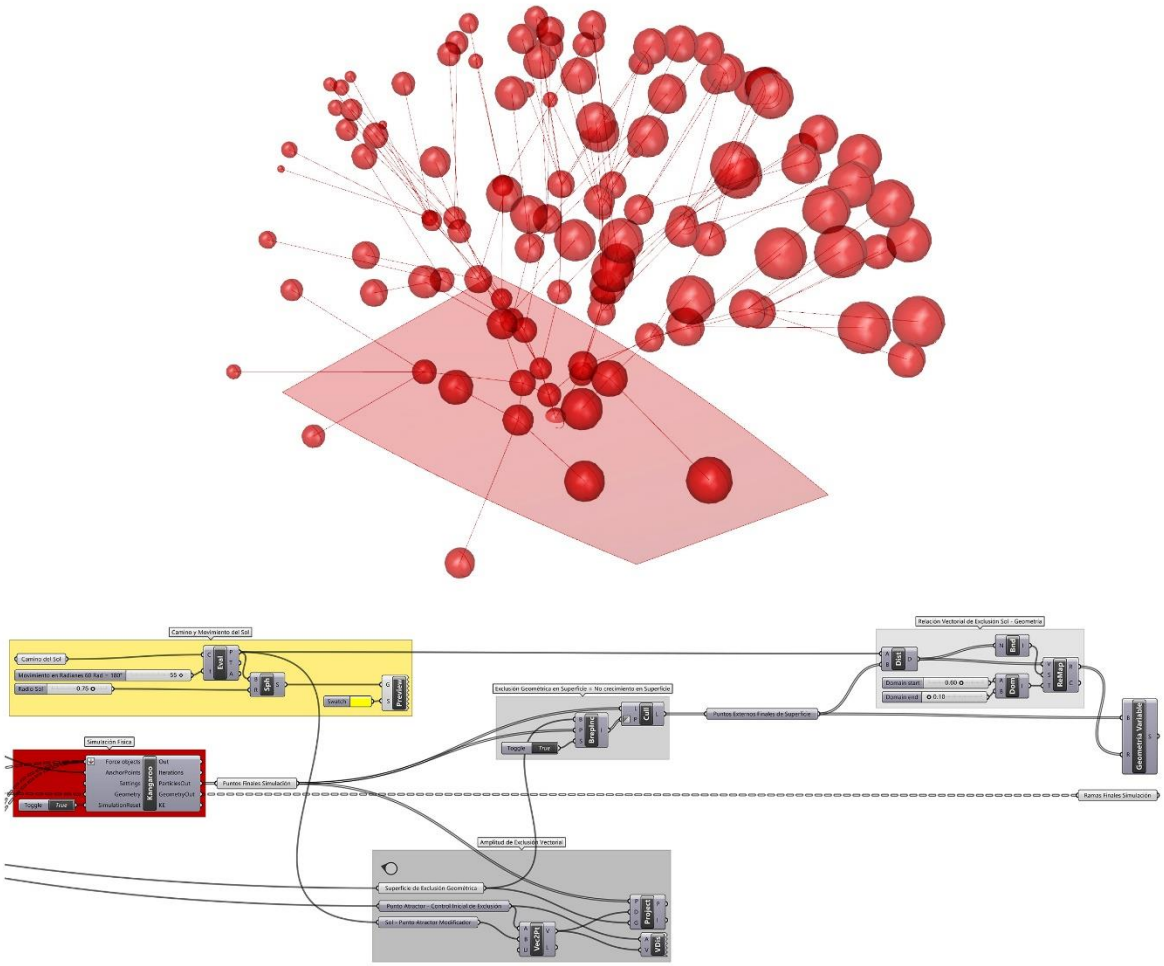


Figura 92. Elemento generativo variable en escala, en función del desplazamiento y la posición del sol que afecta al sistema de ramificación fractal; representativo de la *[3] _ Etapa – Enfoque: [3 – 4] y [4 – 5 – 6].*

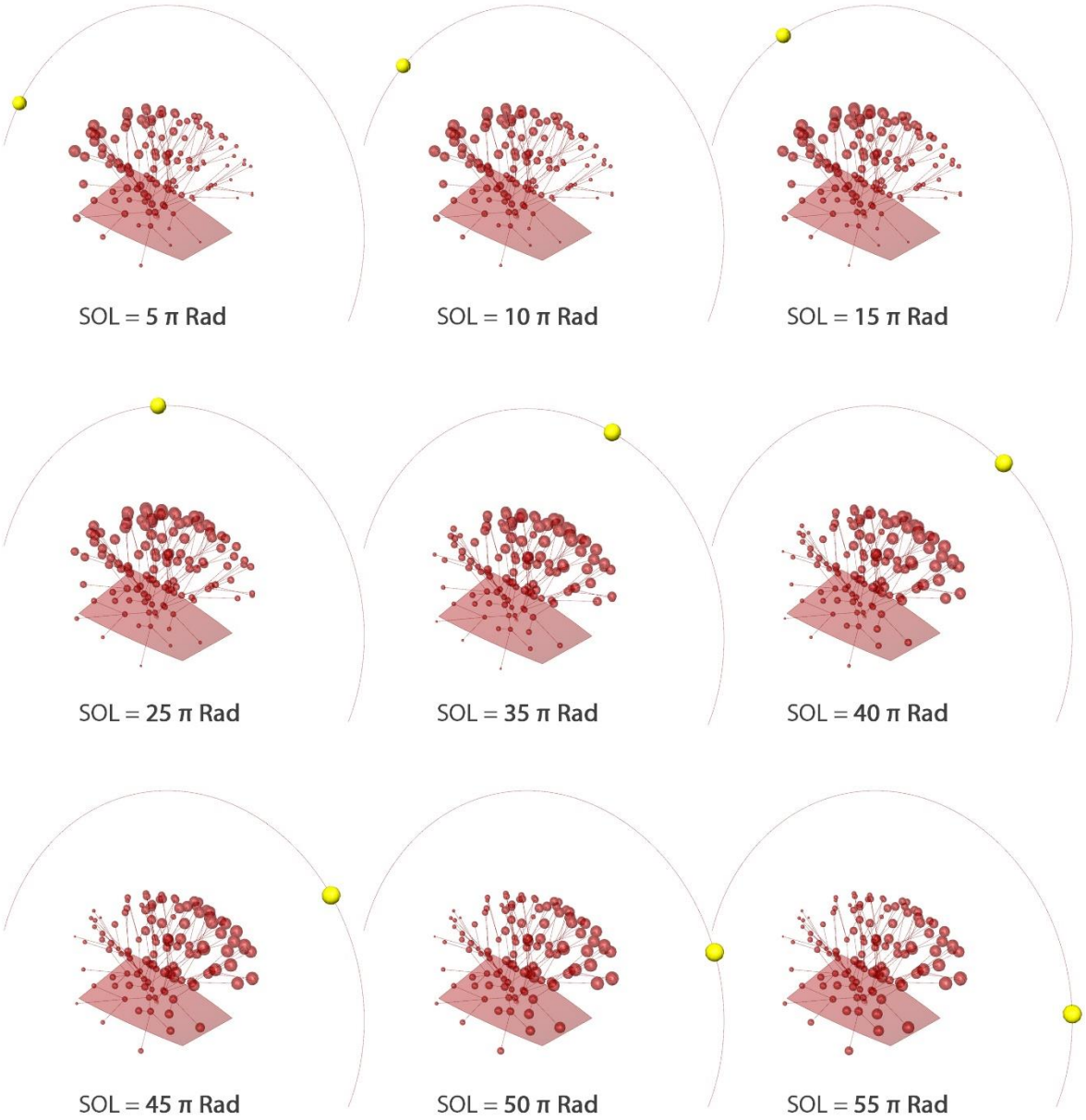


Figura 93. Variabilidad en escala de elementos geométricos finales del sistema, dependientes del desplazamiento y posición del sol, y de las fuerzas físicas aplicadas; representativas de la *[3] _ Etapa – Enfoque: [3 – 4] y [4 – 5 – 6].*

4.8.4 – [4] _ Etapa – Enfoque: [5] y [7 – 8]

En esta última etapa de diseño, vamos a representar de manera digital a nuestro sistema, mediante la incorporación de geometría referida a mallas. Las mallas geométricas nos ayudarán a imitar la configuración formal final del organismo digital hasta ahora desarrollado, en función de la cantidad de materia contenida en cada una de las ramificaciones, referida a al desplazamiento del sol. Es por eso que en esta etapa utilizaremos conceptos relacionados con la biomimética, las matemáticas y la geometría.

En el enfoque anterior determinamos una variabilidad de escala en la representación geométrica, relacionada con el desplazamiento y el movimiento del sol. En este último enfoque, desarrollaremos una relación de parámetros de control en escala y la geometría final, que representará el resultado formal de este proceso de diseño. Es importante aclarar que, hasta esta etapa, el proceso de diseño se ha desarrollado únicamente contemplando la generación de ramificaciones, la aplicación de fuerzas, exclusión y variabilidad escalar en respuesta a un agente atractor – modificador. A partir de este momento nos limitaremos a generar únicamente la geometría de representación final, imitando la configuración morfológica del organismo natural a través de un algoritmo de programación gráfica (*Grasshopper*) y un plugin extra (*Cocoon*), que nos permitirá optimizar la geometría final de mallas, con el propósito de no consumir demasiados recursos de *hardware* y que el tiempo de cálculo no sea tan extenso para poder observar los resultados variables.

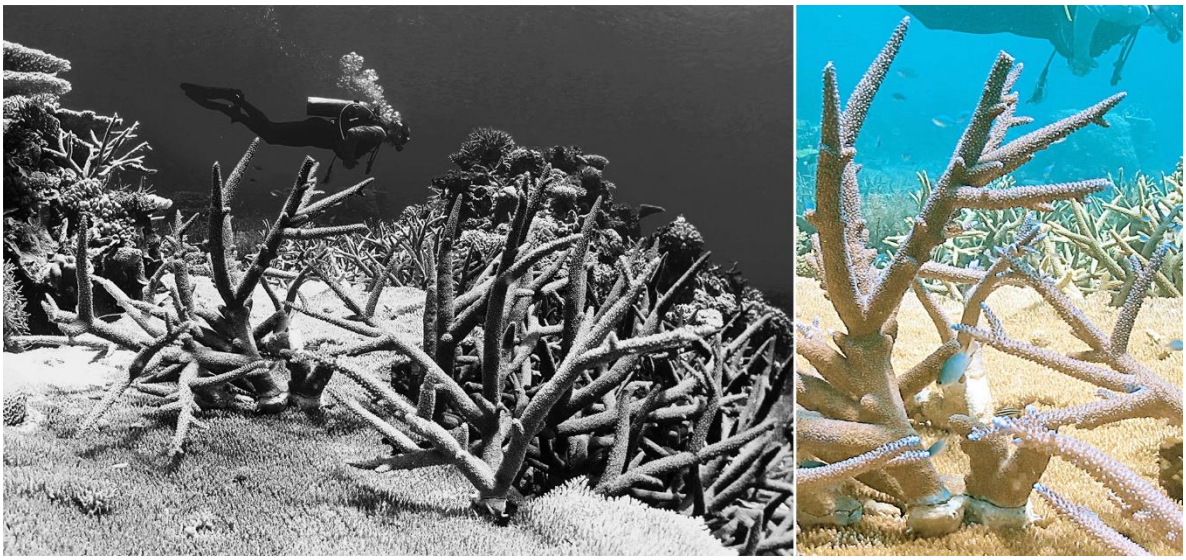


Figura 94. La configuración morfológica de un coral del género Acropora, nos servirá para establecer una relación geométrica final (Biomimética) a través de un componente de programación gráfica y un componente referido a la producción y optimización de mallas. Fuente: Fotografía de Dr. Ray Berkelmans / Australian Institute of Marine Science.

Por otra parte, es necesario contemplar los resultados del anterior enfoque, relacionados con el comportamiento a partir de la aplicación de fuerzas en el organismo digital y los parámetros numéricos de control de escala variable entre el sistema de ramificaciones y el desplazamiento del sol. En este caso, los tres datos más importantes son los valores numéricos que establecen una variación de escala *[R]*, la geometría variable que será afectada y que puede ser reemplazada (se pueden excluir las esferas, ya que solo es una representación para observar el correcto funcionamiento de la variabilidad) y los objetos lineales modificados por la aplicación de fuerzas (*Ramas Finales Simulación*) (figura 95).

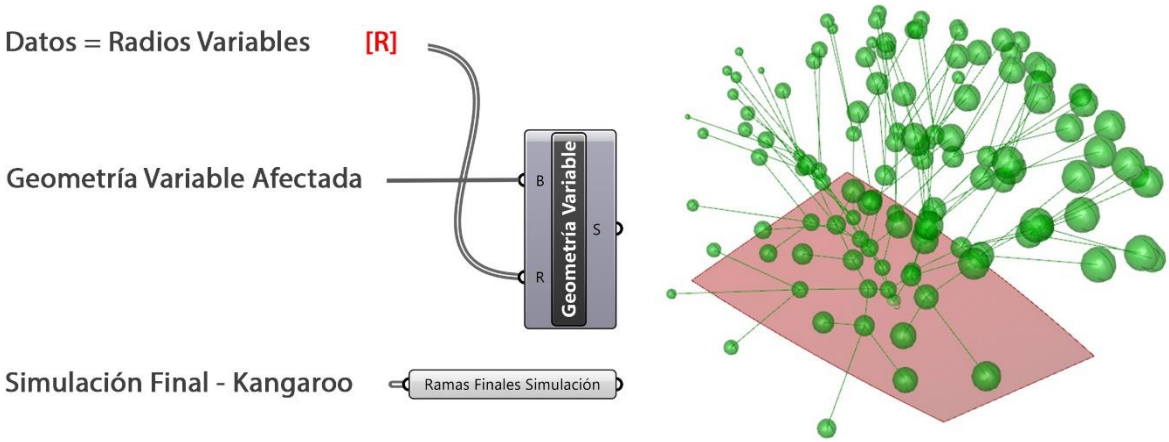


Figura 95. Datos iniciales de generación geométrica final.

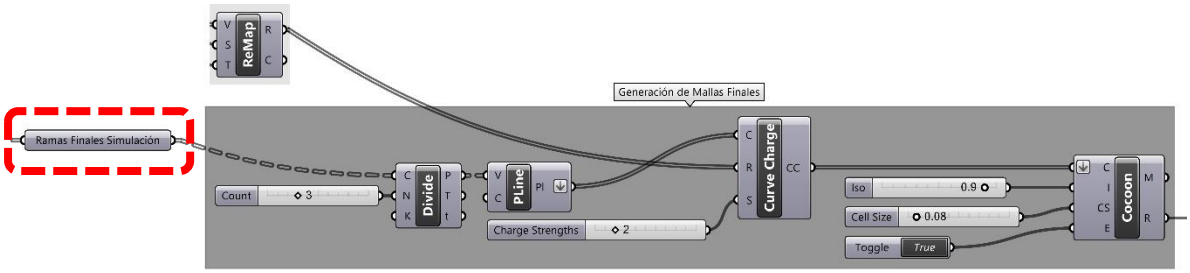
Una vez que hemos contemplado estos tres últimos datos, el paso siguiente es realizar la relación de datos a geometría que nos permita imitar un resultado formal similar al organismo natural. Es de considerar en este esquema (figura 95) que el parámetro de los radios variables se llevó a cabo en función de los puntos finales de la simulación, al igual que las ramificaciones; es por lo anterior que el sistema siempre producirá geometría variable en escala a través de la relación que existe con el desplazamiento del sol. Los componentes de generación de geometría pueden variar, siempre y cuando se contemple un parámetro de radio.

[A] – Transformación y Optimización de Ramificaciones Finales Simuladas

Para poder llevar a cabo la primera relación en este último enfoque, utilizaremos el plugin extra *Cocoon*[60] con el fin de optimizar el proceso de producción de mallas geométricas finales. Es necesario referir los tres datos finales del enfoque anterior como parámetros de inicio para la generación formal.

Grupo A – Generación de Mallas Finales

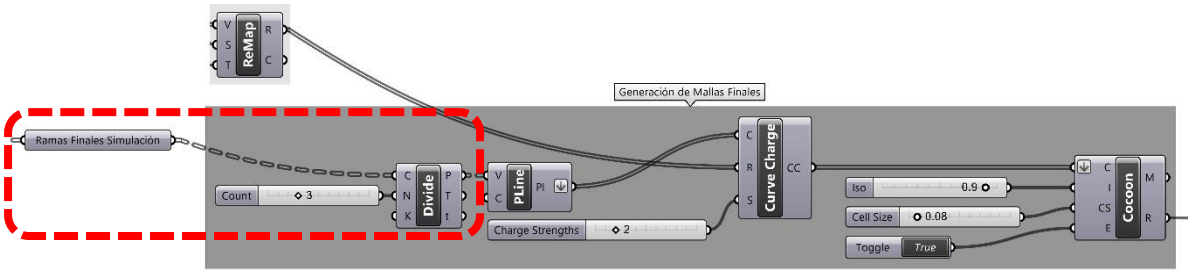
El primer paso es optimizar las *Ramas Finales Simulación*, que son elementos conformados por líneas que no pueden ser “relajadas” para adquirir un cierto grado de curvatura.



01 – Geometría lineal simulada = inicial generadora.

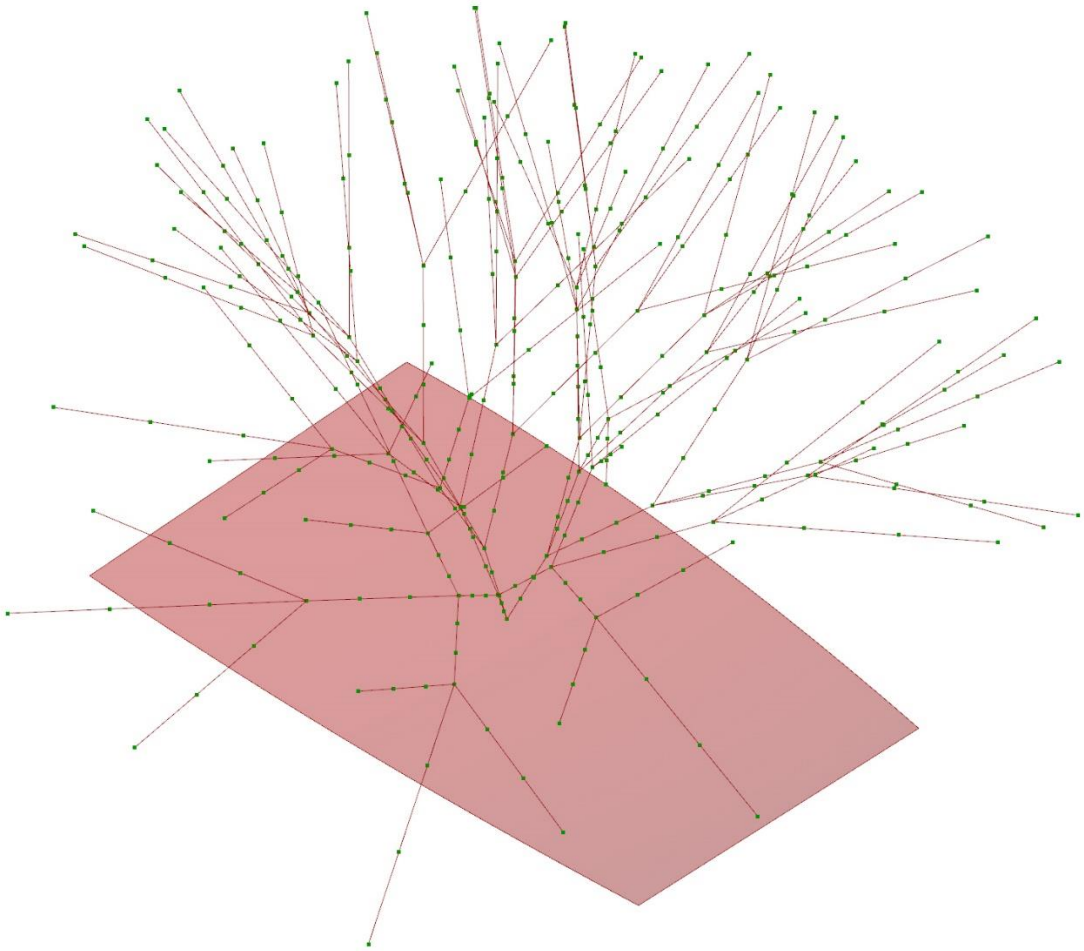
60 El plugin *Cocoon* es un complemento extra (plugin) para la interfaz de programación gráfica Grasshopper en Rhinoceros 5.0. Es un algoritmo muy sencillo para convertir iso-superficies en mallas poligonales. Está dirigido a envolver elementos geométricos existentes lineales, en combinación con puntos, Breps, curvas y parámetros de control numéricos, permitiendo a los usuarios tener variaciones potenciales en relación a la geometría a representar por medio de mallas optimizadas. Actualmente se encuentra en desarrollo por David Stasiuk. Para mayor información consultar: *http://www.bespokegeometry.com/2015/07/22/cocoon/.*

Para poder llevar a cabo lo anterior descrito, es necesario convertir las líneas de las ramificaciones en elementos de polilíneas, generados a partir de puntos de control en base a las *Ramas Finales Simulación*.



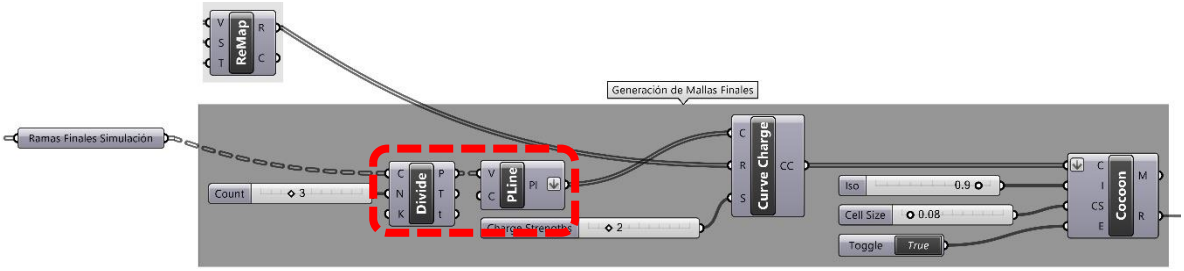
02 – El componente *Divide Curve* nos permite generar los puntos generadores de las polilíneas.

Con el fin de generar curvas de control a partir de las *Ramificaciones Finales Simulación*, utilizaremos el componente *Divide Curve*, el cual nos permite dividir cada una de las líneas en tres partes, a través de la relación de entrada *N* y un *slider* con un parámetro numérico de 3. A partir de estos puntos generados, en base a las líneas finales simuladas, podremos generar las curvas que nos permitan tener un mayor control y relajamiento en las ramificaciones, como componentes directrices de las mallas geométricas finales.



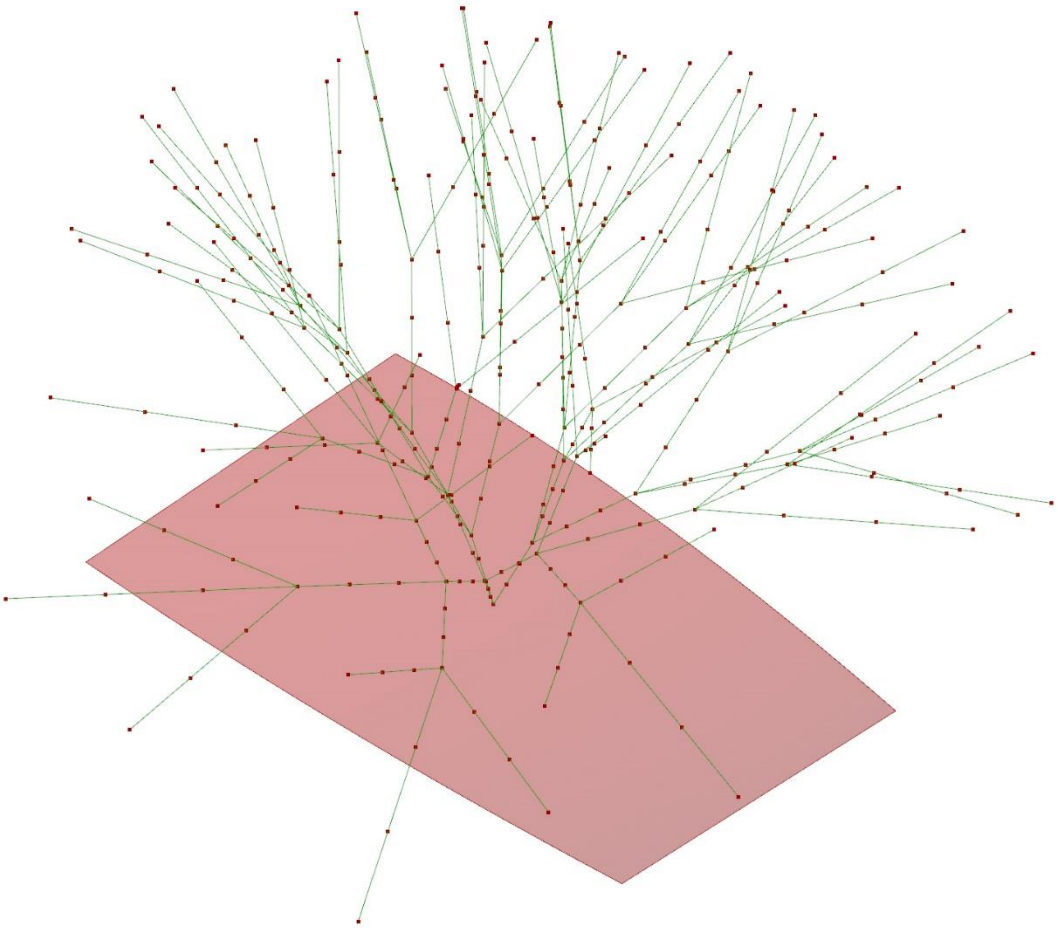
02A – Puntos generadores iniciales para la posterior producción de curvas directrices de control.

A partir de la generación de puntos de control y generadores, en base a las ramificaciones finales simuladas, podremos incorporar un componente que nos permita producir las curvas directrices por medio de polilíneas en nuestro sistema de mallas de representación finales.



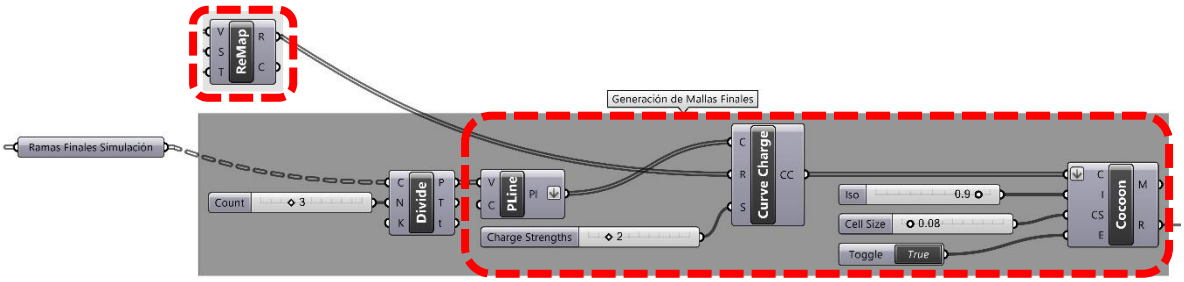
03 – El componente *Polyline* para producir las curvas de control, por medio de puntos generadores.

Una vez definidas las curvas directrices de las mallas finales, a través de la relación de la salida *P* (*points*) y la entrada *V* (*Vertices*) del componente *Polyline*, necesitamos relacionar todas las directrices no como elementos aislados en el sistema, sino como un sistema donde todas sus partes están interrelacionados y son codependientes, es por lo anterior que la salida *Pl* (*Polyline*) está activa con la condición de *Flatten* (flecha hacia abajo).



03A – Curvas directrices de control para la generación de mallas primarias finales.

Necesitamos ahora establecer una relación entre las curvas directrices y los datos numéricos de escala variable, con el propósito de limitar la cantidad de materia en cada una de las ramificaciones en función del desplazamiento del sol.



04 – El componente *Curve Charge* de *Cocoon*, generador primario de mallas geométricas.

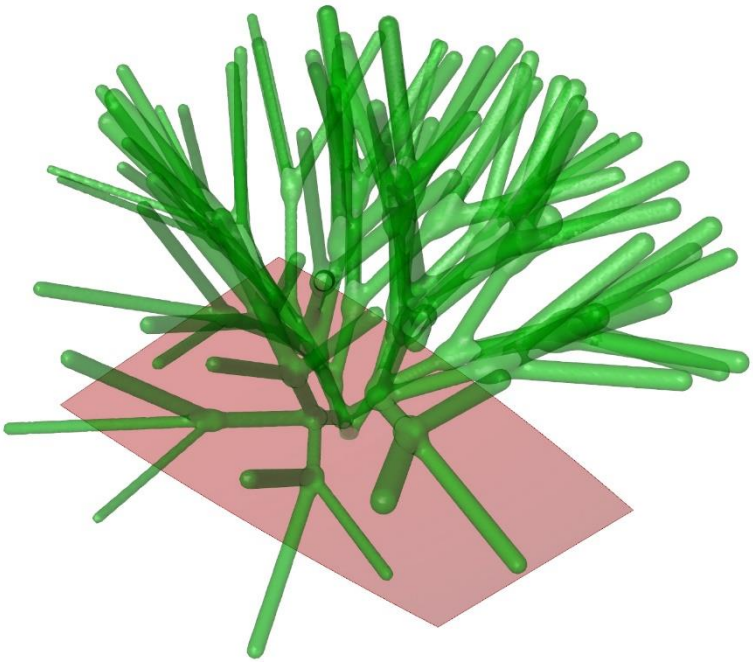
Curve Charge (Curva de Carga), es un componente que nos permite relacionar las directrices, es decir, las ramas finales (ahora polilíneas) con datos numéricos de control, referidos a la variación de escala, la cual puede ser llevada a cabo por medio de la relación de la salida *R* de *ReMap* con la entrada *R* de *Curve Charge*. Con lo anterior, las ramas tendrán una variación de escala dependiendo del desplazamiento y posición del sol, ya que el parámetro *R* de *ReMap* se generó a partir de este principio. El parámetro de entrada *S* (*Charge Strengths*) controla la intensidad con la que se lleva a cabo está relación a partir de un *slider* y un valor numérico de *2*; si el valor es menor, no habrá tanta variación en la escala de geometría de las ramificaciones, si es mayor, esta variación será más notoria.

Una vez que tenemos las curvas con un parámetro de control generador variable en función de la escala, es representado por medio del parámetro de salida *CC* del componente, necesario para establecer la relación para la generación del primer sistema de mallas, a través del componente *Cocoon*, el cual posteriormente, será optimizado. Para generar las mallas primarias, es necesario relacionar las curvas directrices con parámetros de control variables (*CC* de *Curve Charge*) y la entrada *C* del componente *Cocoon*, y restringir la lectura de datos a un elemento unitario, por eso activamos nuevamente la condición de *Flatten* (flecha hacia abajo). Otros aspectos a considerar son los parámetros de entrada *I,* CS y *E* del componente, explicados a continuación:

- ***Iso [I]:*** controla la relación entre las células de generación, mientras este valor sea mayor, las células estarán más cercanas, propiciando la generación de mallas continuas, en este caso, necesitamos un valor alto, por eso lo hemos establecido en 0.90 unidades a través de la relación con un *slider*.
- ***Cell Size [CS]:*** controla el tamaño de las células que se relacionan para generar las mallas continuas. Mientras este valor sea pequeño, las células se podrán observar como esferas independientes a lo largo de las curvas generatrices. Si el valor es mayor, las células podrán conformar mallas continúas en las generatrices. El parámetro es variable en relación al tamaño del objeto y de las ramificaciones, en este caso, se definió un valor de *0.68* unidades a través de un *slider*, adecuado al tamaño y escala de todas las curvas generatrices del sistema.
- ***Execute [E]:*** lleva a cabo o no el cálculo para la generación de geometría. Es controlado a partir de un parámetro booleano, *True* es encendido y *False* apagado.

El cálculo y procesamiento de los datos, parámetros y geometría del componente *Cocoon-Grasshopper*, nos brindará la oportunidad de visualizar un resultado formal basado en mallas de generación a partir de células continuas y su relación a través de curvas de

generación (directrices). Este primer acercamiento geométrico deberá de ser optimizado con otro componente, para variar la cantidad de materia en cada una de las ramificaciones, es decir, mallas irregulares en relación a las ramificaciones.



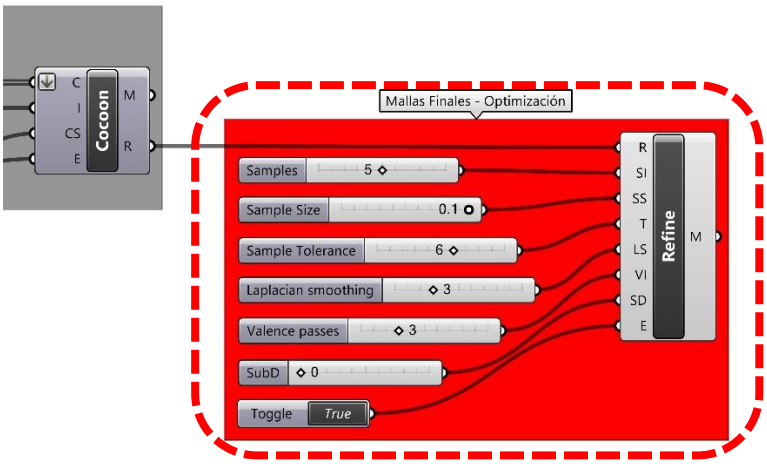
04A – Mallas geométricas de generación primaria en el sistema.

[B] – Optimización Final de Geometría

La optimización referida a la generación de geometría, es un proceso que nos permitirá producir mallas más complejas en relación de sus divisiones (teselaciones) y el control, a partir de parámetros de modificación generativos; independientemente de los recursos limitados o no del hardware en dónde se desarrolle.

Grupo B – Mallas Finales - Optimización

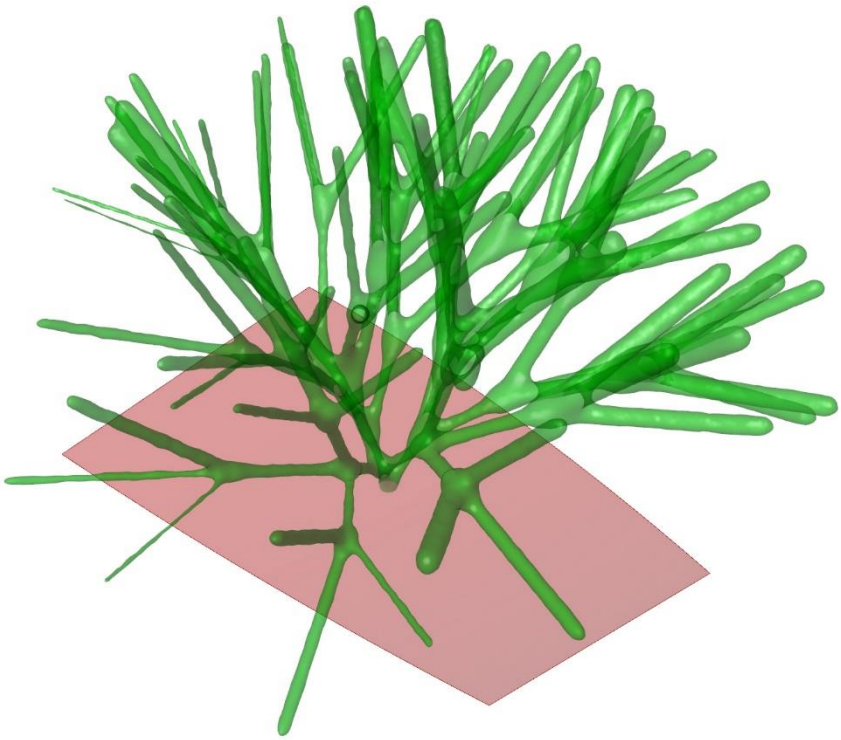
Para poder llevar a cabo la optimización de las mallas geométricas de generación primaria, utilizaremos el componente *Refine* de *Cocoon-Grasshopper*.



01 – El componente *Refine*, para generar las mallas secundarias finales optimizadas.

A fin de imitar la configuración morfológica del organismo natural, hemos incorporado un elemento de optimización de geometría, representativo de la cantidad de Carbonato Cálcico distribuido a lo largo de las ramificaciones del sistema, por medio de mallas variables en las teselaciones que las conforman. El componente *Refine* optimizará la cantidad de materia a representar por medio de mallas geométricas. Los controles y parámetros necesarios del componente para generar la geometría se explican a continuación:

- ***Refinement Data [R]:*** datos geométricos a optimizar, es decir, la salida *R* del componente *Cocoon*, el cual representa el sistema primario de mallas.
- ***Samples [SI]:*** número de repeticiones en las cuales serán divididas las caras de las mallas en relación a las directrices; el valor asignado en el *slider* es de *5* divisiones máximas, con el fin de equilibrar procesamiento y la visualización geométrica en 3D.
- ***Sample Size [SS]:*** el tamaño de las muestras, se refiere al tamaño de las células de generación de mallas continúas, si el valor es bajo, la relación de cada célula con otra será muy cercana, propiciando una malla continúa; si el valor es grande, la separación será más mayor, generando un sistema de mallas no continuo.
- ***Sample Tolerance [T]:*** generación de continuidad en mallas a partir de la proximidad de las células, si el valor es grande, habrá mayor espacio para llevar a cabo la relación, si es bajo, no habrá espacio y ni continuidad. El valor definido en el *slider* es de *6*, de gran continuidad espacial.
- ***Laplacian Smoothing [LP]:*** control de la resolución de las mallas. El valor definido en el *slider* es de *3*, ya que se requiere una alta resolución en la visualización final.
- ***Valence Passes [V]:*** controla la sustracción de geometría en las mallas, en este caso, el valor definido a través del *slider* es de *3*, con el fin de obtener una malla irregular, pero continúa en las teselaciones que la producen.
- ***SubD [SD]:*** incremento de las subdivisiones en las mallas, en este caso es definido en un *slider* con el valor de *0*, para evitar la generación lenta de geometría final.
- ***Execute [E]:*** lleva a cabo o no el cálculo para la generación de geometría. Es controlado a partir de un parámetro booleano, *True* es encendido y *False* apagado.



01A – Mallas secundarias optimizadas - representación geométrica final del sistema.

Hemos optimizado la geometría de mallas finales a través de un componente (*Refine-Cocoon*), que nos permite tener un mayor control y una mayor calidad en resolución y representación en el entorno de modelado 3D de *Rhinoceros 5.0*. Este sistema geométrico de mallas finales, está producido a partir de los primeros ciclos de generación de las ramificaciones (*Anemone*), de la aplicación de fuerzas para modificar el sistema (*Kangaroo*), de la variación de escala (cantidad de materia) en relación al desplazamiento-posición del sol (*Rhinoceros +Grasshopper*) y la optimización de la geometría por medio de parámetros de control simplificados (*Cocoon*); cumpliendo con un proceso de diseño paramétrico, dependiente de cada una de sus etapas de generación, a través de un algoritmo de programación gráfica. Analizamos e interpretamos algunos de los comportamientos y procesos generativos relativos a la forma de un sistema orgánico natural, a un proceso humano de producción formal artificial, basado en la naturaleza.

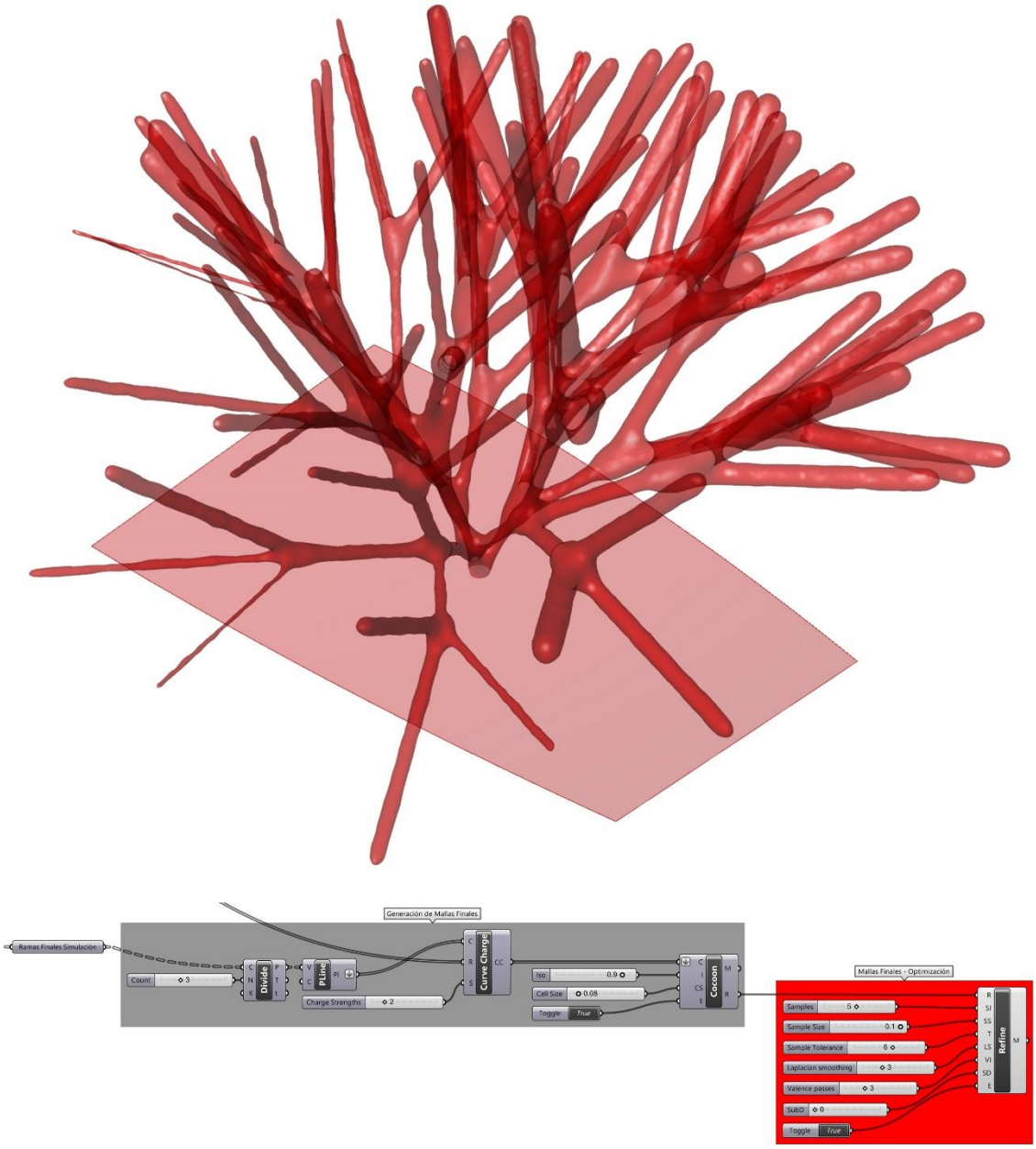


Figura 96. Configuración de mallas geométricas finales, resultado de un procesamiento lógico – logarítmico de control físico y formal, representativo de la *[4] _ Etapa – Enfoque: [5] y [7 – 8].*

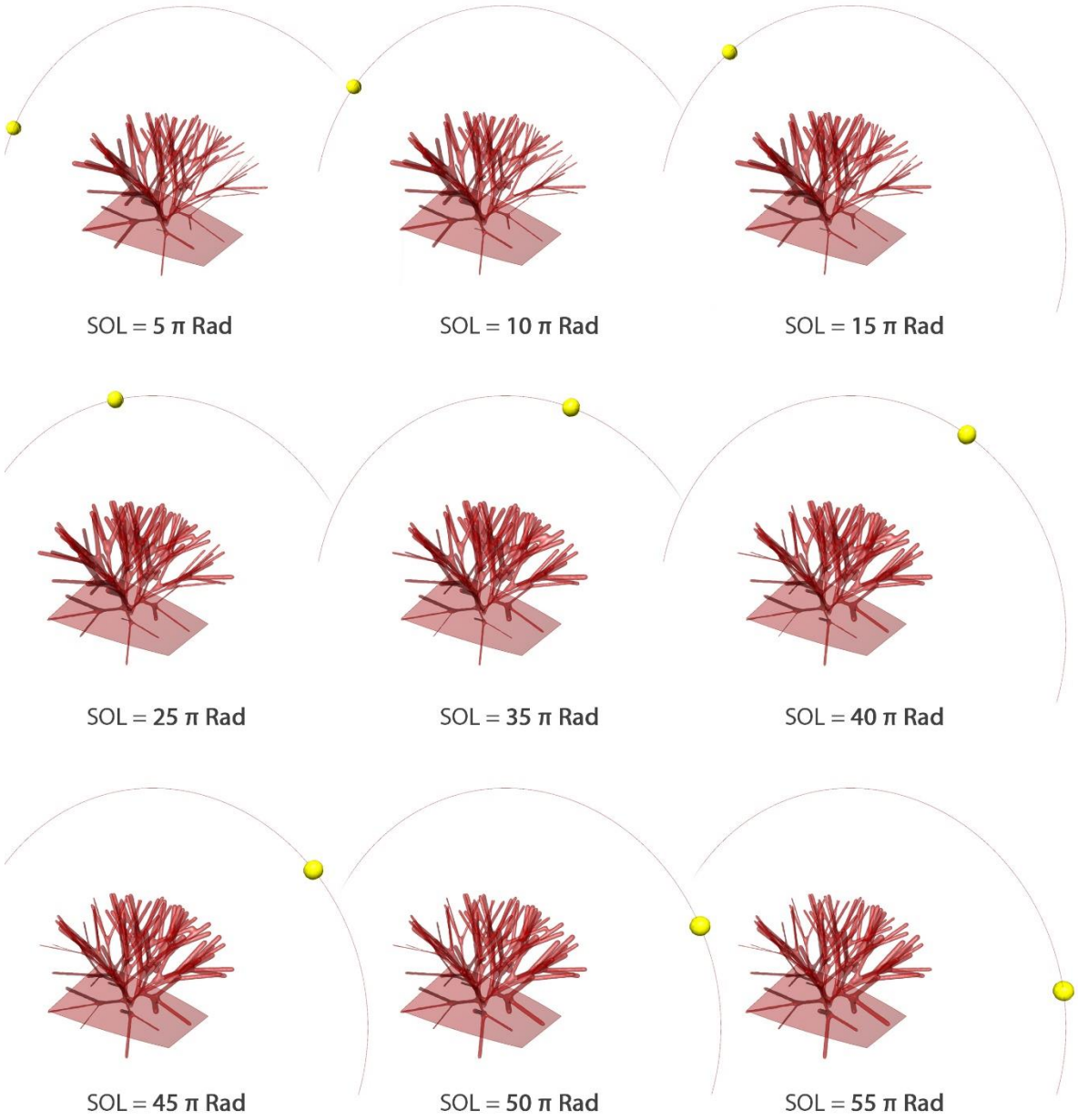


Figura 97. Variabilidad en escala de mallas geométricas finales del sistema, dependientes del desplazamiento y posición del sol, y de las fuerzas físicas aplicadas; representativas de la *[4] _ Etapa – Enfoque: [5] y [7 – 8].*

Figura 98. Iteración G = +30 seg - Incidencia solar = 45 π rad. Visualización superior final.

Figura 99. Iteración G = +30 seg - Incidencia solar = 45 π rad. Visualización frontal final.

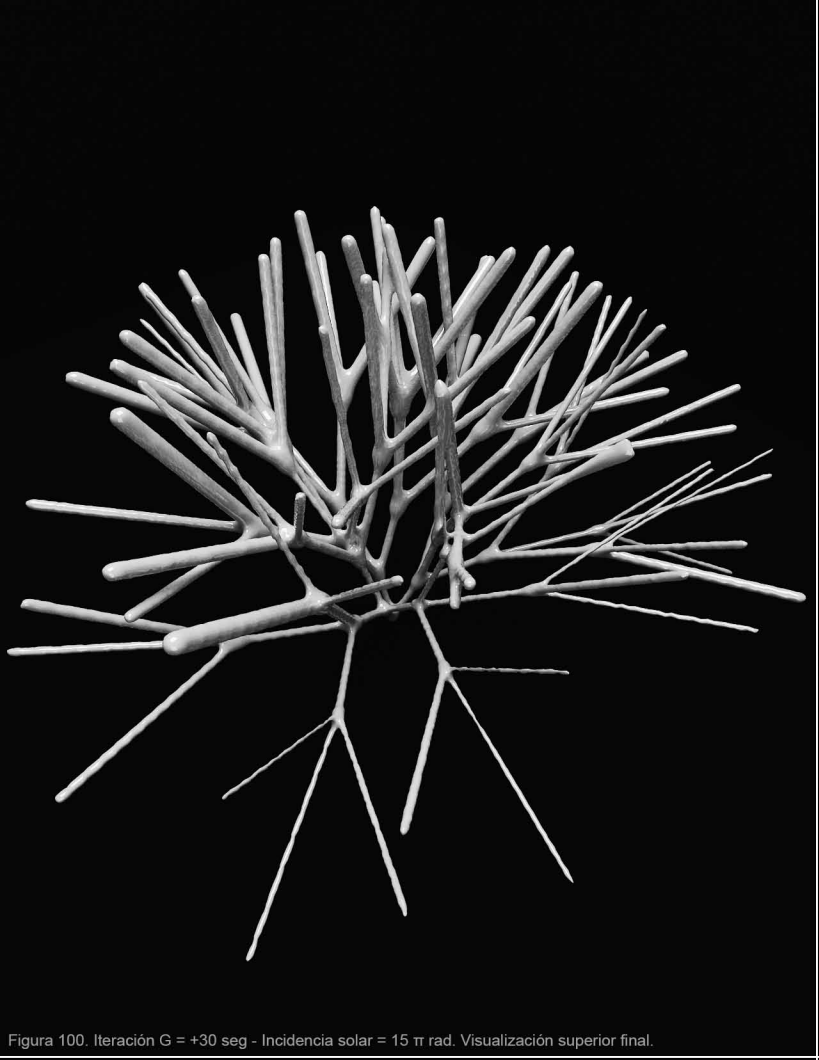


Figura 100. Iteración G = +30 seg - Incidencia solar = 15 π rad. Visualización superior final.

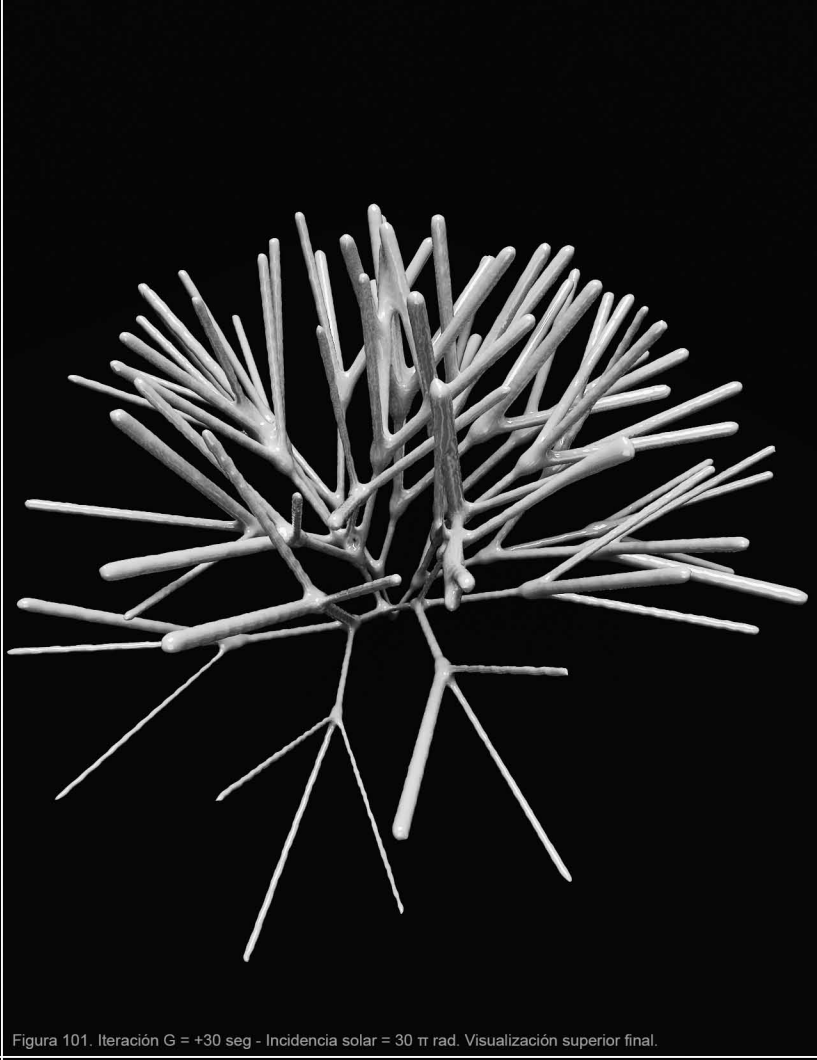


Figura 101. Iteración G = +30 seg - Incidencia solar = 30 π rad. Visualización superior final.

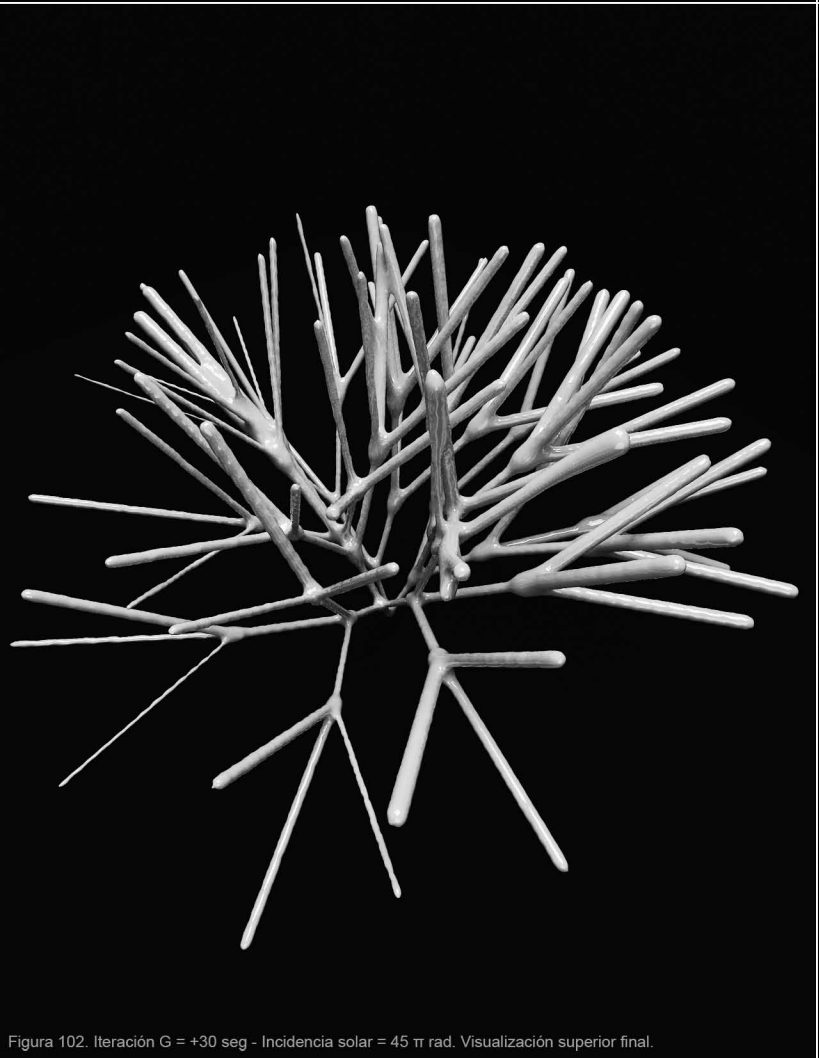


Figura 102. Iteración G = +30 seg - Incidencia solar = 45 π rad. Visualización superior final.

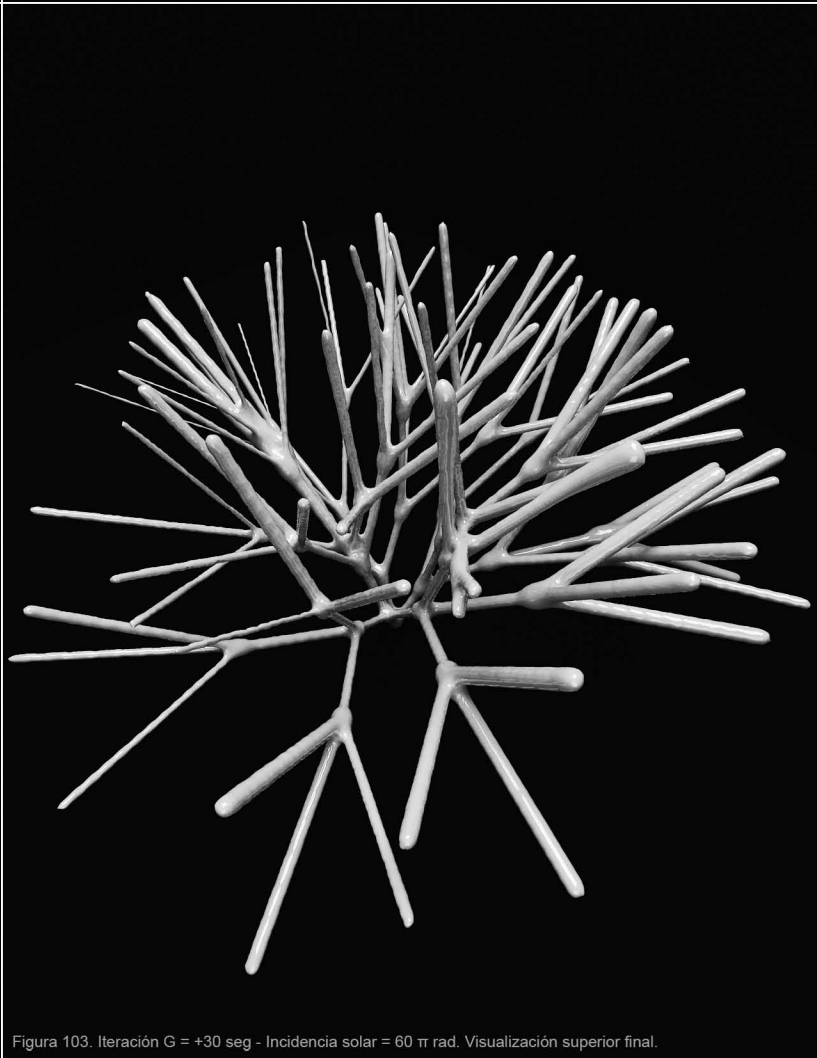


Figura 103. Iteración G = +30 seg - Incidencia solar = 60 π rad. Visualización superior final.

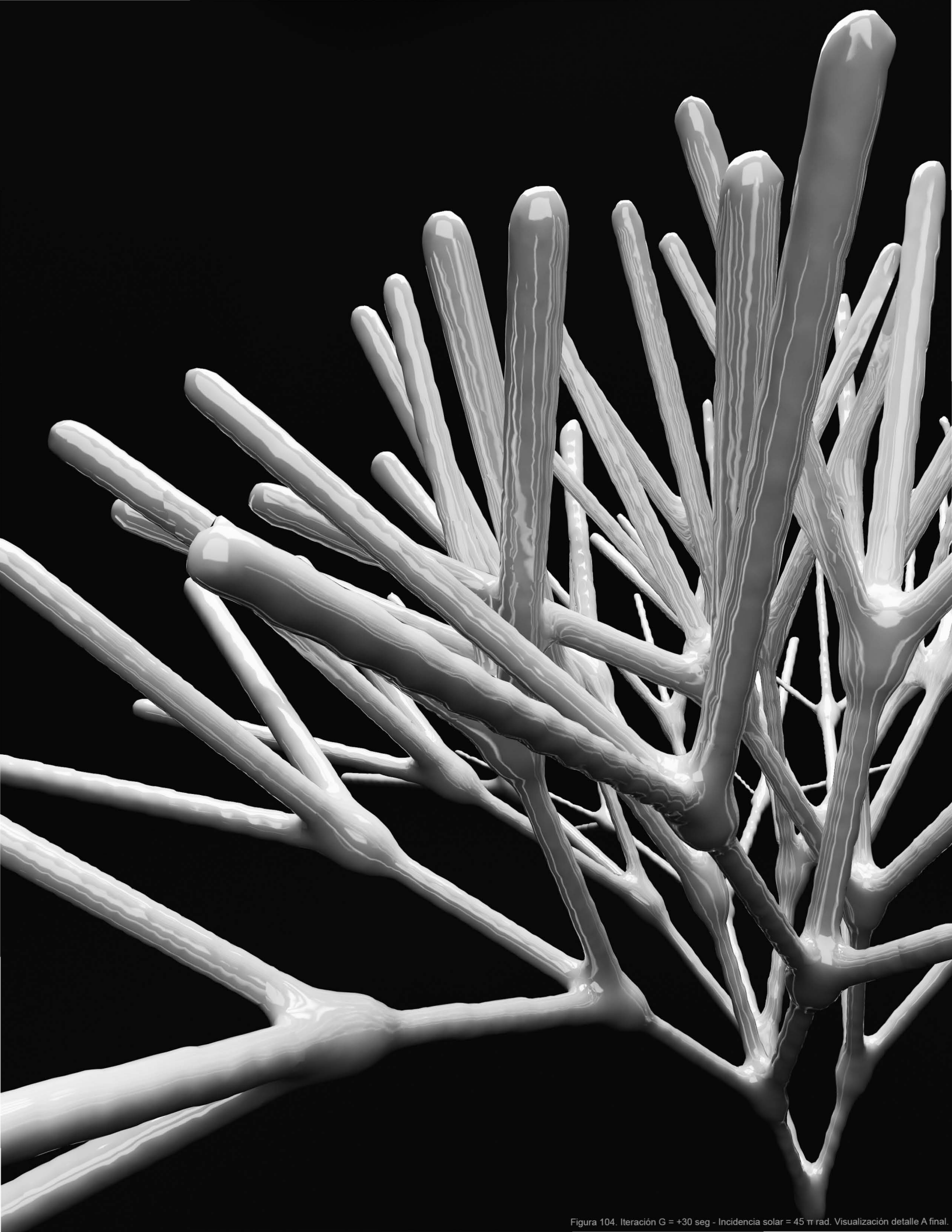


Figura 104. Iteración G = +30 seg - Incidencia solar = 45 π rad. Visualización detalle A final.

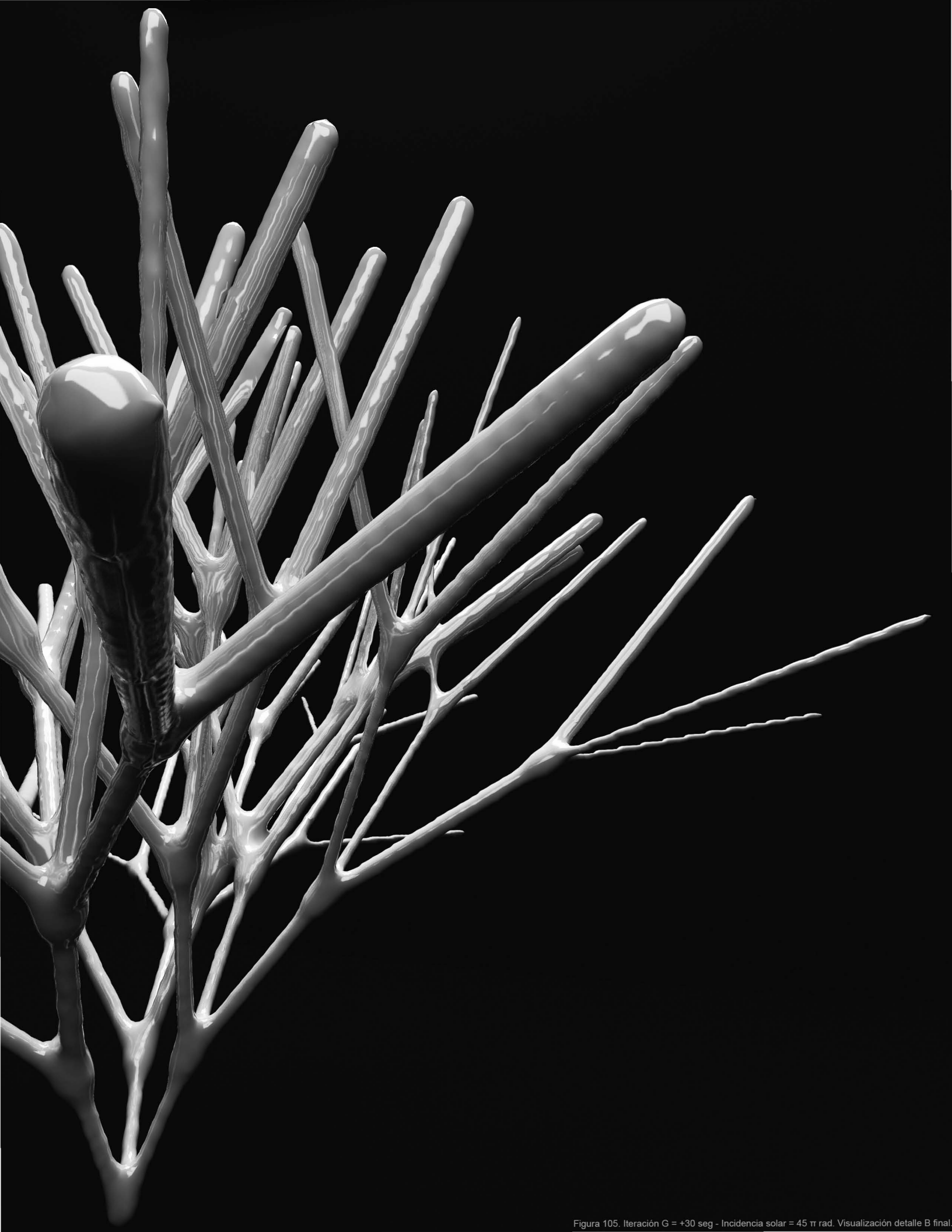


Figura 105. Iteración G = +30 seg - Incidencia solar = 45 π rad. Visualización detalle B final.

4.9 – Conclusiones

Actualmente existe una “pequeña conexión explícita” entre la práctica del modelado paramétrico y la práctica de la ingeniería del software aplicada al proceso del diseño arquitectónico, escribe Robert Woodbury (2010, 66). En la investigación desarrollada a través de este caso de estudio, hemos establecido tales conexiones a través de la exploración de cómo la ingeniería del software, y en específico, la programación lógica gráfica, puede ayudar a producir modelos paramétricos flexibles basados en las disciplinas biológicas, físicas, matemáticas y geométricas. De manera más específica, referimos cuatro conceptos y enfoques aplicados al entendimiento de un sistema de crecimiento ramificado fractal, relacionado a su proceso de generación inicial, la aplicación de fuerzas, la transformación geométrica y la representación de materia por medio de la optimización de mallas geométricas. Al usar una metodología interdisciplinaria y aplicada a un proceso de diseño específico, puede afectar al resultado geométrico y, por consiguiente, al formal, cuando una variable de generación depende de parámetros numéricos restringidos al entendimiento de la geometría presente en los organismos de la naturaleza.

En relación al nivel de complejidad obtenido durante el desarrollo de la gráfica *DAG*, producida en el caso de estudio de este capítulo, utilizaremos la ecuación de complejidad ciclomática de Thomas McCabe (páginas 31 – 31):

$$CC\ (G) = e - n + 2p$$

Dónde:

- *G*: la gráfica.
- *e*: número de segmentos.
- *n*: número de nodos.
- *p*: número de gráficas independientes.

$$CC\ (Logaritmo\ Gráfico\ Coral\ Acropora) = e\ (205) - n\ (123) + 2(1)$$

Asumiendo que p = 1, ya que todas las gráficas se relacionan para generar todo el sistema paramétricamente y ninguna es independiente, se simplifica en:

$$\mathbf{CC\ (Logaritmo\ Gráfico\ Coral\ Acropora)} = e\ (\mathbf{205}) + i\ (\mathbf{146}) + u\ (\mathbf{119}) - n\ (\mathbf{123})$$

Dónde:

- *i:* número de entradas.
- *u*: número de salidas.

$$\mathbf{CC\ (Logaritmo\ Gráfico\ Coral\ Acropora) = 347}$$

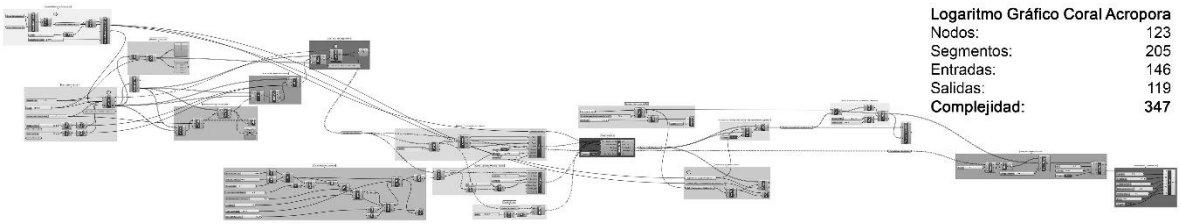


Figura 106. Gráfica *DAG* completa y final, causante de la generación del sistema en ramificación – fractal del coral Acropora, basado e interpretado en variables de control paramétrico biológicas, físicas, matemáticas y geométricas.

El nivel de complejidad alcanzado (347) para desarrollar la representación y simulación natural – digital de este organismo es media, dado las variables y conceptos interdisciplinarios aplicados al proceso de diseño. Esta complejidad, medida en datos cuantitativos es importante, ya que nos permite evaluar numéricamente un proceso de diseño complejo en su generación geométrica. Por otra parte, no se incorporaron todas las variables que dan origen al sistema orgánico natural, debido en parte, a la carencia de equipos computacionales capaces de analizar, cuantificar y simular relacionados, por ejemplo, con el PH y nivel de producción de materia a partir de la síntesis del alimento del organismo, durante el desarrollo de esta investigación; pero queda abierta a un desarrollo posterior más complejo, que permita evaluar de manera cuantitativa y cualitativa su proceso generativo a partir de un algoritmo de programación gráfica.

Implicaciones para la educación

El libro *The Software Engineering Body of Knowledge Version 1.0* (Hilburn; 1999, 20), no solo representa la práctica de conocimientos totales de los ingenieros, es también representativo de lo que los ingenieros pueden enseñar. Si las soluciones a problemas arquitectónicos e ingenieriles pueden resolverse en conjunto, y si el cuerpo de conocimiento de la ingeniería del software puede sugerir prácticas para resolverlos, la pregunta es: ¿Podrían los arquitectos ser aprendices de los conceptos de la ingeniería del software, cuando conozcan acerca del modelado paramétrico gráfico, aplicado a un proceso de diseño arquitectónico?

La enseñanza del modelado paramétrico gráfico, por lo general, pierde la referencia a la práctica de la ingeniería del software, lo que no resulta sorprendente, dado la falta de conexión entre el modelado paramétrico y la ingeniería del software en niveles de conocimientos más avanzados. Robert Aish (2005, 12) dice que tiene como objetivo conseguir que los diseñadores piensen de manera algorítmica “sin exigir que los diseñadores se conviertan en programadores”; y llega a sugerir que los diseñadores se beneficiarían de lo que es la antítesis de la ingeniería del software aplicado a sus procesos de diseños particulares o colectivos: una reducción en el “formalismo lógico” de los modelos paramétricos. Esta ambición es notoria en el modelado paramétrico en los entornos computacionales que Aish ha desarrollado mientras trabaja en Bentley y Autodesk. Cuando los arquitectos aprenden a utilizar estos entornos de trabajo, se les enseñan normalmente los comandos escritos a través del teclado, con el objetivo de lograr resolver tareas más específicas (Woodbury, 2010). En consecuencia, la típica pedagogía enfocada al modelado paramétrico sigue en las prácticas de la enseñanza del software CAD no paramétrico mucho más de lo que se sigue enseñando en las prácticas de programación y la ingeniería del software. Woodbury sostiene que la causa es una consecuencia que proviene de ser “diseñadores y programadores aficionados”, y naturalmente, con ganas de “salir de la abstracción, generalidad y reutilizar” todo de los programadores reales.

Esta investigación sugiere que puede ser inadecuada la enseñanza del modelado paramétrico gráfico sin el acompañamiento de conceptos relacionados con la ingeniería del software. Con el modelado paramétrico lógico – gráfico se simplifican las pulsaciones de las teclas en un teclado de la computadora, y aumentan los *clicks* para establecer las conexiones de las gráficas *DAG*. Independientemente de la causa, la consecuencia es que las gráficas *DAG* no estructuradas y organizadas no son comprensibles para la mayoría de los diseñadores. Esto puede ser una situación aceptable, si los diseñadores tratan de hacer una producción geométrica con motivo de la descripción y origen del aspecto formal, para un problema en específico. Sin embargo, las dificultades que implican realizar cambios y

relaciones interdisciplinarias de conceptos aplicados a un proceso de diseño, en los modelos paramétricos, indican que muchos diseñadores están generando modelos de soluciones inmediatas a problemas muy complejos. Los diseñadores desarrollan modelos paramétricos que cambian y evolucionan a lo largo de la duración del proyecto, y frecuentemente usan los modelos para tratar de solucionar un problema particular en un mismo modelo que captura toda la lógica de producción y representación de un proyecto. El verdadero peligro potencial, en relación al proceso de diseño arquitectónico, es que la falta de conocimiento relacionado con la ingeniería del software y la incorporación de conceptos de otras disciplinas, en lugar de hacer más fácil el modelado paramétrico, lo vuelve más difícil para la aplicación en la práctica.

La pregunta de cómo el modelado paramétrico debería de ser enseñado se encuentra abierta, y merece mayor atención por parte de los arquitectos. Esta investigación indica tentativamente que los diseñadores, y en específico, los arquitectos requieren una cierta comprensión de la ingeniería del software, así como de otras disciplinas del conocimiento, para ir más allá de los resultados meramente formales en respuesta a un problema arquitectónico particular, producidos por una herramienta que solamente soluciona aspectos relacionados con la representación de geometría. Como tal, puede haber un espectro más matizado en la identificación de los procesos de diseño, que hagan uso de este tipo de herramientas computacionales, sin embargo, se encuentra fuera del alcance de esta investigación. Especulo que la manera en la cual son instruidos los ingenieros del software, puede ser una conexión de muchas, que orienten la instrucción futura de los arquitectos.

Implicaciones en la práctica arquitectónica

Otra cuestión abierta implicada en esta investigación se refiere a la forma en la que el modelado paramétrico gráfico tendrá un impacto en los procesos de generación de las etapas de desarrollo en la arquitectura. A medida que la práctica de la ingeniería del software ha mejorado, el proceso ha cambiado radicalmente. Los programadores se encuentran organizados en jerarquías de conocimiento rígidas, para impulsar y desarrollar un proyecto a través de secuencias lineales flexibles de requerimientos, diseño, construcción y mantenimiento. Esta investigación sugiere una práctica alternativa en la que las iteraciones y la retroalimentación continua de conocimientos de otras disciplinas, permitan a los arquitectos y diseñadores aceptar y hacer uso del “cambio y la evolución progresiva” en los procesos de diseño, incluso de manera tardía. Los pequeños equipos de programadores se auto-organizan en espiral a través de las etapas de planificación, desarrollo y presentación del software, lo anterior sugiere que este tipo de procesos son ágiles en relación a los tiempos establecidos de producción, con un porcentaje mucho menor en tiempo de entrega y en sobrecostos (The Standish Group; 2012, 25).

El caso de estudio del coral Acropora en este capítulo, es una señal de cómo las manipulaciones de parámetros y variables, basados a partir de conceptos específicos de otras disciplinas, pueden afectar a la práctica arquitectónica, no por su resultado formal, sino por los procesos evolutivos, generadores y descriptivos. Este proceso de diseño comenzó a partir del entendimiento estructural, matemático y geométrico de un ciclo de ramificaciones, luego la exploración de variabilidad de forma mediante la aplicación de fuerzas basadas en conceptos de la física, la exclusión y diferenciación de materia generadora del organismo en función del desplazamiento y posición del sol y, de la cantidad de materia presente en cada iteración de ramificación representada por medio de la optimización computacional de mallas geométricas. Mediante el uso de un modelo

paramétrico, basado en una gráfica *DAG*, se determinó el resultado formal, a partir del entendimiento y aplicación de conceptos interdisciplinarios en este proceso de diseño.

En referencia al proceso de diseño, basado en el entendimiento y desarrollo a través de un modelo paramétrico de programación gráfico, este tipo de metodología puede aplicarse a proyectos arquitectónicos, en el desarrollo de elementos estructurales con propiedades de cargas puntuales y repartidas, por medio de elementos en disposición geométrica ramificados, no solamente aplicados al resultado formal final de la estructura, sino como una oportunidad de obtener variaciones de forma optimizadas, según la incidencia de las cargas y de la conformación propia del material generador, en relación a las propiedades físicas anisotrópicas y químicas propias de la materia (figura 107, 108 y 109).

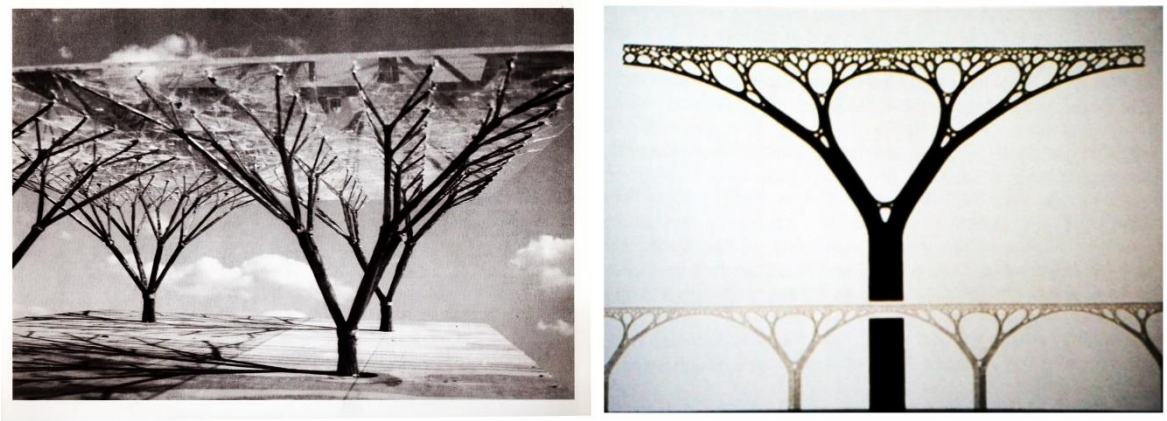


Figura 107. Proyecto “Estructuras de Árbol”, ILEK, Frei Otto y Bodo Rasch. Stuttgart, Alemania. 1960.

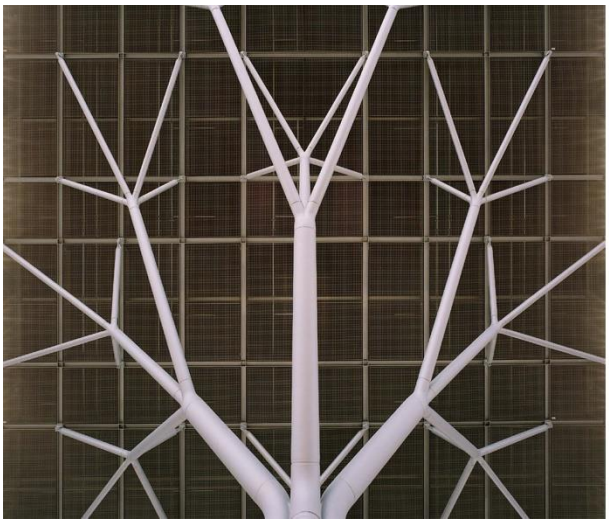


Figura 108. Estructura de soporte principal para cubierta de la Terminal 3 – Aeropuerto de Stuttgart. Schlaich Bergermann Architects and Engineers Partners. Stuttgart, Alemania. 2004.

Figura 109. Centro Nacional de Convenciones de Doha, Estructura Principal. Arata Isozaki. Doha, Qatar. 2011.

El modelado paramétrico, por otra parte, introduce una nueva estrategia de diseño, el aplazamiento, que consiste en cambiar el orden en que las decisiones de diseño y modelado computacional se pueden hacer, lo cual es una característica importante en las estrategias deliberadas a seguir durante el desarrollo del diseño y modelo paramétrico. De hecho, un argumento financiero sobresaliente es la capacidad de soportar cambios rápidos al final del proceso de diseño, únicamente por la modificación de variables o parámetros causantes del resultado final y basado en las capacidades de carga de la materia generadora de estructura y de la forma consecuente. La definición final de programación gráfica utilizada en este caso de estudio, demuestra cómo los modelos paramétricos pueden interpretar y aceptar cambios en las etapas finales de un proceso de diseño, en consecuencia, de la aplicación de fuerzas externas y variabilidad estructural – geométrica. Por otra parte, la flexibilidad es otro aspecto sobresaliente, referido al cambio en los modelos paramétricos, que puede ayudar a reducir el costo del cambio en las etapas finales de un proyecto arquitectónico. Esta investigación sugiere que la consecuencia de elevar el costo del cambio se extiende más allá de la capacidad de realizar cambio de última hora, la consecuencia real puede ser una práctica de diseño más interactiva y maleable en relación al modelado computacional tridimensional; y una práctica dónde el posicionamiento y las restricciones en el diseño no estén dictadas por los costos del cambio, sino más bien por las necesidades del proyecto.

El desarrollo y conclusión de esta metodología, aplicada al caso de estudio del coral del género Acropora, contribuyen a nuestro entendimiento del modelado paramétrico gráfico, a través de la relación que existe entre la ingeniería del software y las conexiones de varias disciplinas. Este argumento se basa en las relaciones que existen alrededor de los cambios, la investigación, los métodos y las prácticas de diseño “compartidas”. Estas conexiones posicionan a la ingeniería del software como un importante precedente y campo de estudio para arquitectos y diseñadores; una relación que tiene implicaciones en cómo es enseñado o si se enseña el modelado paramétrico en las escuelas de arquitectura como herramienta de solución a problemas arquitectónicos, en cómo el modelado paramétrico gráfico se puede integrar en la práctica diaria profesional del arquitecto y, en cómo es concebido el modelado paramétrico por cada persona involucrada en las disciplinas del diseño.

Capítulo 5 – Caso de estudio Lógico – Textual

"¿Es posible considerar a la arquitectura como una cosa que crece y cambia, o es capaz de moverse de un lugar a otro? ¿Se puede considerar al espacio como una cosa que incluye acción, movimiento, generación y cambio? A partir del hecho de que la arquitectura se mueve, involucraría necesariamente el concepto del tiempo. La arquitectura no es solo espacio; pienso que la expresión y la interpretación del tiempo inherente en la arquitectura es importante. Es un asunto de disolver lo fijo, lo lógico clasicista y la reordenación de la arquitectura"

- Arata Isozaki

Objetivo

En la actualidad, la mayoría de los seres humanos habitamos en ciudades, en algunos años más, casi toda la población humana del planeta vivirá en las ciudades. Debido a esto, la complejidad en redes de transporte, de comunicación y edificios aumentará, dando lugar a grandes sistemas de datos, que almacenarán toda esta información. Uno de los aspectos principales de toda ciudad es la movilidad. El desplazamiento que los seres humanos necesitamos realizar para llegar de un punto a otro, ya sea a través de caminar o utilizando un medio de transporte alterno; es un aspecto que puede ser llamado "flujo", que involucra cambio y movimiento constante, indeterminado a partir de los deseos y objetivos de los seres humanos y, al mismo tiempo, complejo y generador de la configuración de las ciudades, independientemente del medio natural en dónde se desarrollan; además de ser una característica sobresaliente de cualquier organismo biológico. Cada movimiento posee una serie de propiedades y es el resultado de las capacidades individuales de cada individuo o sistema.

Las propiedades del movimiento siempre son actuales, ya que involucra al concepto del tiempo presente, sin tiempo no hay desplazamiento. Por otra parte, la capacidad no es necesariamente actual si, por ejemplo, un ser humano u objeto no está en movimiento. Esto implica que las capacidades pueden ser reales sin ser actuales. El término técnico para esta condición ontológica es "virtual". Lo virtual no es opuesto a lo real, sino a lo actual. La realidad en lo virtual consiste en diferenciar elementos y relacionarlos entre sí, a partir de circunstancias singulares que los definen. La realidad virtual es una estructura, en la cual debemos de evitar aquellos elementos y relaciones que observamos en la realidad, como límites actuales, y generar nuevos límites a partir de la estructura virtual computacional.

En este caso de estudio, desarrollaremos una estructura virtual de una ciudad prototipo, representada mediante volúmenes random 3D, como límite actual. Como nuevo enfoque, circunstancia actual y límite, desarrollaremos una estructura virtual basada en los movimientos (flujos) *random* o indeterminados de los seres humanos en una estructura actual, como lo es una ciudad; el medio de generar y representar a tales sistemas, será un entorno de trabajo tridimensional y, las herramientas, el software *Rhinoceros 5.0* como visualizador de tales flujos, *Grasshopper* como instancia de representación de geometría y generación de datos y, por último, el lenguaje de programación *Python* (por medio de la incorporación del componente *GhPython* en *Grasshopper*), que nos permita controlar las interacciones como medios generativos iniciales, además de la representación inicial y final del nuevo sistema paramétrico y virtual.

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

5.1 - Trans

La palabra *Trans* connota cambio, conversión, ir más allá, comprender, atravesar e intercambiar. La dinámica de tales transformaciones físicas, psicológicas y espaciales, se encuentran en los procesos de conversión de elementos inmateriales, como el sonido, la luz y el movimiento en una forma material: el sonido llega a ser visible a través de una matriz tridimensional que representa ondas o vibraciones, la luz aparece representada a través de un espectro electromagnético, conformada por electrones que se dispersan en el espacio contenido, y los flujos, por medio representaciones vectoriales de movimiento. Este proceso también involucra la transformación y organización de la forma arquitectónica y la incidencia que tiene en el observador. Mejor dicho, este inicio es capaz de crear una experiencia teatral dinámica a través de un conjunto de luces y objetos en movimiento. El punto focal es el evento que está en constante cambio.

Así como la experiencia del teatro se basa en la observación directa a los actores en el escenario, diseñado para hacer estallar la imaginación del espectador, esta experiencia arquitectónica involucra al cuerpo y a la mente del participante, volviéndose fenomenológico y psicológico a la vez. La arquitectura y los espacios urbanos emergen solo dentro de un cuerpo y mente que los ha experimentado. La comprensión del teatro en los años de 1970, a través de las discotecas experimentales, incluye *Nine Evenings of Theater and Technology* en Nueva York (1966) y *Night-club Electric Circus* de Andy Warhol (1967 – 1971), que combinan proyecciones psicodélicas, música rock, danza y teatro; llegaron a ser tiempo después, la inspiración de conciertos que intensificaban y transformaban la percepción de sonido por medio de procesos electrónicos físicos, lumínicos y visuales.

Los procesos de transformación fundamentalmente buscan materializar lo que se puede o no observar, como las fuerzas o el tiempo, particularmente de un espacio. Esta noción, del tiempo, por ejemplo, es evidente en la vida y cultura del Japón, que puede ser visto a través de los altares *Himorogi*, delimitados por cuatro polos y rodeados por cuerdas (*shimenawa*) – en dónde los espíritus de la naturaleza (*kami*) – descienden según la creencia de las personas. Esta noción de *Trans*, sin embargo, engloba el mundo sacro y profano con el mundo físico y material.
Tal proceso puede ser observado en la matriz de proyecciones auditivas, lumínicas y visuales de la Plaza del Festival en la expo de Osaka de 1970, diseñada por el arquitecto japonés Arata Isozaki, en dónde las técnicas de movimiento y transformación de estas matrices generan ilusiones por medio de la eliminación del cuerpo del escenario entre los asientos, con el fin de crear una experiencia “híper-transformativa” (figura 110).

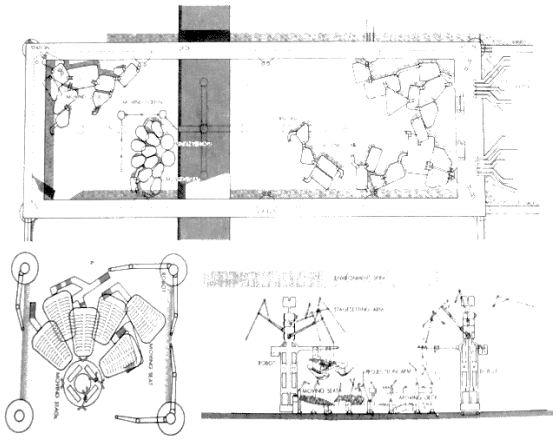


Figura 110. Festival Plaza, Expo ´70 Osaka, Japón. Monumento invisible, plano y elevación con escenario movible y asientos controlados y desplazados por brazos robóticos.

Trans, tal como un proceso de cambio físico y psicológico, está incorporado en un sentido más amplio en relación con el espacio "*contenido*", que implica una transición en el tiempo o un enlace entre países y culturas. Este concepto opera de dos modos, el de transformación *dentro* del espacio contenido de cada proyecto arquitectónico y *entre* los proyectos.

5.2 – Espacio desconocido: La estructura espacial de las ilusiones

5.2.1 - Ilusión

La luz hace de los espacios barrocos mágicos, los cuales expresan un constante conflicto al introducir la iluminación externa a manera de provocar asombro y misterio. La dualidad es vital para este tipo de espacios, ya que están basados en dos puntos focales y son contrapuntos tejidos entre la luz y la oscuridad. Pueden estar llenos de formas muy complejas y son transformados por la introducción de luz y la generación de sombras a partir de los diversos ángulos de penetración. La distribución de iluminación intensifica la forma y estimula la percepción de los espacios por sí solos.

Los flujos y las formas mezcladas de los espacios barrocos no son la intención final. Mejor dicho, son naturalmente desarrollados a partir de constantes esfuerzos para producir ilusiones espaciales, y naturalmente, el elemento para unificar armoniosamente a esos efectos es la luz.

En todas las instancias de este aspecto, la luz es agrupada y transportada a dónde el diseñador desea. Los espacios son ensamblados a partir de los principios de los flujos, cargas y disposición de la luz. Al poder impresionar a las personas con la entrada de luz, el diseñador transforma los espacios. Para poder realizar las transformaciones y poder observar el "drama" que la luz provoca, las formas espaciales también deben de ser transformadas.

5.2.2 – Control por códigos

En una ciudad existen un sinfín de automóviles miniatura con personas encerradas en una jaula confinada, esos automóviles están dotados de una manija inútil que los conductores deben sujetar cuando el semáforo indica que pueden avanzar. Los autos sorpresivamente se desplazan en todas las direcciones posibles. Colisionan, se separan y vuelven a colisionar. Los conductores ocupan un espacio con un límite fijo de tiempo, en dónde una serie de sucesos indefinidos ocurren. Como las moléculas en una sustancia gaseosa, los vehículos miniatura colisionan uno con otro a partir de un movimiento no determinado y cada colisión propicia otra.

Al pensar que el espacio es un fenómeno que ocurre dentro de la mente humana, ¿Cuál es la naturaleza de la imagen espacial percibida por el operador del vehículo? El piloto de un avión que vuela en la noche o en un clima malo debe apoyarse enteramente de instrumentos computacionales para localizar su posición en el espacio. Incapaz de confiar en su propia percepción, debe depositar su confianza en interfaces computacionales, las cuáles son un código específico en el espacio. Los espacios modernos, especialmente los urbanos, están compuestos de una multitud de señales. Por ejemplo, las señales de tránsito transmiten eventos y sistemas específicos, sin ellas, el conductor no tiene información y confianza sobre cómo llegar a un punto o detenerse en otro. Las extensas redes de códigos

para la transmisión de comunicaciones cubren a nuestras ciudades y son indispensables para poder interpretar los espacios urbanos.

Los espacios dominados por este tipo de señales son abstractos a partir de la realidad y podrían ser llamados espacios controlados por signos codificados, generan complejidades muy altas y se entrelazan a los que ya existen. Las ilusiones en los espacios dominados por los signos y códigos, y no por la luz, deben surgir cuando la disposición de la rutina se interrumpe por una serie de eventos erráticos. La ilusión está relacionada esencialmente con los espacios urbanos modernos; para comprender el espacio urbano, debemos usar signos y códigos para producir y representar una entidad ilusoria espacial llevada al extremo por cualquier medio.

5.2.3 – Esquema espacial

Las ilusiones que aparecen sobre la ruptura de nuestras estructuras normales de percepción, se presentan de manera más efectiva a través del análisis deliberado de las características espaciales. Para llegar a interpretar al espacio como una ilusión, podemos generar dos tipos, la primera es creada por cosas como los espejos y la luz, la segunda, generada por códigos y disoluciones de tales códigos. Las condiciones actuales en relación a los espacios arquitectónicos, demandan ir más allá del desarrollo de este enfoque. Para tal motivo de producción, se propone el siguiente esquema conceptual.

En este esquema hipotético, el eje central es el espacio tridimensional, una forma de entender al espacio a partir los estudios de Descartes, pero no el único; que es el aspecto primario de las percepciones normales humanas. En un eje se encuentran los espacios que contienen las imágenes relacionadas con lo desconocido – eso es, los espacios asociados con la profundidad psicológica y los deseos -. En el otro eje se encuentran los códigos y los signos relacionados con la abstracción y los espacios multidimensionales de ilusión.

A partir del punto de vista del observador humano, el encuentro con el espacio desconocido es un proceso de investigación en las profundidades de su propia consciencia personal. En el espacio de ilusión, el ser humano se compromete a desarrollar un proceso analítico y teórico; el mundo pluralista de flujos complejos presentes en el espacio urbano.

El aspecto espacial central de este esquema hipotético es el tridimensional ($x + y + z$), el espacio actual que es el sustento de todas las percepciones humanas normales. Restringido al eje z, el sentido vertical al cual los seres humanos atribuimos lo desconocido (inferior) y a la ilusión, a lo conocido (superior). En el polo del eje [z-] está un orden espacial relacionado con lo desconocido – que son las imágenes conectadas con la profundidad psicológica, la magia y los deseos -. En el polo opuesto de ese eje [z+] se encuentra un sistema espacial relacionado con la imagen de la ilusión, que son códigos de relación, abstractos y multidimensionales. Las imágenes de lo desconocido y la ilusión, sugieren unas condiciones extremas espaciales, que lidiarán con el espacio urbano (ciudad), la creación artificial más grande de hábitat del ser humano, capaz de lidiar con estas condiciones y ser el medio de generación perceptiva - formal (figura 112).

A través de varios modos y medios de percepción, esos espacios se manifestarán por sí mismos y en consecuencia, deberán de ser analizados de manera concreta e individualmente. La amplificación del carácter, referido al aspecto formal y espacial, en condiciones extremas, revela el alto rango de complejidad en los problemas y es inherente a las contradicciones durante el proceso de diseño por parte del arquitecto.

Si “lo desconocido genera ilusión”, tal como afirma Jun'ichirō Tanizaki (1888 – 1965), este poder de generación no estará limitado al pasado, sino que irá más allá, siendo relevante en la actualidad como espacios contemporáneos que podrían también, causar ilusión futura. Sin embargo, en el presente, no estamos seguros de si estos dos polos representen condiciones extremas que se encuentren lejos de las percepciones humanas normales. Ellas podrían estar, quizá, conectadas de una manera aún no comprendida, que no puede ofrecer hipótesis alguna para explicar cómo podría ser.

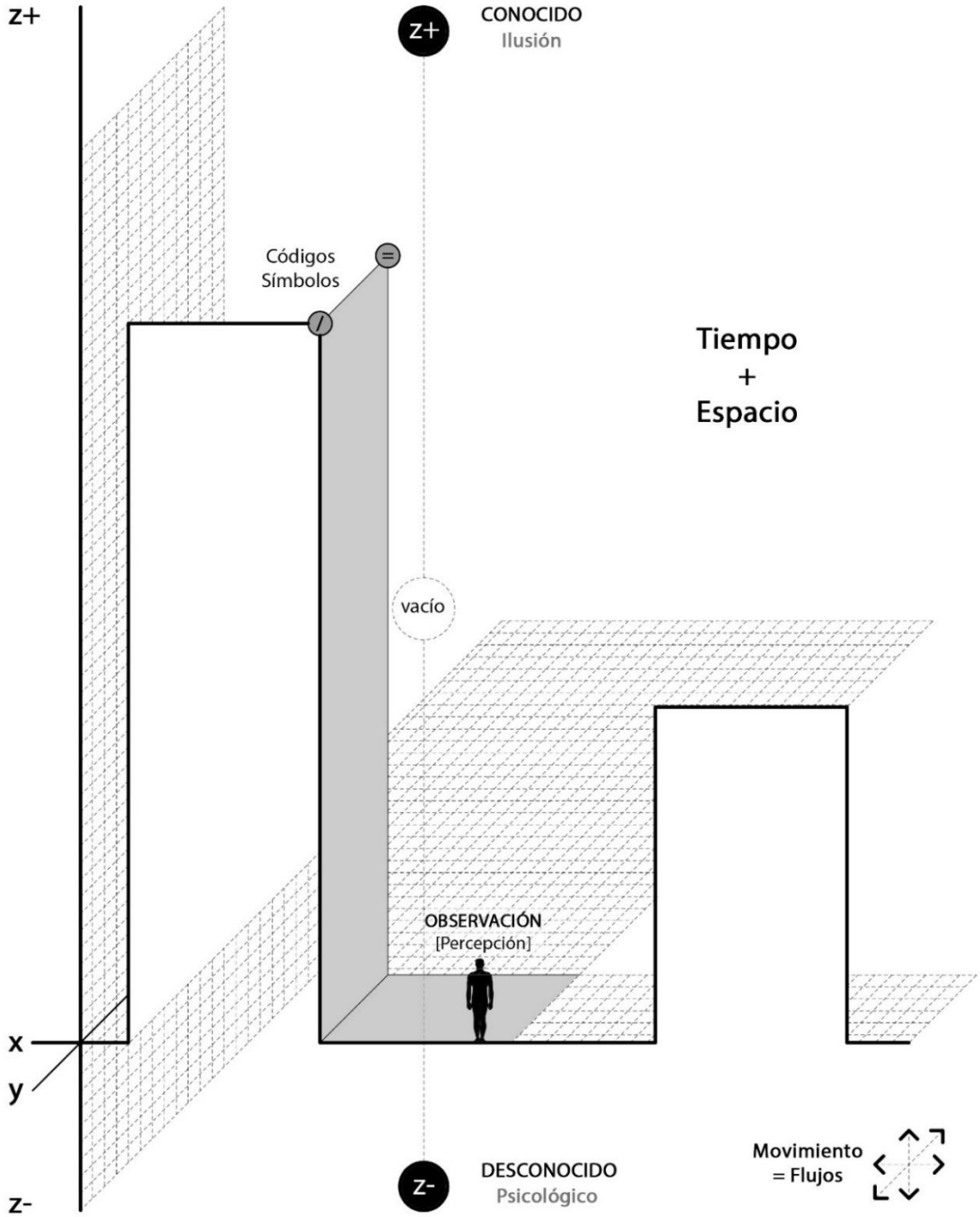


Figura 111. Esquema conceptual-espacial inicial.

5.2.4 - Vacío

En las culturas orientales, y en específico, en la cultura japonesa, los conceptos de espacio y tiempo han estado simultáneamente comprendidos por medio de una palabra que puede describir los dos conceptos, "*ma*". *Ma* es definido literalmente como "el intervalo natural entre dos o más cosas que existen en una continuidad", o "el vacío entre dos cosas, una abertura; el espacio comprendido entre dos columnas o pantallas plegadas" o "la pausa natural o intervalo en el cual un fenómeno se genera a través del tiempo". Por comparación la palabra "*ma*" no se diferencia bastante entre la compresión de las culturas occidentales en los términos que involucra, tales como el espacio y tiempo. Ambos son descritos, espacio y tiempo, a través de una noción de intervalo. Esta utilización es la base para la comprensión de todas las esferas del ambiente, la vida y el arte, con el fin de extenderlo y representarlo en la arquitectura, las bellas artes, la música, el teatro y los jardines en Japón, que son conocidos comúnmente como el "arte *Ma*".

Este entendimiento sin diferencias en los términos de tiempo y espacio, puede ser considerado como un fundamento importante para la emergencia de diferencias en las expresiones artísticas tanto de oriente como de occidente. Todavía la palabra "*ma*" se resiste a establecer una coincidencia extraña con teorías actuales que la igualan en significado con los términos de tiempo y espacio.

Las percepciones preliminares del espacio estuvieron basadas en la articulación visual de la naturaleza. Esto estuvo invariablemente restringido a la atadura impuesta por la era y los diferentes puntos de vista, relacionados con la naturaleza y el cosmos. La percepción del espacio es más probable que surgió a partir de un esfuerzo para visualizar y formalizar a las divinidades de la naturaleza (*kami*) con sus comportamientos únicos, que fueron pensados para permanecer en el universo. El control y generador de las divinidades fue el sol, que articula intensamente el tiempo y el espacio a partir de su desplazamiento y su posición respecto al planeta Tierra. Esta presencia fue establecida a partir de la dualidad entre lo positivo y lo negativo, el día y la noche, la luz y la oscuridad, el mundo divino y el inframundo, la construcción y la destrucción.

El espacio que se produjo fue fundamentalmente blanco. Cada uno de los objetos sólidos se diseñaron para ser vacíos por dentro y capaces de recibir el *kami* que desciende en ciertos momentos para llenar tales espacios con los espíritus naturales (*reikon*). La representación de este momento de inhabitabilidad llega a ser el principal aspecto de muchas representaciones artísticas de este país. El espacio es percibido a través de los eventos que aparecen en el lapso del tiempo. Se puede decir que el espacio es generado y reconocido como tal, solo a través de la meditación en el tiempo.

Este es el único aspecto, relacionado con la percepción de la forma, en la cual Japón se diferencia del resto del mundo, referido a los términos de tiempo y espacio. Si se toma esto al extremo, el concepto del espacio-tiempo en la cultura occidental dio lugar a imágenes fijas de una continuidad infinita y homogénea, tal como lo presentó Descartes, que llegó a ser la base de las concepciones modernas. Sin embargo, el espacio y el tiempo en Japón estuvieron omnipresentes y eran partes intrínsecamente relacionadas. En una condición caótica y mezclada, el espacio no podría ser percibido independientemente del tiempo. De igual manera, el tiempo no era un concepto abstracto sino un regulador, un aspecto que permitía la fluidez y que podía ser utilizado para establecer la relación del movimiento en los espacios estáticos.

5.3 – Corpóreo – Real e Híper

5.3.1 – Lo desconocido y la ilusión

Los espacios arquitectónicos que comenzamos a diseñar están limitados en su inicio por los “espacios teleológicos”[61] y el concepto de tiempo – espacio; en el texto “Espacio, Tiempo y Arquitectura” de Sigfried Giedion (1941), la idea del espacio en cuatro dimensiones no trasciende, ya que es una versión popularizada del principio de la relatividad espacial de Einstein. Utilizaremos una frase oximorónica, “el espacio tridimensional actual”, ya que los espacios que los seres humanos percibimos se extienden más allá del espacio teleológico, con el cual, los arquitectos trabajan, pensando que tales limitaciones son incomprensibles y no habría manera de diseñar tales espacios, sino al romper con tales restricciones.

En un viaje realizado, recuerdo el momento en el cual un rayo de luz penetró a través de una abertura en el domo de una iglesia. Esto sucedió en Europa, pero me recordó un momento triste y parecido, cuando observaba la luz de la luna en una galería de un templo antiguo en la ciudad de Kioto, y sentí miedo de despertar enfrente de las sombras oscuras de las piedras del jardín. Fue solo un rayo de luz en medio de las sombras. Las formas han sido borradas y los contornos no eran tan claros. No obstante, sentí que ambas situaciones sucedieron en el interior de un espacio arquitectónico. El mundo que percibimos los seres humanos puede llegar a ser translúcido, lo que produce un nuevo espacio urbano ilusorio.

Ambas experiencias duraron solo un instante. Lo que fue una revelación para mí, es el hecho de que he experimentado cosas a través de mi cuerpo y mente, aprendiendo que las ciudades y la arquitectura aparecieron como momentos transitorios. Y estaré dispuesto, a partir de ahora, dedicar mi vida a ellos. El “espacio desconocido” es un intento de describir a la arquitectura y a las ciudades en esta manera. Podría ser imposible de comunicar a otras personas, sin embargo, las percepciones de las personas están basadas simplemente en la memoria. Incluso los espacios urbanos percibidos normalmente brindan imágenes diferentes según cada individuo. El hecho de la variación de imágenes y significados depende de la sensibilidad, la experiencia, el interés y la condición física, que muestra que solo podemos experimentar el espacio a través de un filtro en la percepción individual. Los espacios arquitectónicos y urbanos emergen solo dentro de un cuerpo que los ha experimentado.

5.3.2 – Espacio fenomenológico

Claude Merleau-Ponty ha propuesto con anterioridad una explicación fenomenológica para este concepto. El espacio de lo desconocido no refleja su teoría completamente. En lugar de eso, investigaremos una manera de asimilar lo que parece diametralmente opuesto a los tipos de espacio, descritos por lo desconocido y la ilusión dentro de un método propio. Existen imágenes relacionadas con la psicología, los deseos, la magia y los símbolos que aparecen en las profundidades del cuerpo y mente del receptor, y que descienden verticalmente. Esto podría ser llamado espacios intensivos. Luego existen las imágenes relacionadas con los códigos, con la abstracción y lo multidimensional, que se encuentran en la escala y ejes verticales urbanos.

61 La Teleología es la rama de la metafísica que se refiere al estudio de los fines o propósitos de algún objeto o algún ser vivo, o literalmente, a la doctrina filosófica de las causas finales.

Ellas atacan a receptor a partir de su salida y entrada, manifestándose cuando están forzadas a esparcirse energéticamente. Esto podría ser llamado espacios extensivos. Ambos conjuntos de espacios existen con certeza y ocurren repentinamente. La pregunta es si un enfoque teórico puede enlazarlos en un tiempo específico para poder ser construidos, permitiendo a los profesionales en las sociedades humanas, nombrados arquitectos y urbanistas, expresarlos y representarlos. Ambos espacios podrían estar conectados de alguna manera que aún no hemos observado, pero no podemos ofrecer alguna hipótesis para explicar cómo podría ser. El interés en la representación del espacio se enfoca en tres áreas separadas: el espacio desconocido, el espacio normal y el espacio de la ilusión. Subsecuentemente, la explicación de las tres condiciones está concentrada y limitada por un filtro que es creado por la disolución del aspecto multidimensional. Esto es también un intento de descubrir una manera para enlazar tres áreas separadas en imágenes enlazadas que generar la respectiva representación.

5.3.3 – La división entre lo desconocido y la ilusión

El interés es determinar si las percepciones humanas pueden o no ser llamadas “híper”, generadas por una gran escala, y si podrían ser desarrolladas a través de la tecnología, de manera más específica, por medio de un lenguaje de programación textual. Lo desconocido penetra en nuestro interior a través de la incertidumbre, que naturalmente necesita a la memoria del ser humano, al pasado depositado en nuestro cerebro. Las peculiaridades físicas arraigadas a través de los hábitos de cada individuo a lo largo de su vida, así como un grupo de memoria impresa en los genes serán expuestas. Los grupos de teatro inhabitan las grietas de la ciudad, utilizando los movimientos físicos como método de expresión de significados, porque creen que el cuerpo solitario es el canal a través de cual, lo desconocido puede encontrarse y enlazarse con la realidad.

Es precisamente lo que sucede en las plazas o espacios libres de las ciudades, se crean ilusiones que pueden llegar a existir o no. La incursión de la tecnología actual propicia que esos espacios libres puedan ser ilusorios, desconocidos, multidimensionales y con varios puntos focales de generación espacial. Por lo anterior, los efectos de luces generados en los rascacielos de Hong Kong por las noches, se superponen a los lugares que los generan, y se transforman en algo elusivo. Un rango de visión ordinario llega a ser trascendente. Pero aún está lejos de ser sublime, una cualidad romántica del siglo XIX que describe un fenómeno extraordinario. Lo sublime es algo absoluto que nos influye profundamente sin poder resistir la impresión de tal acto. La alucinación, la levitación y la involución son generadas dentro del cuerpo y mente de los seres humanos. El espacio exterior se transforma primero, es una percepción humana instantánea; el siguiente instante ahora ha cambiado. Diferentes imágenes enteras aparecen sobre un periodo extenso de tiempo. Por eso las híperpercepciones son también percepciones diarias.

Las ciudades han llegado a ser un escenario para las expresiones con propósitos políticos. Las demostraciones radicales en las calles y las actuaciones teatrales en las grietas de las ciudades son similarmente políticas. Lo desconocido es ahora lo ilusorio y real.

5.3.4 – La transformación de los límites

Las recesiones económicas mundiales y las crisis energéticas de años anteriores han impedido, de cierta manera, el desarrollo de proyectos arquitectónicos complejos dependientes de la tecnología. Cada proyecto, sobresaliente tecnológicamente, se encuentra normalmente en oposición al lugar en dónde se localiza. Este evento inesperado y contradictorio podría ser, quizá, considerado como un deseo terco para introducir anomalías y establecer diferencias, generando de este modo disimilación. Por lo anterior, los proyectos tecnológicamente concluidos ocupan un punto medio entre dos categorías opuestas. Los espacios culturales, por ejemplo, son categorizados, y se puede argumentar que cumplen una función de acuerdo a su oposición: alto contra bajo, museo contra diversión, este contra oeste, *avant-garde* contra pop. Es interesante cómo los sucesos pueden ocurrir sobre una línea divisoria de categorías que, eventualmente, toman lugar en los límites entre una y otra (figura 113).

Figura 112. Vista superior de la Kunsthaus Graz, Austria. Arquitecto Peter Cook. 2003.

Los seres humanos estamos cubiertos por luces, sonidos e imágenes. Las personas no pueden estar ocultas en una zona segura de estos aspectos; con el fin de percibir más que solo una imagen, algo que podría sentir en el interior y a través de su piel. La imagen podría no ser fija, sino estar en movimiento según la percepción individual. Y podría amplificar el ritmo cardiaco por medio de la experiencia momentánea. Lo anterior podría ser llamado visual, argumentando que el arte en tiempos antiguos ha sido totalmente una experiencia física, pero más allá de eso, las percepciones actuales son nombradas progresivamente en función del poder tecnológico que las propicia, es decir, un torbellino ilusorio. Sumergido en este torbellino, se podría perder el equilibrio, se podría flotar y extasiar. Dentro de un espacio, por ejemplo, de una discoteca, la tecnología es usada para producir flujos de luz y sonidos en gran escala, con el propósito impactar la percepción humana y producir ilusiones resultantes en imágenes interiores no definidas. Una explicación podría ser que actúan y tienen un efecto tal en el cuerpo, que penetran en los órganos de la percepción, y los sentidos, que forzosamente crean híperilusiones.

Un espacio con un alto grado de desconocimiento propicia, por consiguiente, un espacio con un alto grado de híperilusión, y pueden ser descritos como *espacio literario* y *espacio dramático* respectivamente. En el espacio literario, la escritura por medio de palabras, propicia progresivamente imágenes e imaginación. En el espacio dramático, la escritura

propicia la formación de imágenes a partir de las profundidades de lo desconocido y torbellinos de luz. El medio de expresión son los parámetros, que al mismo tiempo los definen. Los espacios anteriores propician la formación de imágenes, también, a partir y dentro del interior de la percepción de cada individuo. Actualmente, sin embargo, estas imágenes deben ser forzadas a aparecer por medio de las palabras, códigos y experiencias físico-psicológicas; lo cual sugiere una necesidad de nuevas maneras para producir imágenes. Lo que es claro, es que las imágenes llegan a ser representaciones visuales de cosas que las palabras o las experiencias humanas deliberadamente generan. Las imágenes están influidas y limitadas por los métodos de escritura de las palabras (gramática) o por los hábitos y las experiencias de cada ser humano. Por esto, la modalidad del lugar en que se encuentra una persona se convierte en un factor decisivo. Los signos que definen los comportamientos humanos en una ciudad, por ejemplo, establecen la comunicación del tiempo y trazan la memoria pasada con nuevos elementos transitorios, proporcionado un carácter especial a la modalidad del espacio, en el cual, el cuerpo humano es envuelto.

5.3.5 – La modalidad del espacio

Los dos ejemplos de espacio anteriores en cualquier caso traen implícitos el aspecto de las memorias y experiencias humanas del pasado, consecuentes de la percepción individual presente de cada persona. Por el contrario del teatro, que está diseñado para crear nuevas percepciones a partir de la aparición de viejas memorias. Incluso si iniciamos en una hoja en blanco, el espacio arquitectónico todavía tiene la necesidad de estar basado en la compresión colectiva del arquitecto con los usuarios eventuales o finales. Eso es porque tiene que ser un lugar para actuaciones futuras. En el pasado, la estructura, la función y los materiales han sido lo que las palabras y significados son para el espacio literario y dramático, en términos de espacio arquitectónico individual o colectivo, es decir, está definido a partir de la interpretación de experiencias y deseos de nuevas ilusiones del usuario final. Cada espacio arquitectónico es una forma modelada, la cual es el origen y base inicial. De manera colectiva, esos modelos también son relacionados con tipologías arquitectónicas, que operan también, como metáforas abstractas. Al concebir o iniciar el diseño de un edificio, son útiles como parámetros provisionales. El diseño puede comenzar al elegir un modelo o tipología, pero al mismo tiempo la condición de espacio-tiempo debe ser tomada en cuenta; aquí también lo desconocido y lo ilusorio se relacionan.

5.3.6 – Contenedores en las ciudades

Las ciudades han llegado a ser como el teatro en incontables épocas. Aquí también, las metáforas dramaticales son utilizadas. La plaza es un escenario, es un lugar en el cual los ojos de las personas están atentos. Estos dos contenedores están colocados dentro de la plaza; las superficies que dejan fluir la luz están, deliberadamente, vacías. En este lugar, la piel es dura y opaca en cada situación. La membrana exterior es la cubierta y el piso; y es tratado lo más delicadamente posible porque es parte de su superficie curva tridimensional. En cada caso, cada objeto inarticulado construido en base a una escala urbana, es un escenario para el teatro y escena principal, llamada ciudad.

El contenedor cerrado trasciende a la escala humana, lo que hace que nuestras ciudades dinámicas parezcan muelles en dónde un enorme barco está atracado, o una plataforma de lanzamiento en dónde una nave espacial está suspendida. El objeto, que incrementa la escala de todo lo que se encuentra alrededor de él, generará en nuestra visión, así como cuando nos desplazamos a través de la ciudad, una hípercondición. El sentido de

discontinuidad con la escala humana, producirá sensaciones que trascienden al tacto y generan una hípercondición. El objeto no es tangible, se generará dentro de cada cuerpo como una imagen diferente. Un espacio ilusorio y visual nuevamente emerge.

En resumen, las condiciones espaciales generadas por lo desconocido y la ilusión, serán detectadas a través del cuerpo y mente del individuo, y emergerán como imágenes internas dentro de un contenedor que ha sido generado artificialmente en escala urbana. A partir de un punto de vista intensivo, el espacio se relacionará con la profundidad de los deseos y aspectos psicológicos del ser humano; y a partir de un punto de vista extensivo, con lo multidimensional e ilusorio. Solo como una consciencia orientada hacia lo híperreceptivo.

5.4 – Ciudad In [Visible]

5.4.1 – La complejidad de los mapas exactos

Los cartógrafos actuales basan su trabajo en fotografías aéreas de las ciudades, que son codificadas y transformadas en versiones gráficas similares a los planos arquitectónicos. El principio de operación básico es interpretar las características arquitectónicas y de los terrenos respectivamente, en imágenes. En ambos sistemas, el ojo del observador se focaliza sobre lo exterior del objeto observado. En términos de ciudad, las vistas aéreas son llamadas “vista a ojo de pájaro”, que representan y son la analogía de la escena de la vista propia de un pájaro volando sobre la ciudad. Esto es posible gracias a la invención las máquinas que pueden volar, creadas por el ser humano.

La vista de “ojo de pájaro” es útil solo para mejorar las impresiones conseguidas respecto a la conformación de una ciudad, así como de aquellos puntos focales o más importantes. Las formas - imágenes aéreas de las ciudades y las experiencias espaciales dentro de los edificios siempre están relacionadas. Aunque existen diferencias mínimas, en general las escenas y comportamientos que dominan en las vistas a partir del suelo, también dominan en aquellas a partir del aire. Al estar en el último piso del edificio más alto de cualquier ciudad, la visión es más prominente a partir de la ventaja del punto de observación del pájaro, al igual que un humano que esté en la calle y aprecie la gran escala de la ciudad contenedora. Las formas urbanas y las estructuras de las actividades que toman acción en cada lugar coinciden de manera simultánea. Las representaciones de las ciudades y las composiciones espaciales urbanas están siempre relacionadas.

5.4.2 – La desaparición de la forma en las ciudades

Las ciudades modernas tienden a expandirse fuera de los límites establecidos. Un incontable número de carreteras se expanden al igual que el número de edificios. Lo dinámico y lo estático se expande por igual. Es muy complicado poder generar una imagen clara y comprensiva de las ciudades, ya sea desde el piso o el aire. Con las densidades de población en constante aumento, las ciudades se expanden a una constante e incremental velocidad. Cada persona depende de un medio de transporte para poder desplazarse por la totalidad de las ciudades, y los individuos que no puedan utilizar un medio de transporte, sufrirán, pues las condiciones físicas de los humanos no permiten recorrer a pie estas enormes distancias. Las ciudades podrían convertirse en ruinas colosales. Los habitantes podrían vivir, incluso en espacios solo separados por el diámetro de su cabello ante las crecientes densidades. Las vistas a partir del suelo y en el aire se relacionan ahora de manera más explícita. Las ciudades fueron generadas para tener formas distintivas, ya sea

para defensa o por planeamiento, continuidad física tridimensional y actividades integradas; pero esos aspectos son externos e invisible en algunas ciudades actuales.

Si la palabra “desaparición” es adecuada, algunas de las ciudades actuales ejemplifican la destrucción de conceptos comunes aceptados anteriormente, caracterizados por la alta tensión que gobierna las composiciones físicas de los espacios urbanos, comprendidos por medio de puntos de vista particulares. Esto es más obvio, por ejemplo, en el suelo que en el aire. En las ciudades, todo está en constante movimiento y en constante cambio.

5.4.3 – Espacio control-código

El valor de una torre con un reloj, la fachada de un palacio, un monumento o una pieza de escultura que alguna vez reinó sobre una ciudad determinada ha decaído, correspondiente a la ciudad que adquirió y acepto la “fluidez moderna”. En las ciudades medievales, la plaza central, la iglesia o el palacio de gobierno dominaban la totalidad de la composición de la ciudad, que eran los símbolos y puntos de encuentro más importantes de la comunidad. Las plazas de los mercados de este tipo de ciudades, no eran contempladas para ser guías visuales y puntos de orientación, sino espacios de intercambio y flujos comerciales transitorios, que llegaron a convertirse en referentes importantes. El movimiento definió la importancia de un espacio por las actividades y no por los objetos estáticos.

Para un conductor que maneja un automóvil a 200 km/hr, el paisaje urbano no es más que imágenes que aparecen en un intervalo de tiempo. Sin embargo, no es posible comprender al espacio como un aspecto meramente visual. Los medidores de distancias absolutas no existen. Solo las relaciones mutuas entre objetos son perceptibles. En los espacios urbanos, solo las relaciones tienen significados. Los objetos de la percepción humana son interpretados de manera fragmentada en un espacio físico no continuo.

La transición a partir del punto de vista de un peatón a la del operador de una máquina necesita una seria revisión en nuestras concepciones referidas al espacio. Esto disminuye a su vez el significado de la visión humana; que son las formas exteriores de objetos materiales que nos rodean. El paisaje que un piloto percibe, por ejemplo, se extiende según el movimiento, pero esta característica deriva a partir de la medida de la distancia y los kilómetros recorridos y no de la presencia del observatorio material relacionado con los significados de las formas que lo conforma físicamente. En otras palabras, la extensión del paisaje es trasladado a códigos y producen efecto en la percepción del observador, entre relaciones de objetos y un observador subjetivo. Consecuentemente, para un piloto que vuela en la noche, todo el espacio es condensado en diagramas y mapas localizados enfrente de él, privándolo de cualquier sensación perceptual, sin elección de poder desplazarse hacia donde él desea, es decir, su forma de percepción fue transformada.

Los códigos y señales son por sí mismos coordenadas. La manera en la cual están distribuidos, representa a cada parte del objeto al cual pertenecen y son códigos espacialmente aislados, pero relacionados entre sí.

En el futuro, las vastas y megaestructuras arquitectónicas y los edificios multifuncionales serán difíciles de definir y diseñar, a partir de un solo nivel de comprensión, ya que aparecerán en distintas ciudades y culturas de todo el mundo. Todavía, a causa de su vasta escala de complejidad, son carentes debido a su significado e interpretación de necesidades, ya sea para generar o regenerar un balance adecuado en su configuración. Por ejemplo, serán capaces de producir el tejido urbano de una ciudad como una entidad

unitaria, se convertirán en decoración, que no puede proveer un marco de referencia en términos de relaciones proporcionales, pero puede simular la vida interior orientada según la escala. La ciudad llegará a ser más como un caleidoscopio de colores recurrentes e infinitos, de refracciones y reflejos en constante movimiento en constante evolución. El espacio urbano es un tejido de patrones abstractos basados en códigos colectivos.

Una vez que comience este proceso lejos de las capas de las limitaciones, la alineación de elementos basados en códigos a partir de objetos actuales, no podrá ser soportada. Por ejemplo, los anuncios proliferan sin ningún límite, siendo intermediarios en casi todos los tipos de expresiones que son representadas a través de las pantallas, luces neón, publicaciones digitales, televisión, entre otros; las señales que emiten por medio de los códigos pueden ser localizadas físicamente, pero los anuncios atraen señales sin límites dispersadas a través del espacio urbano e instantáneamente, cubren la totalidad de las luces encendidas en las ciudades de acuerdo a un plan, regido por códigos.

El espacio urbano es sumergido en una compleja red de medios electrónicos. Los elementos codificados, que inicialmente se colocaron junto a objetos físicos, son ahora separados y fluyen libremente en el espacio-tiempo. Por consecuencia, los grupos de código no dependen ahora de los objetos físicos estáticos, al contrario, los objetos físicos ahora dependen de los flujos de los códigos, que definen el cambio a partir del movimiento, a partir del tiempo. Es la única manera de comprender el espacio urbano actual. Figurativamente, los códigos se relevan por sí mismos, solo sobre espacios sensitivos y dinámicos.

Nuestro ambiente urbano es imperceptible al sentido de la vista humana. Las luces, los sonidos, las señales, el transporte, el movimiento de los objetos, códigos visibles alrededor de nosotros y las otras percepciones a través de nuestros cinco sentidos se involucran y relacionan, provocando algo que puede ser llamado “una sensación compleja interior – táctil”. Si todo lo que sucede en la ciudad nos involucra y se define solo a través de la percepción de nuestros cinco sentidos o experiencias, la ciudad sería comprensible solo para las personas que están en constante movimiento en el interior de la red urbana. Naturalmente, es imposible generar una imagen total de lo que la ciudad significa, por ejemplo, por medio de mapas, que son estadísticas aproximadas de distancias. El diseño urbano deberá ser reorganizado tomando como base la consideración los elementos “invisibles” anteriores. Aún estamos en el inicio de adopción de este enfoque.

5.4.4 – Confianza solo en el proceso

El constante movimiento, la difusión, el rechazo a imágenes fijas y el incremento infinito de anuncios y ruido son parte de la vida diaria en las ciudades más grandes del mundo. Actualmente, los entornos urbanos continúan incrementándose con gran rapidez. Explicado en términos de transformación axial según el tiempo, la ciudad es un líquido en constante reproducción orgánica y división. Los límites de ciudad no están tan unidos, tal que la transformación de un aspecto necesita de la destrucción de la unidad de otros aspectos, tan débiles en unión, ya que todos los elementos tienen, la mayoría de las veces, características similares que los hacen conjuntos en sí. En los procesos de generación del universo, aquello que carece de certeza en las imágenes y la comprensión, referida a la física cuántica, la máxima entropía, el calor y la energía, es incapaz de producir algo nuevo.

Hemos enfocado nuestra observación a aquellos fragmentos en las ciudades que involucran procesos de flujos. Estos fragmentos cambian constantemente en fases diferentes. En una ciudad, dónde el exterior se mueve aparentemente y cambia sin parar, estos procesos son confiables. El diseño, o el planeamiento de cualquier tipo pueden ser llamados como "procesos teóricos". La realización de cualquier tipo de decisiones, en cualquier punto en el tiempo, es un equivalente a la ruptura de una imagen de un objeto o suceso en movimiento. Incluso algo fijo, como un proyecto construido de arquitectura, cuando es colocado en la ciudad cambiante, es parte del crecimiento, del cambio, de la evolución artificial. Todos los métodos de diseño postulan una comprensión o una decisión en un punto determinado del tiempo para poder llevar a cabo el proyecto. Por lo tanto, todos están relacionados con actos concretos. El arquitecto deberá intentar prever, a partir del presente, la forma final del objeto que está diseñado. La última condición es el punto de inicio, al cual todas las cosas regresan.

5.4.5 – La ciudad como estructura virtual

Ciertos enfoques son necesarios con el propósito de dar una estructura virtual a una forma actual. Uno de ellos es la abstracción; que es, la creación de un nuevo concepto urbano generado a partir del aspecto cognoscitivo de un ser humano, empleando una herramienta digital computacional y referida a la programación lógica textual. Los pensamientos y razonamientos detrás, deben ser discontinuos y traicionar la continuidad tradicional. Los patrones totales que producirán esos códigos generadores, deberán ser análogos al razonamiento del cerebro humano y a las entidades orgánicas en constante evolución y movimiento, es decir, es algo parecido a una estructura urbana, interpretado a través de flujos y objetos estáticos, controlado por parámetros computacionales y representados por medio de geometría, generada a partir de códigos análogos.

Los elementos de las composiciones espaciales, deberán ser reducidos, por ejemplo, a códigos computacionales, y su respectiva relación entre sí, con el fin de ejecutarse en un sistema computacional y poder generar geometría, una imagen representativa. La computadora es transformada en una herramienta e investigadora de alto rango. Absolutamente, todas las distancias podrán desaparecer, así como los elementos estáticos se convierten en unidades de generación y de medidas temporales. En relación a ser representado en términos tridimensionales, es decir, de perspectiva, el espacio será un esquema codificado y comprimido geométricamente. El ojo del observador aún no se ha encendido para aceptar posiciones radicales y externas como absolutas, pero está dispuesto a analizar, comprender y ser pluralista. Tan pronto como sea sistematizado, las operaciones podrán ser llevadas vinculadas a cualquier equipo computacional, en esta etapa, la ciudad comenzará a ser reorganizada como un sistema modelo. Probablemente la cibernética podrá ser la base en el futuro de esta reorganización.

Antes de comenzar a desarrollar el caso de estudio, aspecto principal de este capítulo, es pertinente hacer una comparación entre la condición tecnológica actual y la de los años de 1970, la cumbre de los planeamientos urbano modernos. Durante los años transcurridos, hubo cambios absolutos que modificaron la apariencia de lo que *"era"* antes de nosotros. El rápido desarrollo tecnológico actual, ha alterado el planeamiento urbano reciente y los enfoques de desarrollo conceptual. Uno de esos enfoques, por ejemplo, se propició en el siglo XX, a través de proyecto "*Computer Aided City*", del arquitecto japonés Arata Isozaki (figura 114).

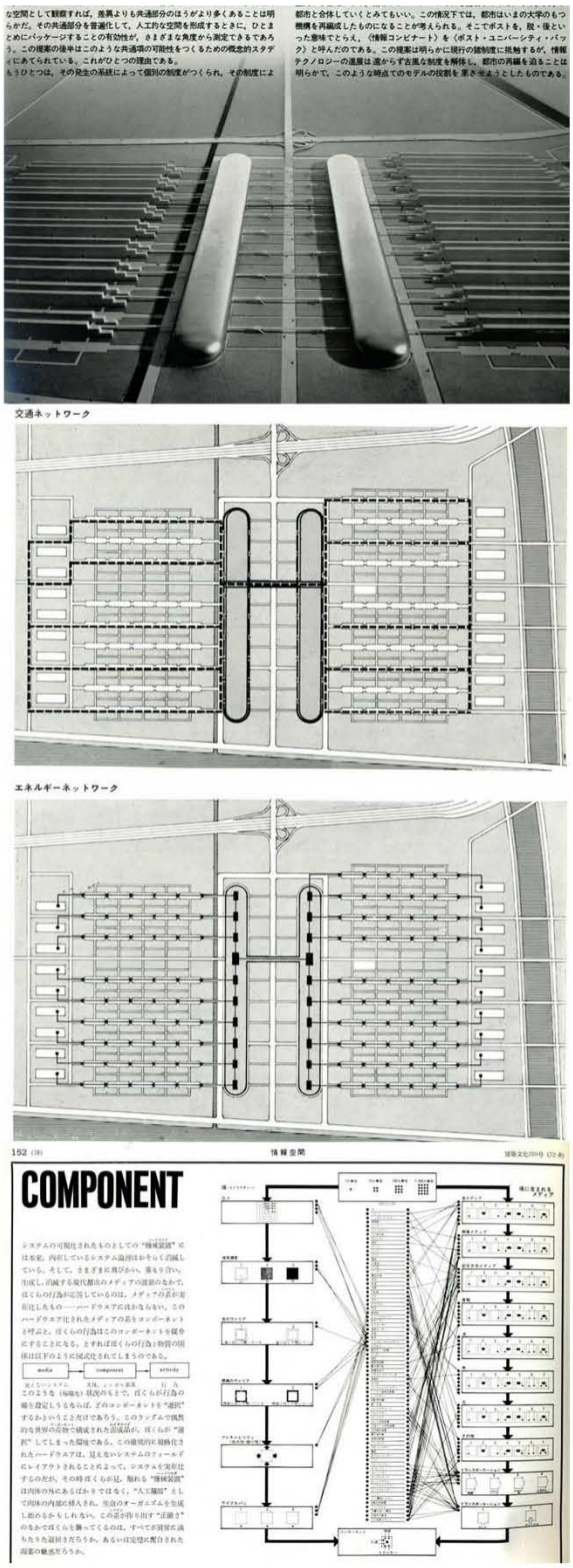


Figura 113. *Computer Aided City*. En esta ciudad, un gran ordenador central determina el funcionamiento de la misma a modo de un gran cerebro. El arquitecto genera una serie de catálogos de opciones computarizadas, que van desde carteles luminosos de publicidad, proyectores, *kits* de aislamiento, módulos educacionales, entre otros. Toda la ciudad se une a través de ferrocarriles rápidos con un gran centro. Arata Isozaki. 1972. Fuente: Kenchiku Bunka [1972] Nº310 agosto.

5.4.5 - De la coordinación a la simulación

En las etapas iniciales de la mecanización urbana, muchos arquitectos y visionarios produjeron planes urbanísticos. A principios del siglo XIX, todos los elementos y máquinas que se encuentran en las ciudades actuales habían sido introducidos, tales como las vías de trenes, los automóviles, el ascensor y los caminos para peatones, es decir, las baquetas; algunos, todavía en fase experimental. Todas estas propuestas, desde las más fantásticas a las innovaciones prácticas, fueron avances, y cada uno de ellos produjo inspiración a generaciones posteriores, por ejemplo, la mecánica. Los avances en la mecánica produjeron avances en la conceptualización de nuevos planteamientos de urbanización, ofreciendo nuevas abstracciones teóricas a través de la representación de imágenes en circunstancias particulares, es decir, la abstracción teórica se desarrolló en paralelo con las visiones urbanas futuras.

La metáfora de la máquina involucra el análisis de un todo a través de las partes que lo componen, asignando a cada parte una función para, posteriormente, ser ensambladas. Lo que puede ser aplicado a principios de composición en ciudades, que son controladas también, por funciones. El proceso de diseño, entonces, puede ser concentrado de manera unitaria a partir de la organización y coordinación de elementos que sean producidos y ensamblados a partir de una teoría.

La tecnología digital actual, por otra parte, nos brinda la posibilidad de poder controlar cada función a través de la simulación logarítmica computacional. Los elementos de las composiciones urbanas pueden ser simulados de manera individual o colectiva por medio de una serie de parámetros de control, con el propósito de comprender la conclusión formal, en base a la combinación de los sistemas invisibles generadores, tales como las funciones.

La simulación es el resultado de la combinación de funciones, que representan a un gran conjunto de sistemas abstractos que proceden de condiciones hipotéticas, al comparar un modelo digital con la realidad. La construcción actual de ciudades virtuales a través de la simulación, es completamente diferente que en los años de 1920. Ahora, esos planes utópicos e imprácticos pueden ser producidos por medio de procesos sistemáticos computacionales; siempre y cuando un modelo es incompatible e irreconocible con la realidad urbana y con la teoría generadora, este modelo puede ser expresado y representado a través de la simulación computacional. Esto significa que, a través de las computadoras, las posibilidades de nuevos planteamientos urbano-teóricos, son posibles de realizar y representar con los flujos implícitos en cada espacio urbano actual. Bajo estas condiciones, la consideración de la totalidad, la producción de un orden unitario visual y formal pierde importancia. En este proceso de expresar ideas en modelos que puedan ser manipulados de manera paramétrica, el arquitecto actúa como un piloto, que no debe balancearse y desviarse de sus conceptos durante el trayecto, con el fin de llegar a su objetivo, lidiando con un constante juego entre la realidad y la hipótesis.

5.4.6 – Ambiente cibernético

En la actualidad, en la época de generar formas urbanas digitales, existe un rango de responsabilidad por parte de los arquitectos y urbanistas, ya el diseño debe de expandirse más allá de las limitaciones pragmáticas impuestas en las escuelas de arquitectura, contemplando los campos de la tecnología aplicados a los sistemas de generación invisibles, en los cuáles, la ciencia computacional y el planteamiento urbano convergen. A partir de este punto de inicio, es claro que tales condiciones pueden ser esquematizadas

hipotéticamente a partir de teorías que las generen, convirtiéndose en propiedades de trabajo en el diseño.

En términos más concisos, el espacio urbano (un concepto extremadamente abstracto), es un lugar en dónde los seres humanos interactúan en un entorno artificial creado por ellos mismos, un estilo de vida actual en dónde la aplicación y el uso de los dispositivos electrónicos toma un papel relevante, no solo como medios de comunicación, sino como control de las actividades humanas.

Este ambiente es un espacio para la transmisión de varios procesos de comunicación e información que sustenta la cultura actual, algo que involucra de manera instantánea a los seres humanos a través de las conexiones de sistemas que cambian y se desplazan constantemente con fines específicos y controlados, y que dan lugar a las grandes redes de información. En este enfoque, el concepto de ambiente revela un potencial para generar experimentos urbanos, basados en componentes intangibles de las ciudades.

En lo personal, la ciudad moderna es un nodo de flujos y objetos invisibles, un laberinto dónde actualmente es posible bloquear las cosas que fluyen en direcciones determinadas a través del control computacional-mecánico. Es imposible todavía saber la conclusión formal que puede tener, como es la de conocer el futuro al mirar y comparar con un espejo objetos antiguos y actuales. El conocimiento de conceptos relacionados con la cibernética es un buen acercamiento para tratar de entender la capacidad de los seres humanos al controlar su ambiente y transmitir esta capacidad en términos de control mecánico y computacional a objetos. El espacio por sí mismo puede ser llamado ambiente cibernético, adoptando una actitud de control humano en algunos aspectos que nos rodean, y que genera los ambientes controlados e internos de las ciudades.

5.4.7 – La utopía está muerta

A mediados del siglo XX, las ciudades habían cambiado en apariencia. Había comenzado una expansión sin límites. Las capitales del mundo comenzaron a no tener límites claros y continuaron expandiéndose, los nuevos medios de transporte fueron introducidos y cualquier muro límite fue derribado. El interés de los habitantes cambió al nombrar a las capitales como metrópolis. Aquí existen, entonces, tres patrones urbanos de control, lo lineal, lo radial y la red de flujos–relaciones, capaces de soportar el cambio constante. Cada una es un sistema abierto que puede expandirse sin límites. La idea de planear el desarrollo completo de una ciudad fue abandonada; la planeación había sido reducida al control de los medios de transporte y a la zonificación. La silueta urbana se generó a partir de una cadena de eventos indeterminados y relacionados entre sí.

El combate entre la imaginación y lo racional hace un manifiesto entre los paradigmas y los métodos. Ambos procesos son una crítica a las diversas teorías del planeamiento urbano clásico y moderno. La realidad está aún muy alejada de la teoría. Cedric Price formuló un manifiesto referido a lo anterior, escribir sobre cualquier utopía es un acto de decadencia espiritual, por lo tanto, la planeación urbana es un acto criminal. La discusión sobre una utopía aquí termina. Finalmente, Price sugiere una regresión a un estado de anarquía denominada “no planeamiento”. Se puede asumir que la generación y la extensión de las ciudades son como un parásito que se reproduce para conformar algo no determinado. Este acercamiento asume la introducción de algunos conceptos importantes relacionados a la formación de las ciudades contemporáneas, a través de la reinterpretación, la transferencia,

la transformación y la regresión continúa. No hay diseño, ya que la lógica de la planeación ha sido descartada, se podría decir que es un nuevo tipo de urbanismo no controlado.

5.4.8 – El gran número

La representación de las ciudades se puede decir que es un paradigma, ya que fue generada a partir de establecer parámetros de control numéricos, que determinan el tamaño de, por ejemplo, un terreno en la ciudad. Por otra parte, las personas conforman comunidades cerradas en las cuales una conoce a la otra, pero pueden expandirse al conocer a otra comunidad y conformar un sistema más grande, tanto en relaciones de comunicación como en tamaño de hábitat, produciendo complejidad. Lo anterior puede ser definido de diversas maneras, pero lo que es un hecho es que las sociedades humanas están compuestas por un gran número de personas, viviendo en ciudades e impactando directamente a través de la formación de comunidades. Ambos aspectos se han desarrollado por medio de paradigmas que involucran una relación directa con los avances en la tecnología.

Los grandes números no son compatibles con la individualidad, ya que deben de lidiar en conjunto con unidades específicas de información, dónde cada unidad es representada por un número, por un parámetro. Los sistemas de zonificación urbanos están basados en una estructura similar. La idea de una imagen total unificada es abandonada, en su lugar, el sistema es dividido en distritos que son clasificados de acuerdo a características impuestas en conformidad con patrones específicos de uso de suelo. Lo anterior también es un ejemplo al lidiar con los grandes números. Los habitantes son convertidos en números para su estudio y la estandarización de la vivienda es sustituida por la del hábitat individual. Todos los ejemplos anteriores son sistemas que tienen que relacionarse con los grandes números, con la alta densidad de seres humanos habitando en este planeta.

A lo largo del siglo XX, los grandes números fueron utilizados para limitar y detener el incremento no controlado referido al tamaño de las ciudades, con el fin de generar homogeneidad y detener a los posibles sistemas expansivos. Con el paso del tiempo, estos patrones fueron cambiando en olas espontáneas de crecimiento que llegaron a ser invisibles, creando una estructura que enlazaba un sistema con control y otro sin control en un objeto unitario denominado ciudad. Aquí también, los grandes números no se relacionaron con la individualidad, sino que representaban a los grandes grupos que adquirían significado a partir de las imágenes generadas por ellos mismos. Esta es la manera por la cual la representación ha llegado a ser un paradigma.

Tal y como el concepto que denominamos “planeación” y que controla nuestro pensamiento, la representación de la “modernidad” actual controla nuestra visión, creando estandarización no solo en los objetos que producimos, sino en nuestros comportamientos y en las estructuras de nuestras sociedades. Por ejemplo, la colonización y la globalización, son la representación del desarrollo de situaciones específicas y compartidas. Sin embargo, tal representación es solo un significado de nuestra cognición de un objeto que es invisible. Lo intangible puede ser identificado; las metáforas sugieren que puede ser una máquina, un órgano, un efecto mariposa o un torbellino, lo que sugiere que existen cambios que ocurren cada momento, y que se desarrollan sin la intervención de nadie. Esto es, el mundo está siendo convertido en una gran concentración de flujos y partículas en movimiento que se esparcen en una densa neblina.

5.5 – Flujo

Los mundos paralelos producen una matriz. Los tiempos múltiples y entrelazados fluyen a través del espacio. El mundo está ensamblado tal y como el espacio-tiempo lo están, el aspecto principal de la ciencia ficción. No es una simple metáfora, sino algo que actualmente fluye debajo del mundo real en el que vivimos. Repentinamente, el equilibro es alterado y una línea diagonal se desarrolla; siendo también continua al escenario familiar del espacio-tiempo, generado en un modelo del mundo real-digital.

Actualmente, los ambientes urbanos continúan incrementándose. En términos de transformación, la ciudad se encuentra en un estado líquido y en constante división y reproducción orgánica. Tal como el matemático Norbert Wiener (1894 – 1964) explicó, la ciudad está lejos de tener límites impuestos, ya que la transformación de un aspecto involucra la destrucción de la unidad de otros aspectos, tan débilmente unidos que todas las partes pueden ser iguales. Wiener describió al mundo como procesos, un mundo que carece de la comprensión humana de la física cuántica a través de imágenes, y que la última entropía o extinción propiciará un estado en el cual nuevas cosas pueden generarse.

Los fragmentos de una ciudad están en constante cambio, y en el interior de fases. En las ciudades contemporáneas, en dónde las apariencias exteriores se mueven y cambian constantemente sin cesar, estos procesos son relevantes. El diseño, o este tipo de planeación, pueden ser llamados procesos teorizados. La toma de una decisión en cualquier punto del tiempo es equivalente a la totalidad de una imagen de un objeto en movimiento. De manera similar, un objeto inerte, como una obra de arquitectura, es parte de una ciudad que se transforma, es parte de su crecimiento, de su cambio, de su metabolismo. La mayoría de los métodos de diseño postulan la comprensión de una circunstancia revelada, es decir, una decisión en un punto específico del tiempo. Por consiguiente, todos están relacionados con actos específicos. La última condición es el punto de origen, al cual todas las cosas regresan.

Flujo es un concepto relacionado con el movimiento, tal como una concepción de dinámica relacionada con la mutabilidad, con la incertidumbre, la trascendencia y con lo no permanente. La arquitectura vista desde este punto de vista, es un cuerpo en movimiento que siempre está cambiando e incrementando el ritmo de su desplazamiento. Los flujos dibujan la representación a partir de la velocidad de los medios de transporte, como los autos, los aviones y los trenes de alta velocidad, así como el movimiento constante de los humanos a través de los laberintos estáticos de la ciudad. Los flujos son representados por medio de las tecnologías actuales y desarrollos en la informática, generando ambientes estructurados y de control programados para materializarse. En otras palabras, los avances en las tecnologías digitales y computacionales han brindado significados concretos a ideas que podrían ser descartadas en el pasado, simplemente por no poder ser representadas.

El flujo, en el sentido más amplio de la palabra, busca convertirse en una arquitectura emancipadora, una búsqueda que conduzca a la liberación a partir de contextos urbanos estáticos y movimientos en dirección a un horizonte dónde los pasados no representados puedan converger con el presente, y ser representados con las herramientas tecnológicas actuales. La fluidez puede ser contemplada tanto en formas arquitectónicas simples y complejas. Tal y como podría observarse en el flujo de un río o en los procesos estocásticos del movimiento Browniano. La noción de una arquitectura con una forma completa y contemplando a todos sus elementos en constante cambio, moviéndose y variable, contradice el paradigma clásico de Vitrubio de *firmitas* (solidez); así como las fuerzas de la

naturaleza, a través de la contradicción del movimiento del agua (dinámica) y su solidificación (estática); o la madera de un árbol que es elástica, pero que al morir se fosiliza hasta convertirse en piedra. En términos formales, el flujo implica algo más que una simple extensión volumétrica o un cambio de estado, es una nueva posibilidad de interpretación aplicado a un proceso de diseño computacional y digital.

5.6 – Enfoques aplicados al proceso de diseño

Los enfoques conceptuales iniciales en la arquitectura se representan a través de dibujos bidimensionales, que son las bases de su proceso de construcción física. Los arquitectos no construyen, los arquitectos proyectan y dirigen; por ello la traducción de un dibujo a una construcción real es siempre problemática.

La representación de un boceto en una superficie de papel nos permite interpretar de manera inmediata una representación espacial-bidimensional, la cual es una condición para la exploración sin límites del dibujo como medio de expresión. Pero qué sucede cuando este medio es superado por las variantes más complejas que son generadas a partir de teorías y parámetros complejos y radicales, entonces se hace uso de los medios digitales capaces de representar la tridimensionalidad más compleja. Aun así, esta característica puede quedar exenta al interpretar grandes datos de información y, sobre todo, a los grandes números; es aquí donde la programación logarítmica textual puede servir como herramienta de compilación y ejecución de tales características en representaciones digitales en tercera dimensión, permitiendo observar lo [In]visible como, por ejemplo, el movimiento y fluidez de los seres humanos.

Hegel describió que lo “nuevo” siempre consumirá a su predecesor, definiéndolo como opuesto, manteniéndolo como la sombra de un cuerpo cubierto por la luz. Y estas sombras siempre ocultarán a sombras anteriores, por lo tanto, las innovaciones culturales y tecnológicas solo pueden ser apreciadas e identificadas por los profesionales interesados en el tema. Tal apreciación es, por lo tanto, relativa, es clasificada y es un acto infinito; lo que es esencial para la cultura de la arquitectura, con el propósito de proponer y generar nuevos enfoques de diseño arquitectónico, como ideas científicas y métodos que puedan acumular conocimiento en base a la experimentación.

Por lo anterior, he definido cuatro enfoques que servirán como generadores teóricos y, al mismo tiempo, como mecanismos que, al ser combinados con el medio computacional, podrán representar los conceptos hasta aquí desarrollados. Apoyándome en el esquema generador inicial expuesto en la figura 111. Los enfoques son los siguientes:

1] Recombinación: colección e hibridación

La dialéctica entre estos dos conceptos es importante, ya que el resultado de una combinación es raramente un compromiso establecido, es decir, las sinergias pueden ser aprovechadas a través de efectos no deseados en una operación, que podría emerger a partir de significados individuales o colectivos de los flujos de movimiento de los seres humanas, afectando la completa taxonomía por medio de diferencias formales dentro de una alineación no definida codificada. Las nuevas combinaciones recontextualizan y reinterpretan a su código generador, así como a sus alrededores.

2] Abstracción

La abstracción implica la anulación de lo común, de lo familiar, de las tipologías ya establecidas, con el propósito de eliminar toda representación condicionada. A través de la reconstrucción de funciones relacionadas con el territorio, con los límites y las interfaces de generación 3D, los significados podrán cambiar y originar representaciones de cuerpos, campos, planos, volúmenes, raíces, flujos, entre otros. La libertad creativa de este enfoque es debido a las configuraciones espaciales abiertas, así como a las entidades abstractas que entran en forma de funciones codificadas en la composición.

3] Analogía

La analogía es un motor muy potente de invención con respecto a los diagramas de organización, los lenguajes formales y los sistemas tectónicos. La codificación de las transferencias analógicas es un aspecto que podemos observar en la naturaleza: bosques, cañones, dunas, deltas de ríos, glaciares, fallas geológicas, flujos de lava, entre otros. Detrás de estas formaciones tan específicas, radican las características formales del paisaje, que conducen a la articulación unitaria, al relacionarla con la arquitectura. La noción del paisaje artificial (ciudades), es coherente con sistemas de generación espaciales individuales y colectivos al mismo tiempo; operan a través de los gradientes, eliminando los límites de generación controlados. Proliferan en variaciones infinitas, eliminando la repetición discreta. Tienen leyes particulares. Son indeterminados, lo que permite la libre interpretación por parte de los habitantes, generando ilusión.

4] Variación

La inconsistencia inicial al representar texturas a través del dibujo, es que resulta en iteraciones interminables que operan como una máquina abstracta, ya que proliferan las diferencias a partir de la forma seleccionada. Una vez que una extraña textura o figura es seleccionada y confrontada con una agenda programática, un nuevo tipo de dialéctica relativa al nuevo objeto es generado. La irracionalidad arbitraria y radical denota conclusiones con resultados variables. Este espejo puede ser explicado al reconocer que todos los funcionamientos son relativos, ya que son articulados por organismos que comenzaron siendo aberraciones monstruosas, eran deficientes y no eran adaptables al entorno, para luego convertirse en organismos organizados de manera muy bella, ser adaptables y funcionales de manera excepcional. Antes de poder descartar formalismos arbitrarios necesitamos experimentar y generar representaciones de tales organismos artificiales, con el fin de poder adherirlos a teorías dogmáticas variables, que deriven en estructuras tridimensionales capaces de cambiar y ser observadas por la civilización tecnológica actual. Una de las maneras de llevarlo a cabo es través de la experimentación computacional-logarítmica, que nos permite examinar un gran rango de diferentes opciones. No hay un objeto óptimo y absoluto, así como no hay una perspectiva humana única. Cada medida comienza con una matriz finita codificada de opciones arbitrarias (random), para comparar, seleccionar, adaptar y trabajar para definir tales arbitrariedades. Su significado variable, en este caso, podrá relacionarse con la lógica de las innovaciones evolutivas, la mutación, la selección y la reproducción, siendo el principal motor de cambio, relacionado con la cultura de la arquitectura contemporánea.

A partir de los enfoques antes descritos, se desarrollará un diagrama de generación inicial, tomando en cuenta, también, el esquema de conceptos relacionados con las etapas de producción algorítmica, referidos al comportamiento externo de las ciudades por medio de los flujos y las codificaciones de comportamientos, que regulan las actividades humanas, la forma y sus aspectos generadores, el medio digital computacional y las herramientas de programación textual, que en combinación con una gráfica *DAG*, proporcionarán el medio de representación más adecuado y óptimo para lidiar con la variabilidad y complejidad geométrica (figura 114).

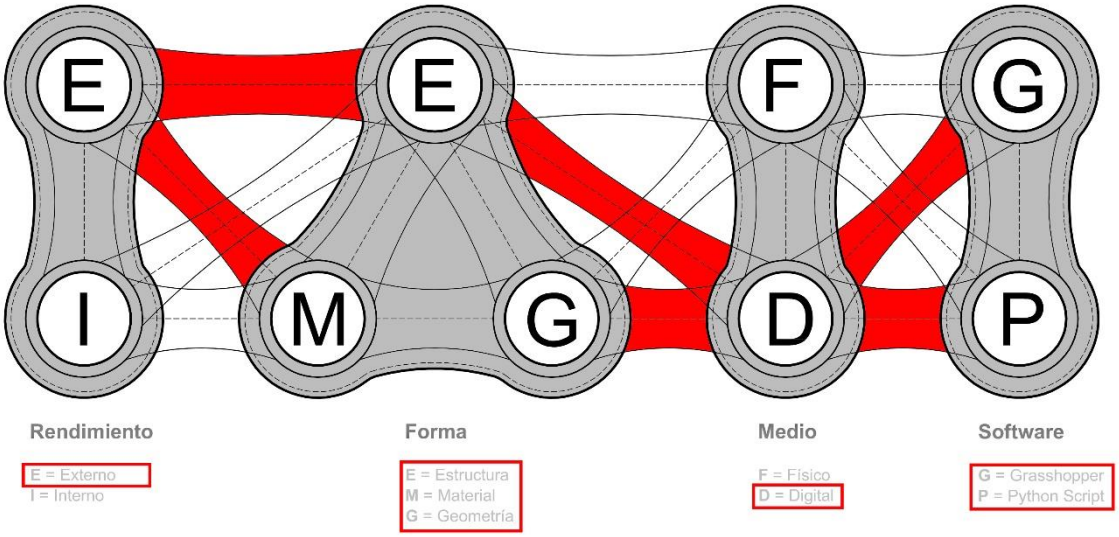
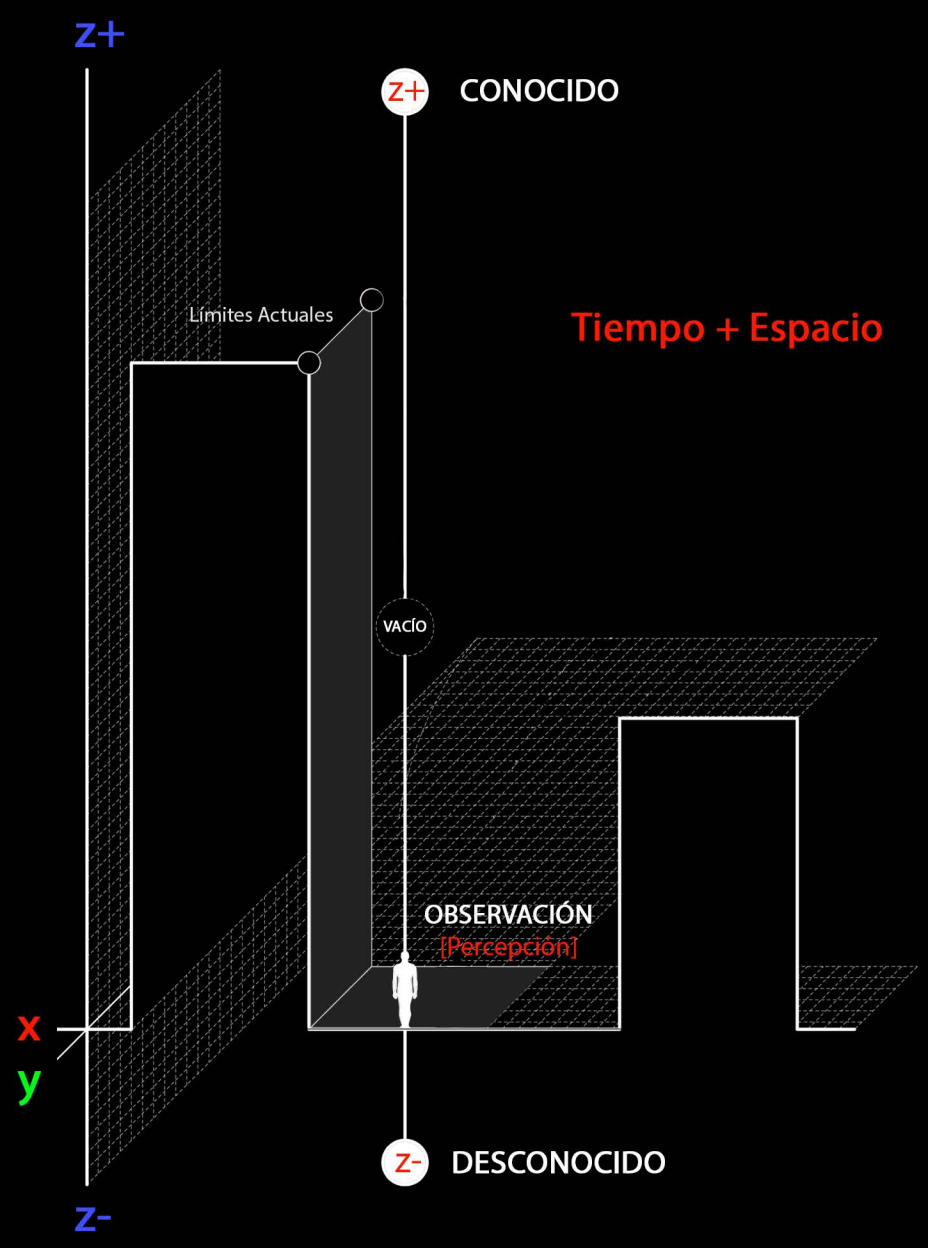
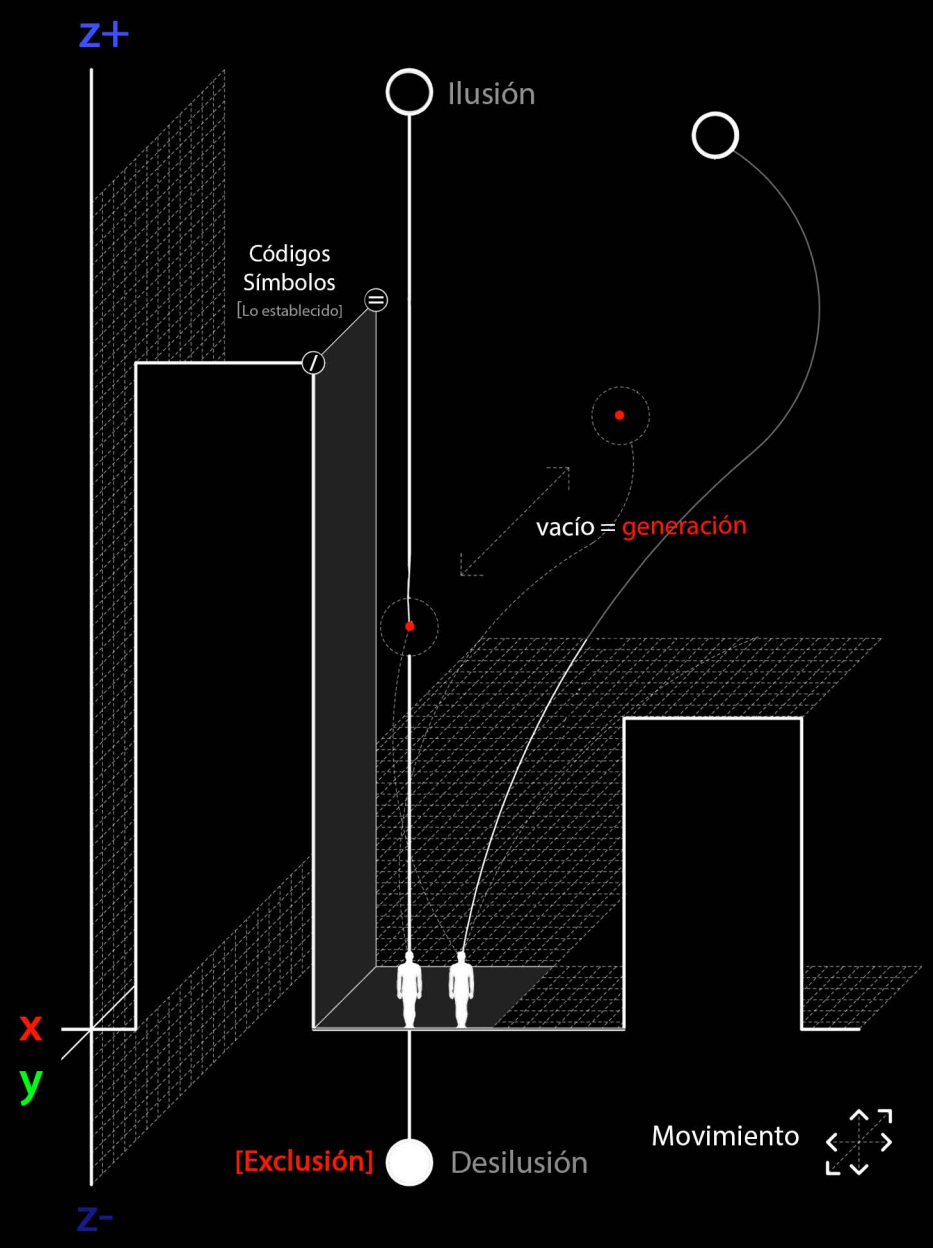


Figura 114. La combinación final seleccionada, a través de un proceso de experimentación con las relaciones de grupos y sus variables particulares, referidas al rendimiento, la forma, el medio y la representación por medio del software. Como primer concepto para el análisis en el segundo caso de estudio, se tomará como aspecto inicial el comportamiento y el rendimiento externo en estructuras artificiales de hábitat humano. El segundo aspecto a considerar se relaciona con las propiedades de generación formal, tales como la estructura, la materia y la geometría. El medio de representación será a partir de conceptos y herramientas computacionales digitales, a través del uso de software del lenguaje de programación textual *Python* + *Grasshopper*, para la generación de algoritmos que interactúan en conjunto con el programa Rhinoceros 5.0, medio de representación geométrica final.

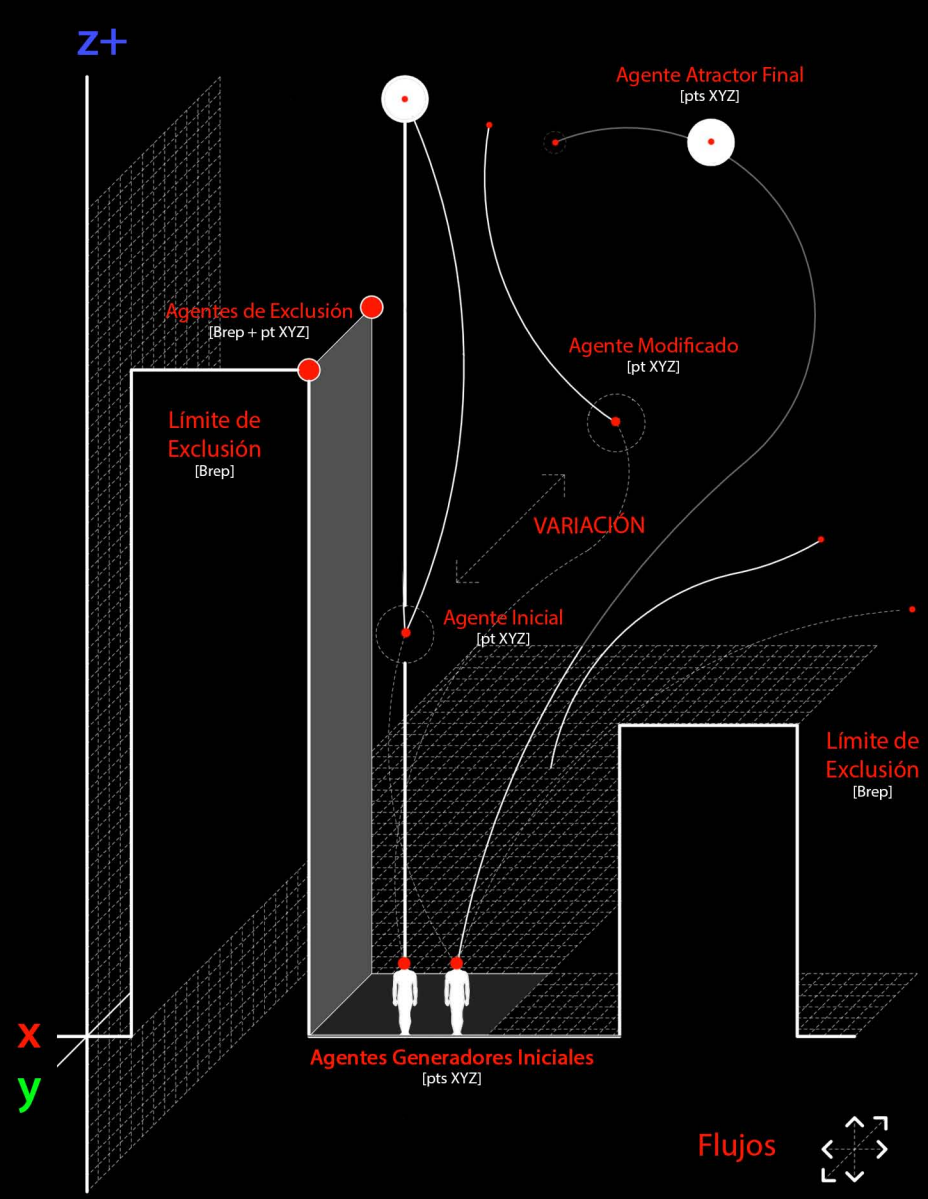
Una vez seleccionados los aspectos de generación algorítmicos de manera conceptual, el diagrama de generación se relacionará con tales aspectos, con los enfoques y con el esquema de generación inicial (figura 111), referido a los conceptos desarrollados a partir de analogías en las ciudades. Este acercamiento, además, nos permitirá relacionar y dividir cada aspecto, con el propósito de generar una sintaxis organizada en el código de programación, que proporcionará el control y el motor de la representación de tales sistemas variables en el espacio tridimensional (figura 115).



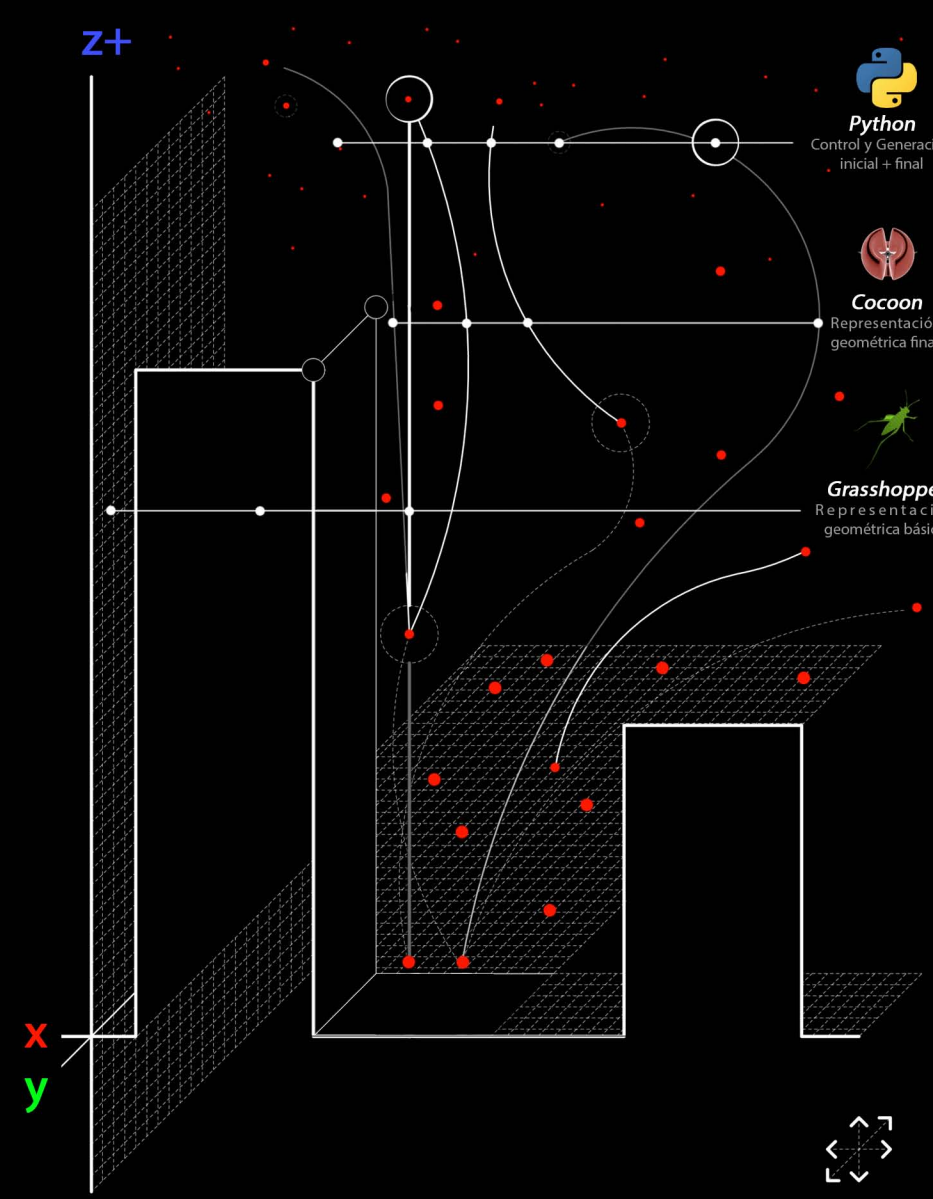
A - Enfoque 01 [Recombinación] + 02 [Abstracción]



B - Enfoque 03 [Analogía]



C - Enfoque 04 [Variación]



D - Representación Final

Figura 115: Diagramas [Generador + Programático] iniciales del caso de estudio 2.

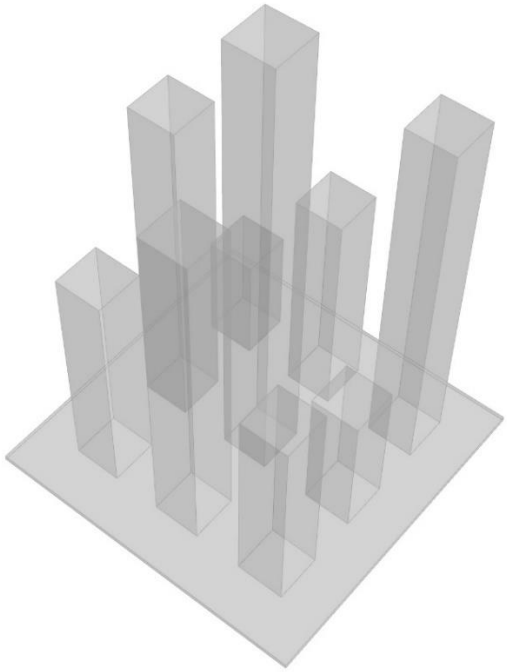
5.6.1 – 1ra Etapa - Rendimiento + Enfoques [1 – Recombinación] + [2 - Abstracción]

En relación al comportamiento o rendimiento externo, nos enfocaremos en definir un sistema urbano geométrico digital random, con el propósito de aplicar los parámetros a otros sistemas definidos, es decir, desarrollaremos la geometría de generación y control inicial. Para ello debemos hacer uso de las herramientas de producción de sólidos de *Rhinoceros 5.0* para definir los edificios random en los cuáles se generarán los flujos a través de la recombinación de posiciones de seres humanos simulados que recorren la ciudad virtual de manera random, y que generan flujos variables. Posteriormente haremos combinaciones de posiciones en relación al movimiento de cada unidad [punto = ser humano], y recombinaremos aquellos que queden exentos de los edificios, dado que el comportamiento es externo y el medio de generación de inicio será el vacío presente en los trazos urbanos.

Para generar los puntos y flujos de manera random, desarrollaremos un código de control basado en el lenguaje de programación *Python* en *Grasshopper* a través del componente *GhPython*, en el cual determinaremos el número seres humanos que se desplazaran en el sistema urbano digital inicial. Es importante, por otra parte, determinar los límites de propagación con el fin de tener un control en la generación de inicio y no generar errores de procesamiento al combinar los códigos y los componentes de *Grasshopper*. Así, la primera parte de este desarrollo consistirá en desarrollar un entorno urbano tanto en *Rhinoceros 5.0* como en *Grasshopper*, para luego incorporar los agentes [seres humanos] que generarán los flujos en base a su desplazamiento controlado por un código.

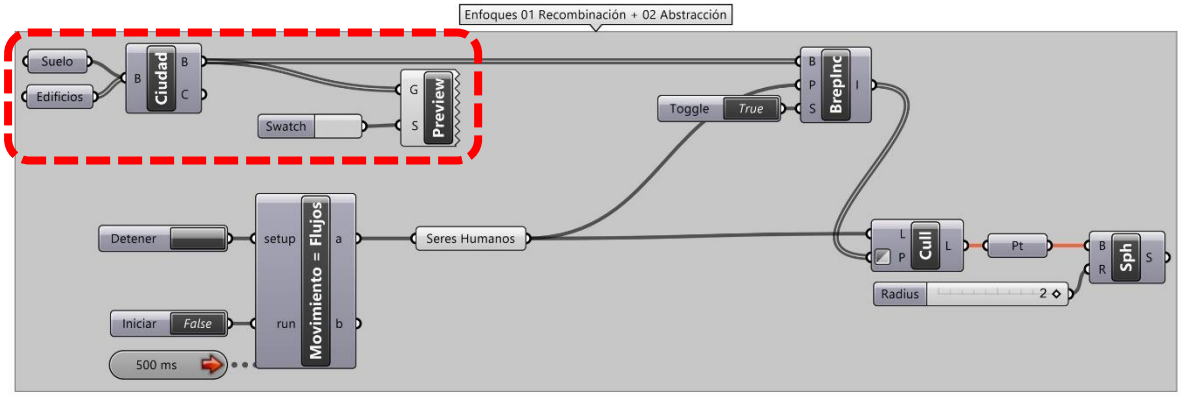
A] Recombinación = Geometría *Rhino* a Datos *Grasshopper* + *Python*

Como primer paso vamos a realizar una instancia de la geometría generada en *Rhinoceros 5.0* que representará una ciudad virtual en *Grasshopper*, es decir, convertimos geometría digital en datos, con el fin de poder manipularlos mediante un lenguaje de programación gráfico y textual.



1A. Geometría inicial representativa de los edificios de una ciudad virtual, mediante la herramienta Box en *Rhinoceros 5.0.*

Al ser geometría representada con un componente de datos en *Grasshopper*, el entorno urbano virtual podrá ser modificado de acuerdo a parámetros específicos de ciudades, este es solo un ejemplo y modelo estándar de generación, que puede ser sustituido únicamente al relacionar otro tipo de geometría con el componente *Brep* de *Grasshopper*.



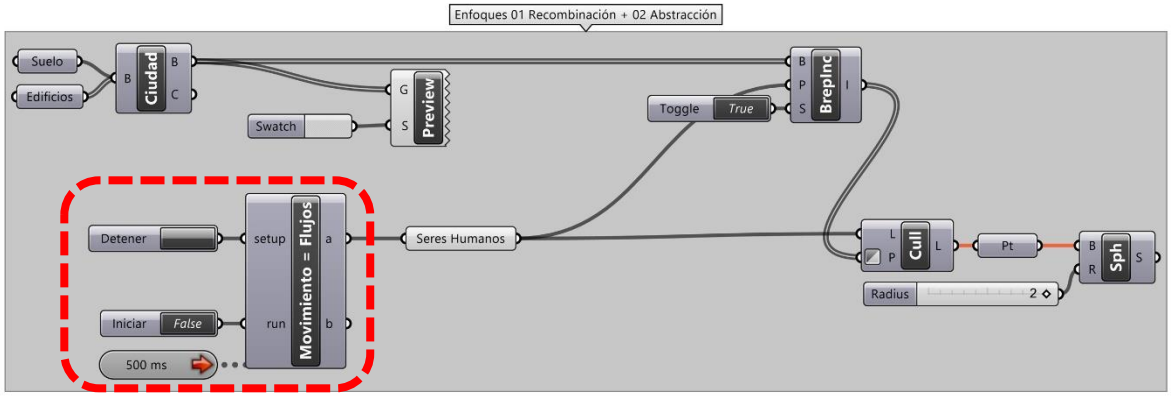
2A. Los componentes *Brep* [Suelo + Edificios], *Brep Join* [Ciudad] y *Custom Preview* para observar la geometría en el entorno de programación gráfica Grasshopper.

La ciudad virtual en este caso es representada por el componente *Brep Join* y visualizada mediante el componente *Custom Preview*, para la asignación de color en el entorno de trabajo 3D en *Rhinoceros 5.0.* La representación de esta entidad virtual se relaciona con los códigos establecidos que podemos observar en las ciudades, que controlan a su vez, las imágenes que los seres humanos generamos al pensar en sólidos simples como una analogía del tejido urbano exterior. En esta entidad, el nuevo medio de generación será el vacío, que nos brinda una serie de posibilidades de experimentación en nuevas representaciones de flujos a partir del movimiento de agentes generadores.

B] Colección + hibridación

El siguiente paso es producir a esas entidades generadoras de flujos de control. Para poder llevar a cabo lo anterior descrito, utilizaremos un código basado en un lenguaje de programación orientado a objetos, en donde los seres humanos son los objetos generadores. El componente *GhPython* nos ayudará a desarrollar un código de control basado en una sintaxis particular del programa y al mismo tiempo trabajar en conjunto con el entorno urbano virtual realizado e instanciado en un entorno de programación gráfico.

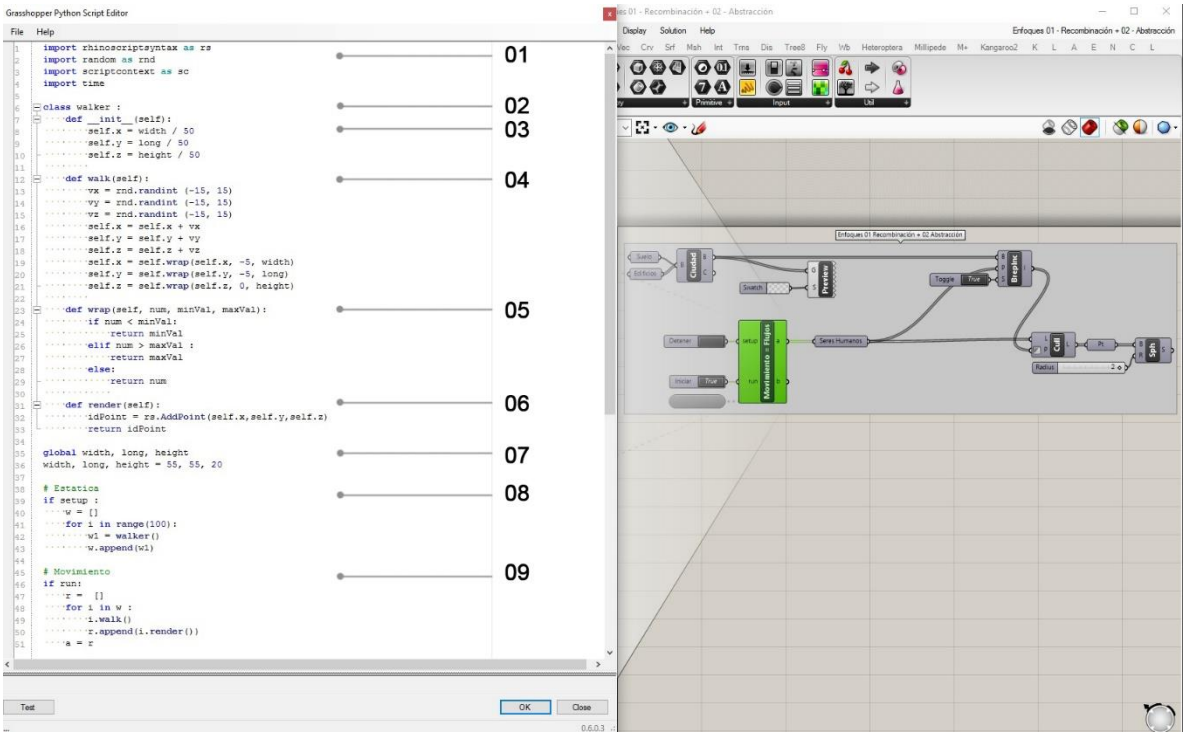
En este apartado dividiremos cada parte del código en secciones, con el propósito de explicar cada uno de sus elementos de generación de agentes. Al ser un código contenido en un componente, tendremos que hacer doble clic para editar la sintaxis de control.



1B. El componente *GhPython* [Movimiento = Flujos] será un elemento de control y generación de agentes.

Un aspecto importante antes de describir cada parte del código es enfatizar el método de programación que utilizaremos, en este caso es relacionado con la “Programación Orientada a Objetos, explicada en el capítulo cuatro de este documento. Al ser un código para generar y modificar objetos es necesario la incorporación de controles gráficos de inicio – fin y ejecución – detención; los componentes *Button* [Detener] y *Boolean Toogle* [Iniciar] cumplirán tales fines, siendo ambos métodos de control de entrada booleanos [*True* = encendido, *False* = apagado]. Por otra parte, al depender de agentes en movimiento, la posición es un aspecto relevante, así como el tiempo. Para asignar movimiento, no podemos restringir a los agentes a una posición específica, ya que los seres humanos nos desplazamos de manera indeterminada, controlados por nuestros objetivos y deseos, que cambian a lo largo del día; es por eso que asignaremos posiciones random, que nos permitirán trabajar de una forma no tan restringida. Para asignar un control de tiempo emplearemos el componente *Timer* de *Grasshopper*, con el fin de evaluar las posiciones de los agentes a lo largo del tiempo, en relación al entorno urbano digital antes generado.

Para generar el código de control deberemos hacer doble clic para escribir o modificar una sintaxis del lenguaje de programación *Python*, que se muestra y explica a continuación:



2B. Sintaxis de control programada en Python, para la generación y control de agentes generadores iniciales.

Las partes de la sintaxis en el código de control y generación inicial son las siguientes:

01 - Importación de librerías:

```
1   import rhinoscriptsyntax as rs
2   import random as rnd
3   import scriptcontext as sc
4   import time
5
```

Dado que la geometría de representación del código será generada en *Rhinoceros*, la librería debe ser importada a la sintaxis de *Python*. Por otra parte, la librería *random* brindará la posibilidad de controlar la generación y movimiento *aleatorio* de los agentes, s*criptcontext* y *time* por su parte, nos ayudarán a controlar el código y su ejecución en relación a una entrada de un parámetro de tiempo [*Timer* de *Grasshopper*].

02 – Definición de objeto + 03 - Posiciones iniciales:

```
6   class walker :
7       def __init__(self):
8           self.x = width / 50
9           self.y = long / 50
10          self.z = height / 50
11          
```

Utilizaremos la Programación Orientada a Objetos a través de *Python*, por lo que tendremos que definir una clase, es decir, un objeto a generar y ser modificado, en este caso la clase es llamada "*Walke*r" [el caminador (es)], que representa a todos los agentes de generación y movimiento iniciales. Al definir la clase, necesitamos proporcionar características al objeto, por ellos necesitamos funciones. La primera es referida a la posición de inicio de todos los agentes a generar, para ello producimos la función *"def__init__ (self):"* para asignar posición al objeto en relación a las coordenadas *x* + *y* + *z*. El operador de división controla el primer movimiento del objeto a los puntos dados en relación a los límites del espacio urbano virtual antes generado.

03 – Variables y límites de movimiento del objeto:

```
12      def walk(self):
13          vx = rnd.randint (-15, 15)
14          vy = rnd.randint (-15, 15)
15          vz = rnd.randint (-15, 15)
16          self.x = self.x + vx
17          self.y = self.y + vy
18          self.z = self.z + vz
19          self.x = self.wrap(self.x, -5, width)
20          self.y = self.wrap(self.y, -5, long)
21          self.z = self.wrap(self.z, 0, height)
22          
```

Una vez que definimos la posición de nuestros agentes iniciales, el siguiente paso es proporcionar datos de desplazamiento a cada uno de ellos, es por eso que definimos otra función "*walk*" [caminar], que controlará el desplazamiento de cada agente de manera random e independiente de los otros agentes; es decir, asignamos un parámetro numérico de control random (*rnd. randint*) para que cada objeto se desplace a lo largo de las coordenadas (*vx, xy, vz*). Por otra parte, las variables *self.x* + *self.y* + *self.z* son definidas con el propósito de controlar los límites de propagación del movimiento de cada uno de los agentes (*self.wrap*), este aspecto es importante, ya que los agentes son las personas en movimiento que se relacionan más allá de unos límites en espacio urbano, pero al ser una simulación tridimensional y controlada por código, es más factible establecer límites para controlar el desplazamiento a lo largo del sistema virtual, de otro modo, no habría oportunidad de controlar a tales objetos, además de requerir recursos de hardware más potentes para llevar a cabo una simulación completa "sin límites".

04 – Función condicional de control:

```
23    def wrap(self, num, minVal, maxVal):
24        if num < minVal:
25            return minVal
26        elif num > maxVal :
27            return maxVal
28        else:
29            return num
30
```

Al definir movimiento a partir de un objeto, la variable del tiempo es imprescindible, es por ello que utilizamos un *Timer*, con el fin de controlar el cambio de posición de cada agente, dado que el desplazamiento depende siempre del tiempo-espacio, está variable es asignada a través del nombre de *"num*", y al ser un parámetro de control generativo e inicial, es producido en una nueva función (*wrap*), la cual establece que si no hay tiempo, no puede haber generación mediante una ejecución condicional [*if* + *elif* + *else]*, es decir, no hay generación de agentes y movimiento sin tiempo.

05 – Datos geométricos de agentes generadores:

```
31    def render(self):
32        idPoint = rs.AddPoint(self.x,self.y,self.z)
33        return idPoint
34
```

Hemos generado datos de posición en base a coordenadas tridimensionales, control de movimiento en cuanto límite y condiciones de ejecución en relación al tiempo, todos estos parámetros y variables deberán ser ahora representados a través de geometría en *Rhinoceros 5.0*, nuestro medio de representación y de trabajo digital. Para ello definimos una nueva función (*render*), al definir la variable *idPoint* a través de *rs.AddPoint*, asignamos puntos a los datos desarrollados en las anteriores funciones, si no hay datos o se genera algún error, entonces se regresa a la última línea de código (*return idPoint*), para corregir errores en las funciones.

06 – Control de posición de agentes en relación al entorno urbano virtual:

```
35  global width, long, height
36  width, long, height = 55, 55, 20
37
```

Representamos datos en puntos 3D, lo que necesitamos ahora es definir la propagación de los agentes en relación a los límites geométricos del entorno urbano virtual; en este caso los límites son: 60 unidades [x], 60 unidades [y] y 150 unidades [z], es decir, el ancho es de 60 unidades, el largo de 60 unidades y la altura máxima de los edificios, por ejemplo, es de 150 unidades. Los agentes se relacionarán y propagarán en estos límites a través de las variables *width* [55], *long* [55] y *height* [20], desplazándose en los límites del entorno urbano virtual y representando el movimiento aleatorio de los humanos. Lo anterior será denominado como "colección" un concepto explicado anteriormente y referido a un conjunto de objetos que producen el primer aspecto de generación en base al tiempo-espacio.

07 – Número de agentes estáticos:

```
# Estatica
if setup :
    w = []
    for i in range(100):
        w1 = walker()
        w.append(w1)

```

La hibridación es un estado físico relacionado con la estática, con lo inerte, con lo que no puede desplazarse; en este caso de estudio, al referirnos a un ser humano, podemos determinar que esta característica también aplica a nosotros. Al definir a los agentes como entidades relacionadas y generadas a partir de la posición y el tiempo que ocupan en el espacio, presentan hibridación, es decir, no se pueden desplazar si no hay control a pesar de haberlo definido en las anteriores funciones. Por ello, necesitamos establecer un límite o número de agentes que interactuarán en el sistema urbano y que dependen de las funciones que los generaron, referidas a datos, geometría y movimiento.

Estática representa al primer estado de generación, es decir, a los agentes inertes. Para controlar el número de agentes en el sistema necesitamos establecer y asignar un parámetro de control numérico, en este caso también es una condición (*if: setup*), ya que al no haber algún número de agentes no puede haber generación geométrica y al ser un número de repeticiones, son un rango (*for i in range*), en dónde (*100*) representa al número de agentes a generar.

Por otra parte, el objeto es definido a partir de una clase, que es generada a su vez por funciones, que son las características que definen al objeto, esto es conocido como relación de método, es decir, el objeto depende de funciones que tienen que ser incorporadas en la representación final, por lo anterior se define la variable *w1 = walker ()*, que representa a todos los atributos (funciones) en un objeto. Tenemos hasta ahora puntos independientes uno del otro, para compactar a todos los puntos o agentes los definimos *w.append (w1)*, con el fin de controlar a todos los objetos de manera general, pero que se generan a partir de variables y parámetros individuales.

08 – Representación de agentes y movimiento:

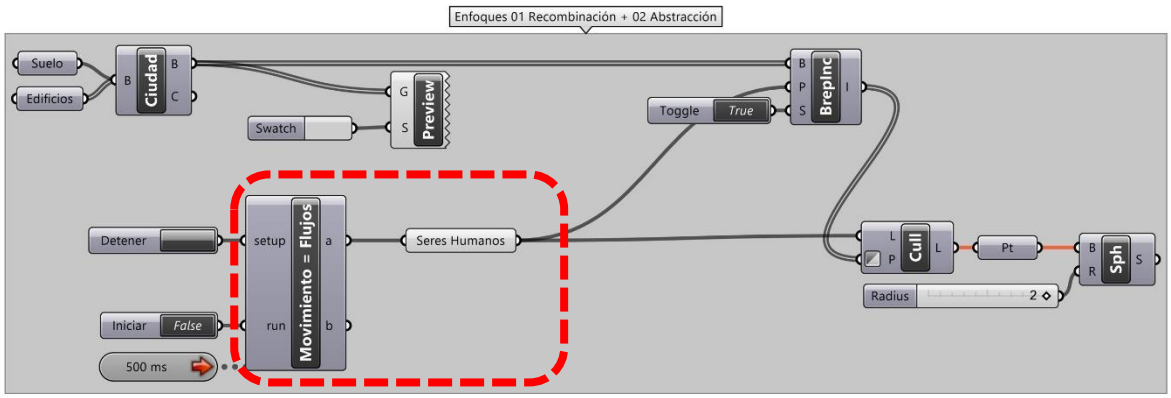
```
# Movimiento
if run:
    r =  []
    for i in w :
        i.walk()
        r.append(i.render())
    a = r
```

Una vez que tenemos a los agentes, es momento de modificar la posición de cada uno por medio de las funciones que los definen. Este último paso representa el control del movimiento de los agentes [puntos 3D]. Como la ejecución depende de un control booleano, es una condición (*if run:*), la variable r es definida para asignar el movimiento a los objetos y el rango define a los objetos que de desplazaran [*i*], es decir, a todos los agentes generados. Para tener control del sistema general en cuanto al movimiento se define

r.append (i.render ()), que son todos los agentes ahora en desplazamiento. Por último, *a = r* es el método y la representación final de todos los datos de salida en el componente ahora producido a partir del código y la sintaxis del lenguaje de programación *Python*. Está representación final son agentes, son puntos y son un concepto de geometría abstracto que simula la posición y desplazamiento random de los seres humanos en un espacio y sistema urbano virtual.

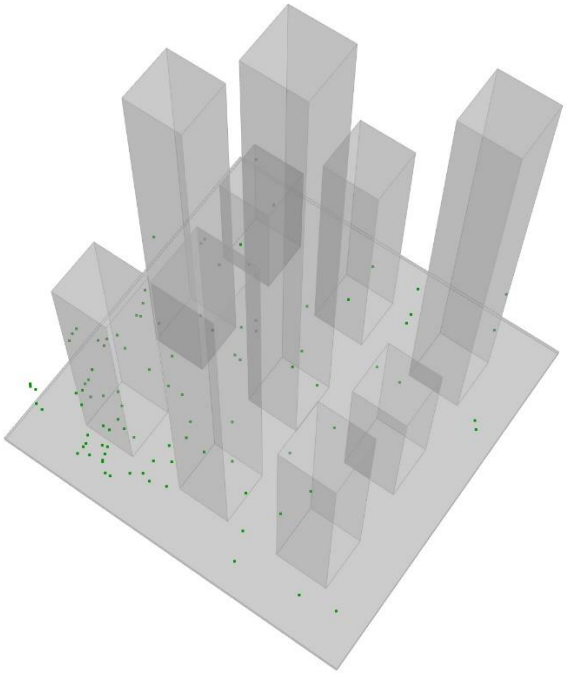
C] Abstracción

En relación a la generación de agentes anterior en base a un código algorítmico de programación textual, el resultado es un conjunto de puntos en el espacio tridimensional y en relación con un sistema urbano virtual.



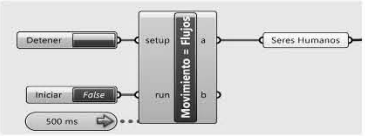
1C. El componente *Point* [Seres Humanos], representa a cada agente de generación inicial

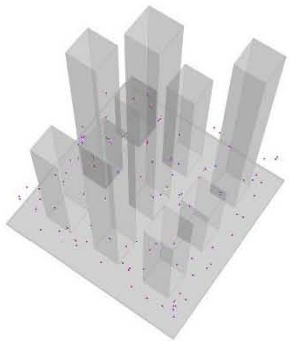
Los puntos son seres humanos estáticos y luego en movimiento, es decir, es una abstracción relacionada y generada a partir de un ser *"natural"* y un ser *"artificial"* producido y simulado computacionalmente.



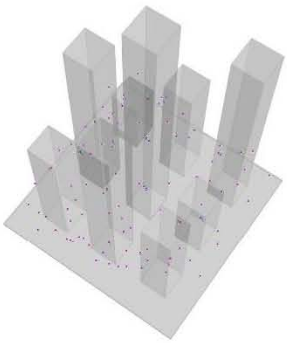
2C. Agentes de interacción y generación geométrica inicial.

Los agentes al depender y ser generados por el tiempo, nos proporcionarán una condición de variabilidad geométrica para el inicio de la representación de los flujos, por lo que cada iteración es diferente, al igual que cada momento o suceso en el tiempo es diferente.

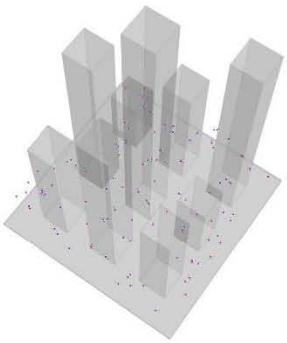




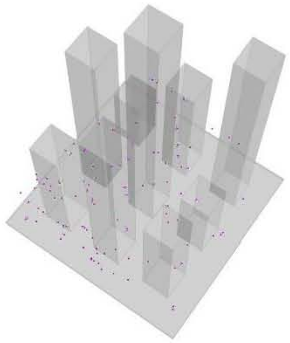
A] + 500ms (x1)



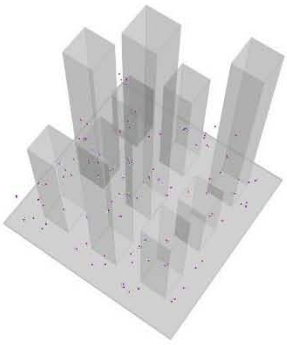
B] + 500ms (x2)



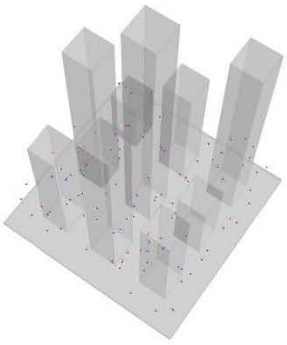
C] + 500ms (x3)



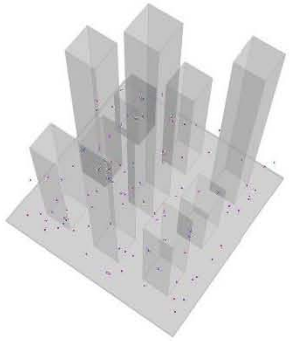
D] + 500ms (x4)



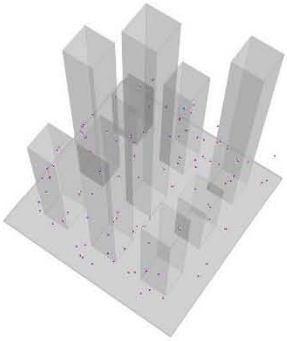
E] + 500ms (x5)



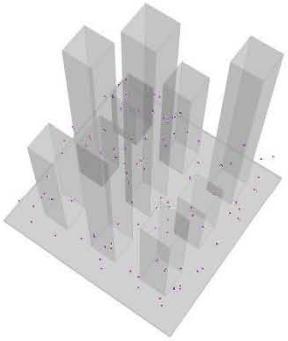
F] + 500ms (x6)



G] + 500ms (x7)



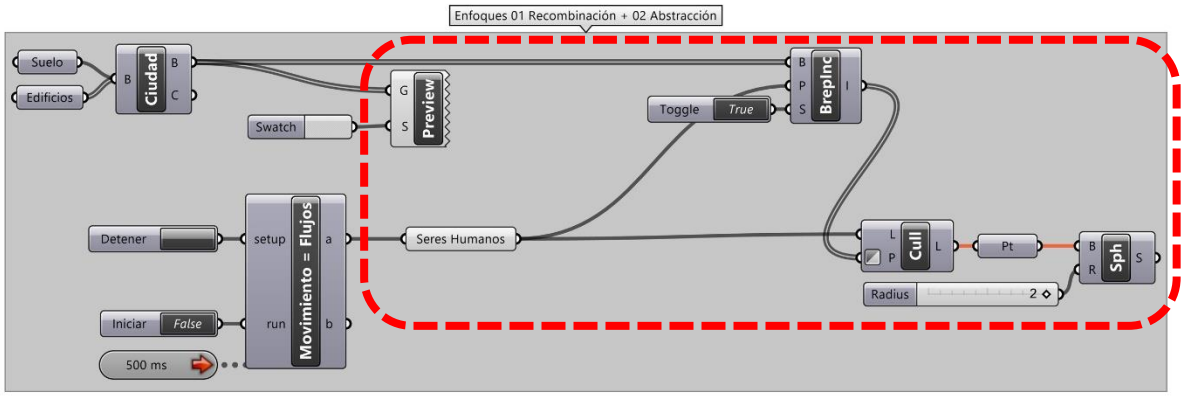
H] + 500ms (x8)



I] + 500ms (x9)

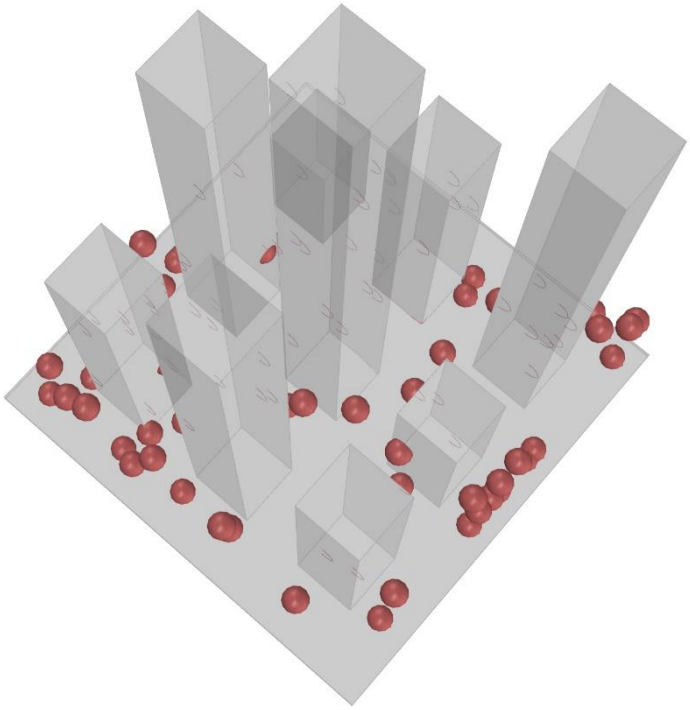
Figura 116. Iteraciones generativas iniciales variables con base en el control del tiempo.

Por último, al tratarse de un comportamiento o rendimiento externo, es necesario excluir a aquellos agentes que interactúen en el interior de la geometría, con el fin de obtener y trabajar con agentes exteriores que puedan ser los generadores de los flujos y se relacionen con los vacíos del sistema urbano virtual, a desarrollar en los siguientes enfoques. La exclusión la realizaremos utilizando los componentes *BrepInc* + *Cull Pattern* de *Grasshopper*, a continuación:



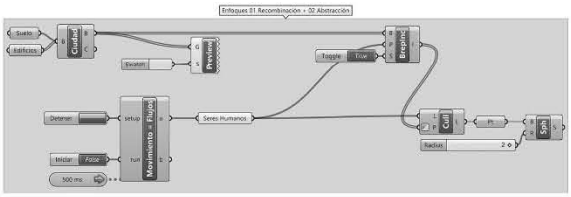
3C. Relación del sistema urbano *[Ciudad]* con puntos y exclusión con los componentes *BrepInc* + *Cull Pattern*.

El componente *BrepInc* (*Point in Brep*) excluirá a aquellos elementos que se encuentren en el interior de la geometría [*Ciudad* – entrada *B*], los agentes son el medio generador inicial a excluir [*Seres Humanos* – entrada *P*], la condición booleana de *True* está activa, ya que deseamos ejecutar tal acción. El componente *Cull Pattern* selecciona a los elementos exteriores a la geometría [la condición *Invert* está activa], con ello todos los puntos exteriores son separados de los que se encuentran en el interior de la geometría. El componente *Sphere* cumple únicamente a con la función de representar tal exclusión de una manera más clara y gráfica.



4C. Exclusión de elementos interiores. Los agentes exteriores son ahora el medio generador inicial en el vacío.

Dado que cada iteración es variable en función del tiempo, se obtienen agentes en movimiento y externos a los edificios del sistema urbano, brindando la libertad de experimentación para representaciones geométricas diferentes en el vacío, como segundo ambiente virtual generador.



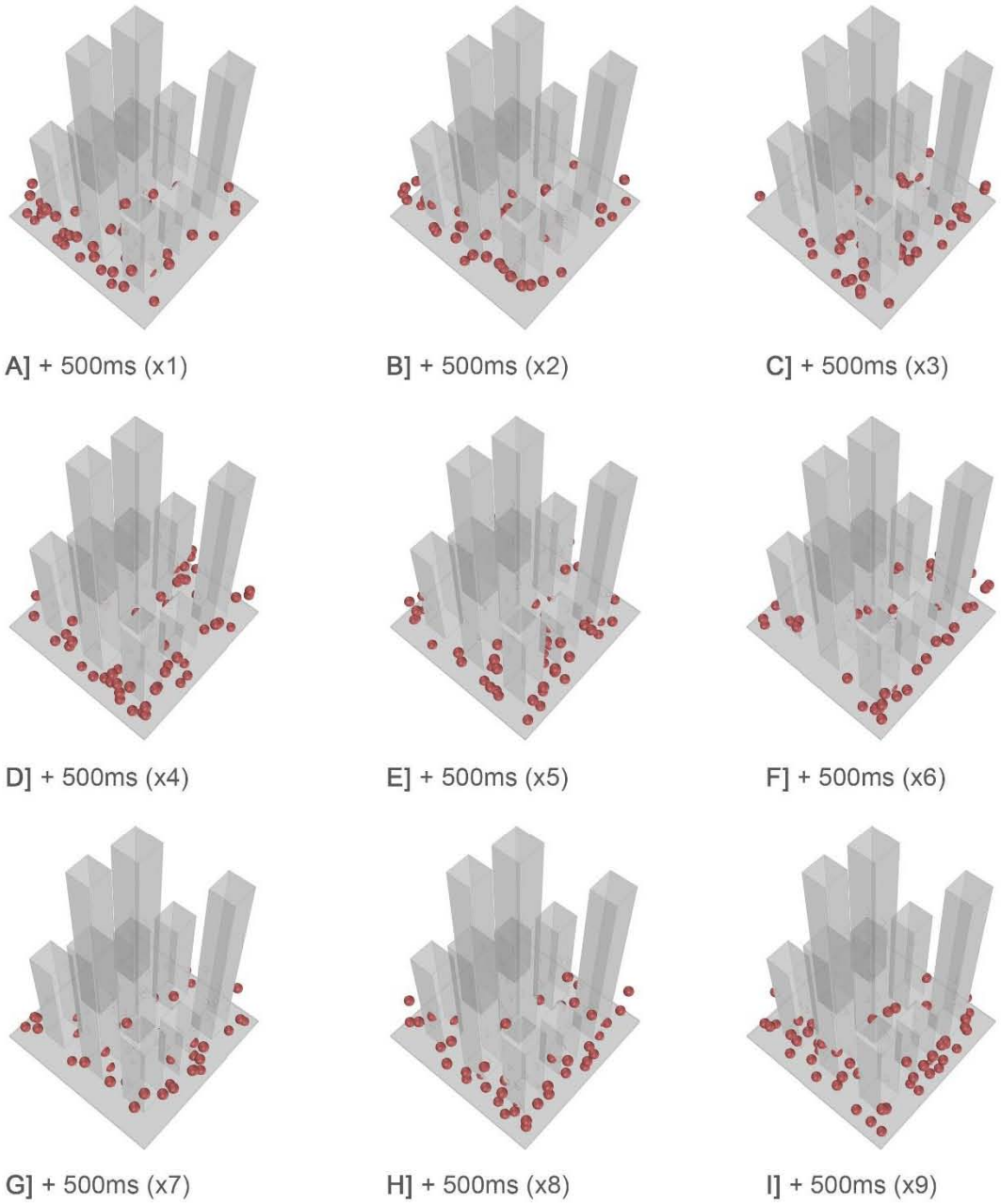


Figura 117. Agentes exteriores de generación inicial espacio – tiempo.

5.6.2 – 2da Etapa - Forma + Enfoque [3 – Analogía]

A partir de la incorporación de los agentes con movimiento, podremos obtener distintas variaciones en cuanto a la posición de cada uno para, posteriormente, relacionarlos como elementos iniciales de producción geométrica. En este caso, y dado los recursos en hardware, trabajaremos únicamente con seis agentes, haciendo una analogía de [1 agente = 1 humano], con el propósito de que la ejecución en la representación geométrica no cause errores y pueda ser llevada a cabo de manera óptima. Queda abierta la experimentación y variación con un número mayor de agentes en computadoras más potentes que puedan ejecutar el *Script Grasshopper* + *Python* a desarrollar a continuación. Los seres humanos (agentes) seleccionados en el sistema virtual urbano anterior y los agentes de exclusión en el sistema urbano se muestran a continuación (figura 118).

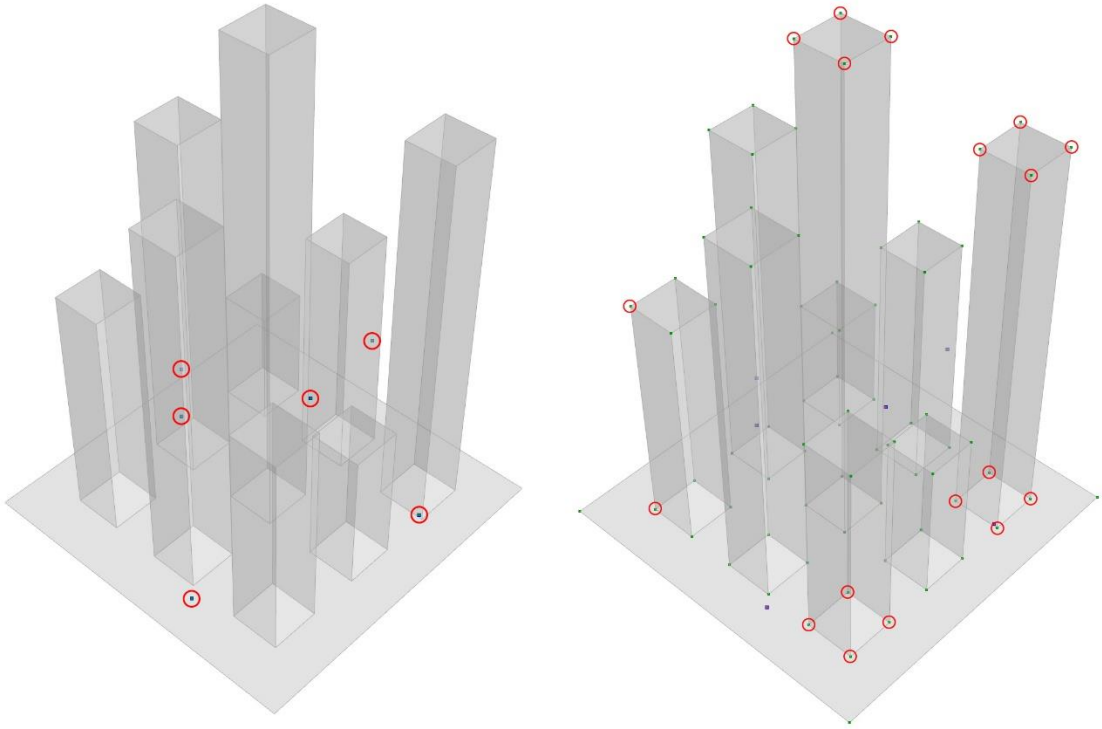
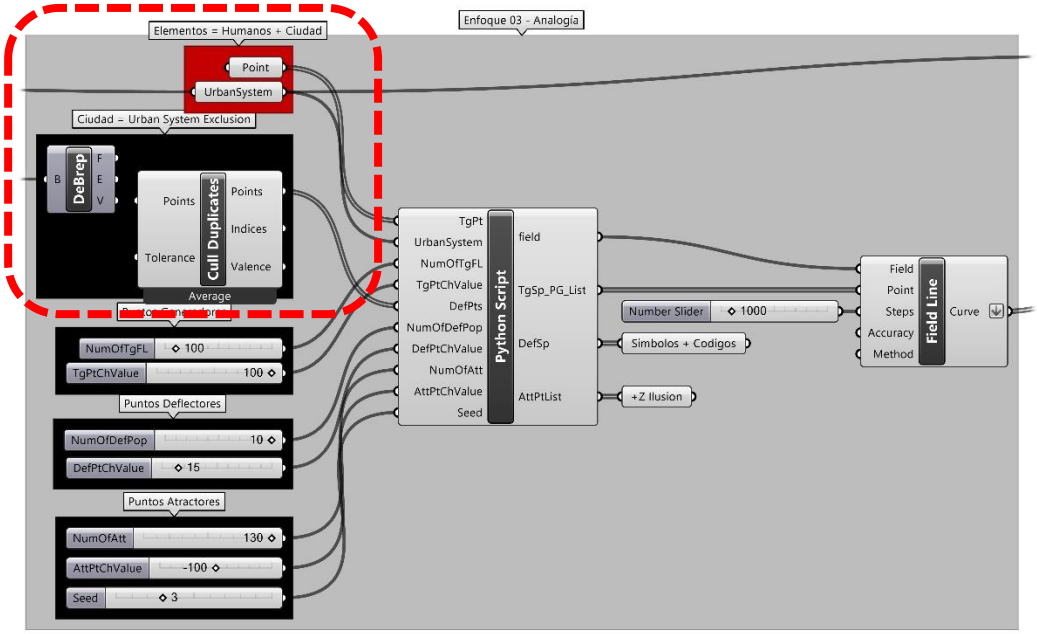


Figura 118. Agentes-Humanos [6] exteriores seleccionados de generación inicial + Agentes de exclusión urbanos

A] Analogía generativa inicial

Una vez que hemos seleccionado a los agentes iniciales, representados a través de puntos en el espacio tridimensional de *Rhinoceros 5.0*, debemos instanciar a la geometría en datos de programación gráfica, utilizando el componente de *Grasshopper Point*, en el cual relacionaremos a los seis puntos como una entidad. Además, utilizaremos el componente *Brep* para representar a la geometría del sistema virtual urbano, que se relacionará con los puntos hasta ahora generados en los espacios vacíos del modelo. El componente *DeBrep* [*Deconstruct Brep*] es empleado para separar a todos los elementos generadores de un cuerpo geométrico sólido, es decir, obtenemos de manera individual cada una de sus aristas, caras y puntos, con el propósito de excluir a los puntos que representan a los símbolos y códigos de todos los edificios del modelo virtual urbano, los límites de generación y, por consiguiente, de exclusión.



1A. Los agentes y el sistema urbano virtual como entradas de parámetros de generación inicial.

Antes de comenzar a describir a los parámetros de control numérico, a través de cada uno de los *sliders* correspondientes y enlazados como entradas al componente de *Python Script* (*GhPython*), es necesario conceptualizar la gráfica *DAG* que brindará la producción y representación geométrica final del sistema de flujos. Para ello, el siguiente esquema fue desarrollado con el propósito de organizar de mejor manera el proceso y los enfoques de diseño aplicados (figura 119).

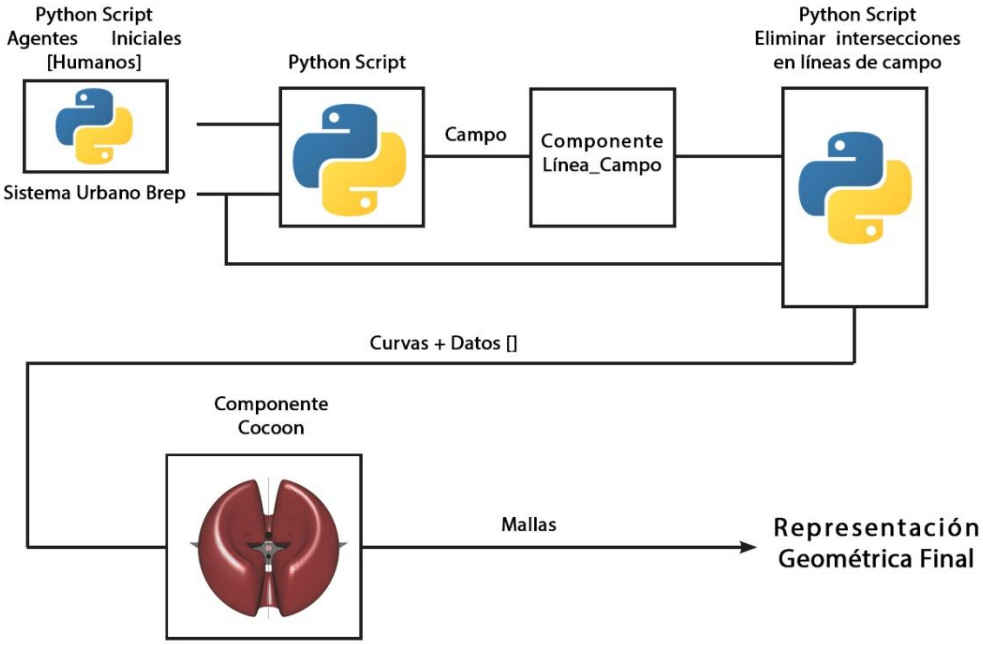


Figura 119. Esquema de desarrollo *DAG* del caso de estudio correspondiente.

En el esquema anterior, desarrollamos inicialmente un componente de *Python* que nos permite controlar y generar agentes de manera aleatoria en un sistema urbano virtual, el segundo *Script* nos ayudará a generar el control y representación de los flujos producidos a partir de la ilusión y los deseos de los seres humanos en el vacío; para ello, utilizaremos el componente de *Grasshopper - Field Lines* como instancia de programación, es decir, a partir de la importación del núcleo de *Grasshopper – Kernel*, relacionado con cada uno de sus componentes, además de otras librerías que nos permitirán representar a la geometría final. Al generar líneas de campo (*Field Lines*), necesitaremos eliminar elementos duplicados con otro *Script* de programación, y el resultado serán curvas y datos que incorporaremos al complemento *Cocoon* a partir de sus componentes particulares explicados a detalle en el primer caso de estudio de este documento.

Las líneas de campo son un elemento de *Grasshopper* que dependen de puntos de atracción y repulsión para ser generadas, por lo anterior, necesitamos identificar y relacionar a todos los puntos que tenemos en el espacio tridimensional, referidos a los agentes y a los vértices de cada uno de los edificios del sistema urbano. Por otra parte, los edificios son identificados como *Brep's* de exclusión, ya que no generaremos geometría en interior sino en el exterior (vacío), por lo que deberán de ser eliminados de la producción de líneas de campo (figura 120).

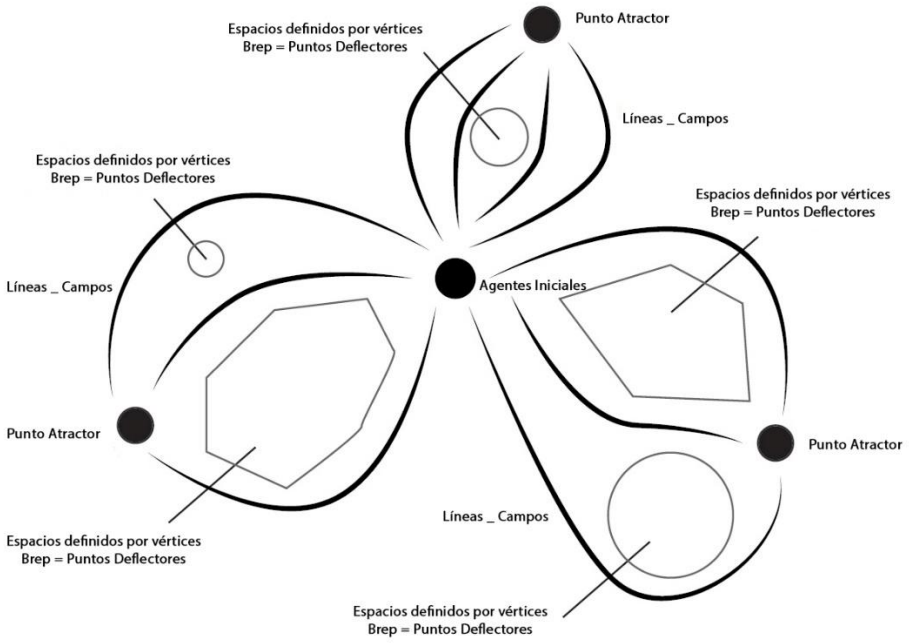


Figura 120. Conceptualización para la producción de líneas de campo, basadas en atracción y repulsión.

Al desarrollar este caso de estudio por medio de la programación textual, el uso del componente *Field Lines* como generador y representación de flujos, constituye un problema sencillo de llevar a cabo con el componente propio de *Grasshopper*; pero al hacer una analogía correspondiente, dentro un entorno de programación textual, el problema se vuelve complejo, por lo que se ha optado por importar las respectivas librerías de *Grasshopper* para analizar el problema de manera más profunda (figura 121).

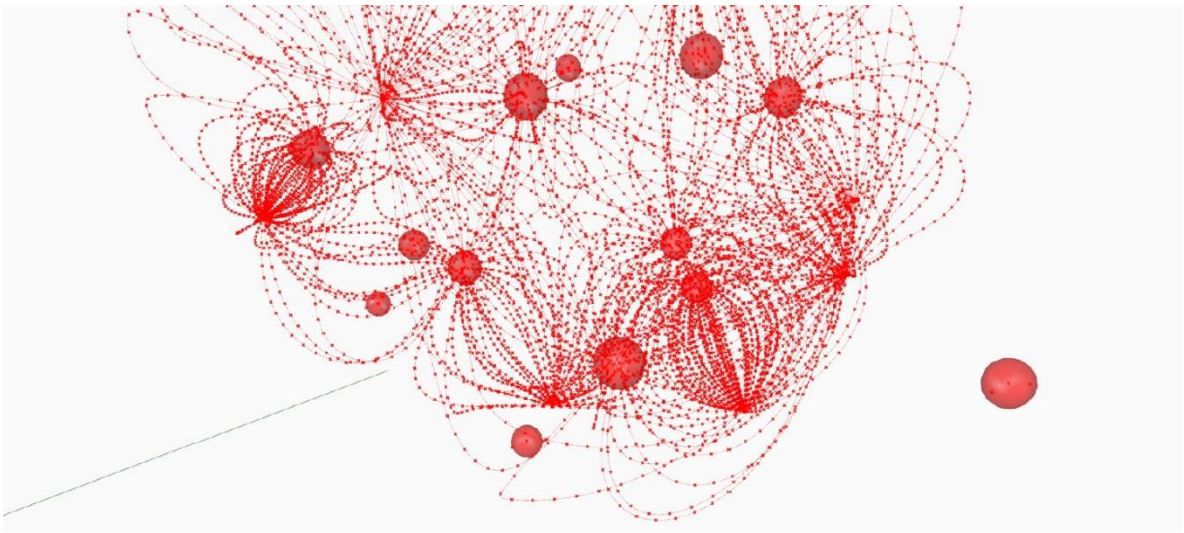


Figura 121. Representación de Líneas de Campo (*Field Lines*) en *Rhinoceros 5.0* a partir del componente gráfico *Field Lines.*

Las líneas de campo en este caso, serán la representación de datos en los flujos producidos por los seres humanos, por lo que los agentes de atracción estarán en la parte superior del sistema, generados por los puntos desarrollados en el primer *Script* de *Python*. Los agentes de exclusión serán los vértices de cada uno de los edificios y los flujos se producirán en el vacío del sistema urbano. Es por lo anterior que requerimos un mayor control, además de conceptualizar una tesis geométrica que nos ayude a programar las líneas de campo mediante la analogía particular referida a la generación de datos geométricos basados en coordenadas cartesianas (figura 122).

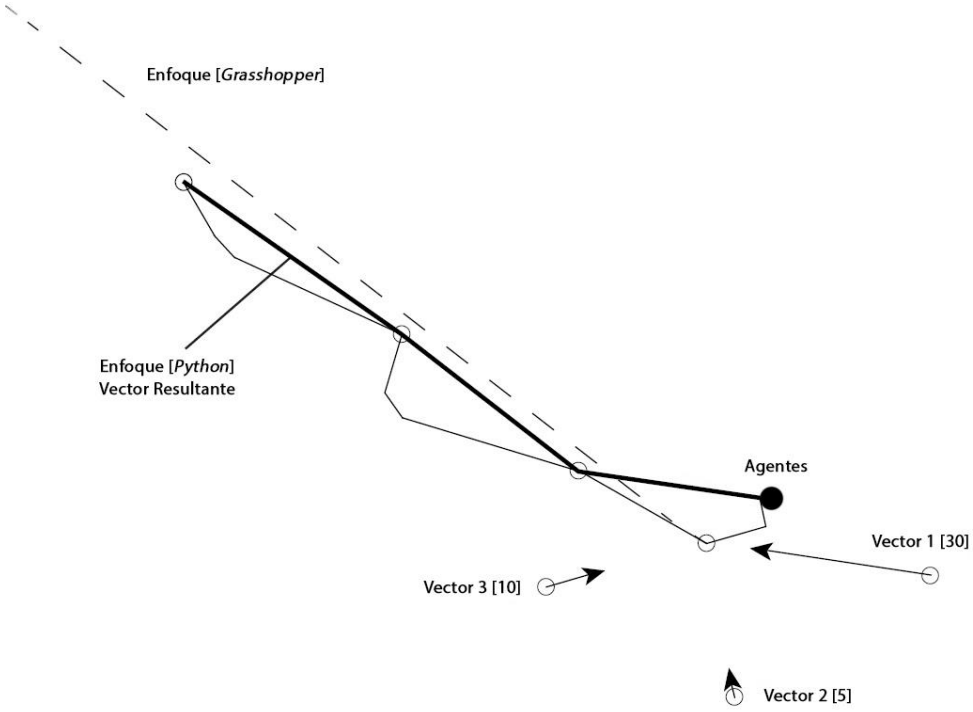


Figura 122. Tesis geométrica para la generación de líneas de campo en un entorno de programación textual basado en *Python.*

El enfoque basado en *Python* contemplará la generación de los flujos a partir de vectores de repulsión producidos por los agentes iniciales, por ello, incorporaremos el componente *Sphere* en cada uno para, posteriormente, poblar a las esferas con puntos generadores (*Grasshopper – Populate Geometry*) de vectores de repulsión, que se dirijan a los agentes atractores exteriores y generen los datos de flujos, por medio del segundo componente de *Python* a desarrollar y explicar en la siguiente sección (figura 113).

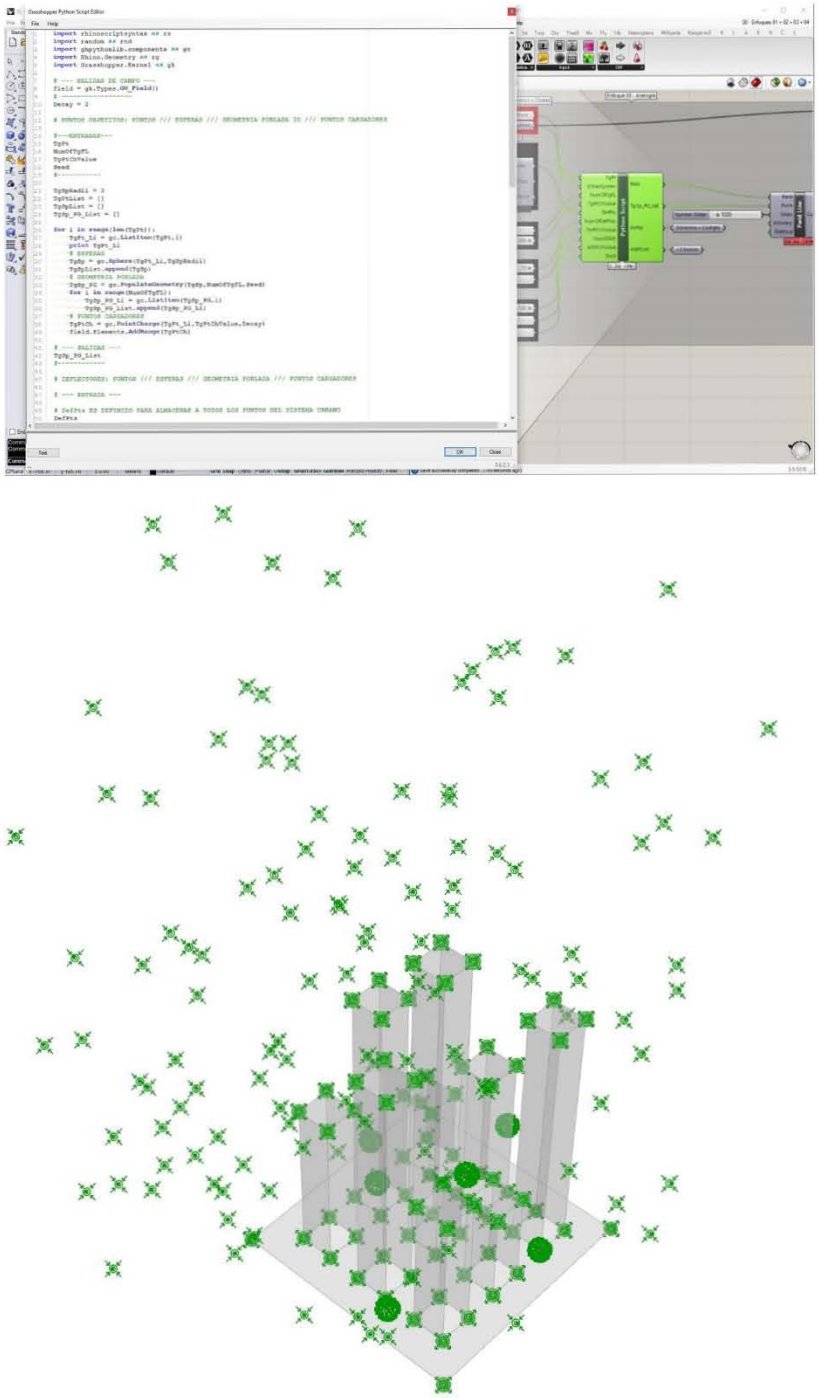
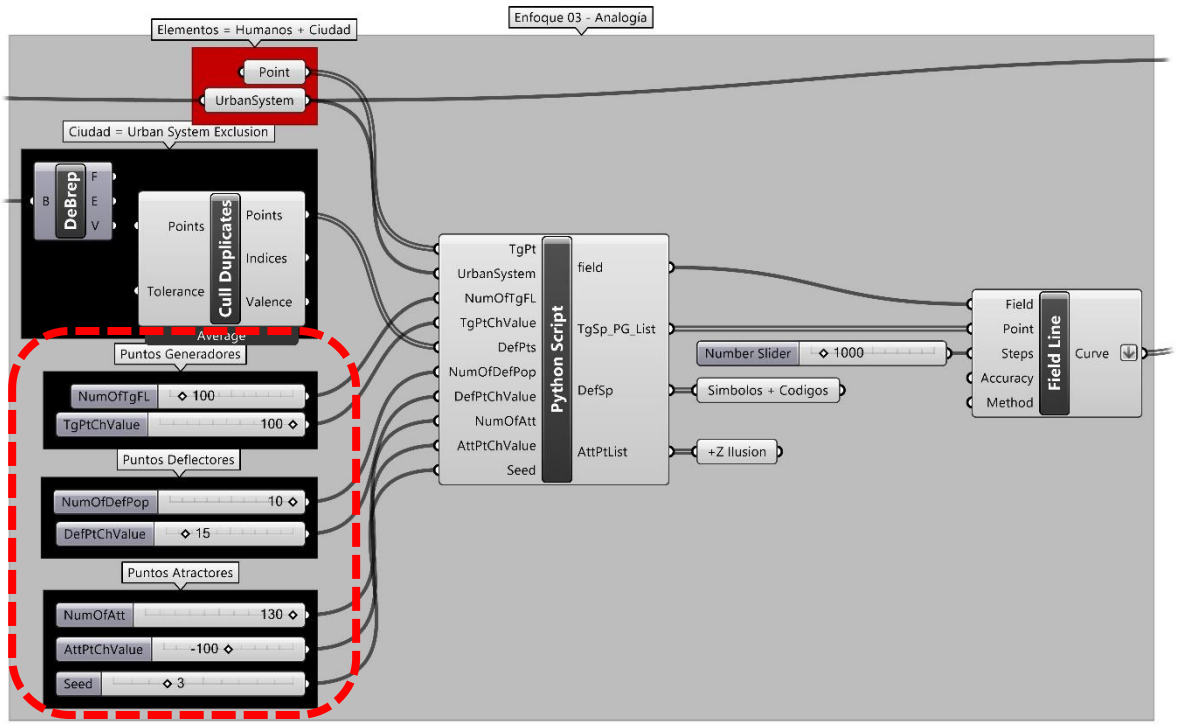


Figura 123. *Script* de *GhPython* para el control y producción de agentes de atracción, exclusión y repulsión.

B] Atracción + Exclusión + Repulsión = Líneas de Campo

Antes de comenzar a explicar las partes de código basado en *Python*, es importante señalar que todos los conceptos y terminología estarán basados en el lenguaje inglés, con el propósito de no alterar la sintaxis original de *Grasshopper* y del lenguaje de programación, siendo el idioma prevalente y original de estas dos interfaces de trabajo, además de tener un mayor control de cada uno de los parámetros de entrada y salida a través de las instancias de generación de datos y geometría.

Como primera etapa, empezaremos a describir los parámetros de entrada numéricos (*sliders*), que nos ayudarán a obtener control y variación en el sistema de datos generado por la sintaxis en el *Script*.



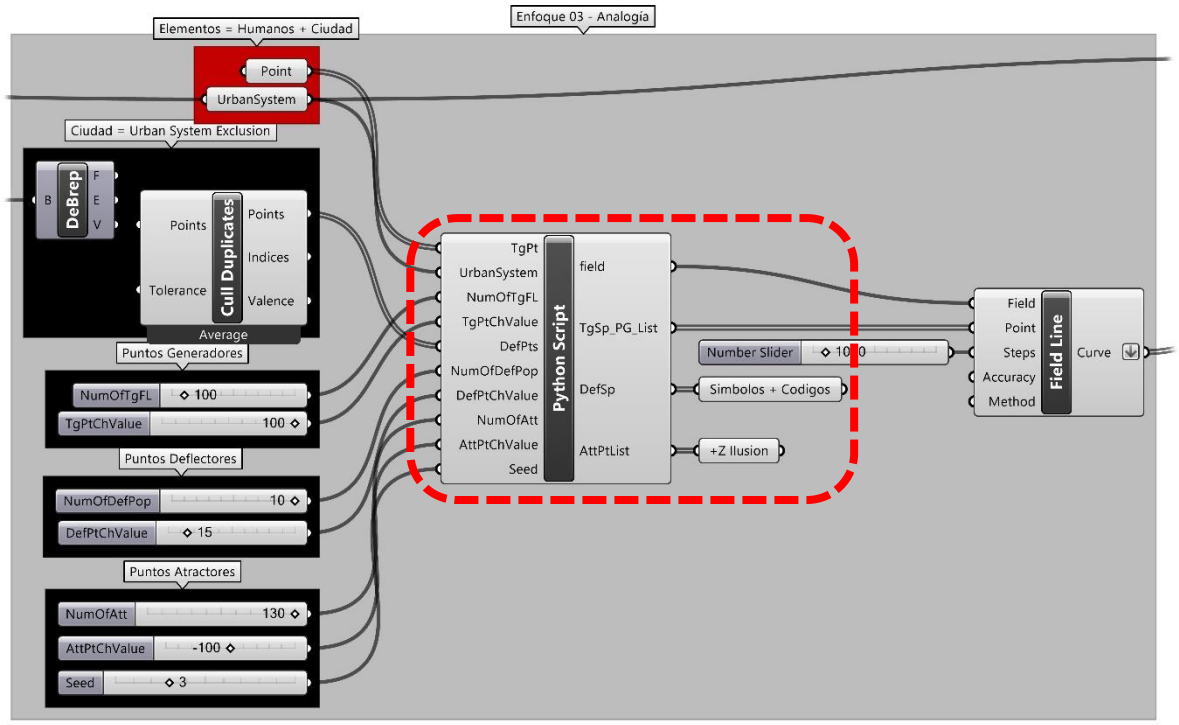
2A. Parámetros de control numérico (*sliders*) del sistema basado en *Python Script*.

A) Puntos Generadores: es incorporado para controlar el número de puntos o vectores iniciales, a partir de los agentes (puntos) iniciales que poblarán a las esferas instanciados en cada uno de ellos*, TgPtChValue* controlará la magnitud de repulsión vectorial inicial.

B) Puntos Deflectores: son los agentes relacionados con cada uno de los vértices de los edificios en el sistema urbano virtual, la condición es que estos elementos sean excluidos para evitar una posible producción de geometría en el interior; *NumOfDepPop* corresponde al número de agentes y *DefPtChValue* a la magnitud de fuerza en cada agente de repulsión.

C) Puntos Atractores: corresponden a los agentes externos en el sistema urbano y el objetivo vectorial de los agentes iniciales; *NumOfAtt* representa al número de atractores, *AttPtChValue* es el valor de atracción de cada uno de ellos y *Seed* representa a la variación aleatoria que puede generarse al modificar continuamente el número, restringido a 10.

Una vez explicados los parámetros de control numérico, continuaremos describiendo la función de los parámetros de entrada y salida del componente *GhPython*.



2B. El componente *GhPython* representará al control y a la generación de flujos basado en líneas de campo.

Los parámetros de entrada numéricos (*sliders*) están relacionados intrínsecamente con las variables de entrada del componente de *Python Script*, por lo que representan a lo mismo explicado con anterioridad, y constituyen controles basados en números enteros (*Integer*) en cada uno. Por otra parte, la relación entre el componente *Point* y la entrada *TgPt* representa a cada uno de los agentes iniciales, instanciados mediante un método de entrada *Point3d*. La entrada *UrbanSystem* es una relación del modelo urbano virtual representado por medio del componente *Brep* y por un método de entrada *Brep*. Por último, la entrada *DefPts* constituye la relación directa con todos los vértices o puntos a excluir en cada uno de los edificios del sistema urbano, con un método de entrada *Point3d*.

Los parámetros de salida serán el resultado de la ejecución interna que lleve a cabo el *Script*, a partir de la sintaxis de relación geométrica y de programación en lenguaje *Python.* Es por eso que, al ejecutar el script de manera satisfactoria, obtendremos como primer dato de salida, coordenadas de líneas de campo, el segundo dato serán los puntos de control de las líneas, el tercero representará a los puntos de exclusión en el sistema urbano por medio de esferas instanciadas en el componente *Brep* (Símbolos + Códigos) y, por último, la cuarta salida producirá a todos los agentes de atracción, generados a partir del movimiento de los seres humanos (agentes o puntos iniciales) y siendo una analogía de las ilusiones y deseos, representados en el vacío urbano artificial y virtual.

Es momento entonces, de comenzar a explicar cada una de las partes de la sintaxis del código de control y generación de flujos basado en agentes, para ello, dividiremos cada parte del código basándonos en la función que tiene cada grupo de la sintaxis.

01 – Importación de librerías

```
1   import rhinoscriptsyntax as rs
2   import random as rnd
3   import ghpythonlib.components as gc
4   import Rhino.Geometry as rg
5   import Grasshopper.Kernel as gk
```

Los estatutos de importación ***import*** **()** son los siguientes:

- *rhinoscriptsintax as rs*: incorporación de todos los comandos de *Rhinoceros 5.0* mediante la abreviatura *rs*, con el propósito de controlar datos y variables dentro de un *Script* de programación.
- *random as rnd:* incorporación de valores numéricos aleatorios, mediante la abreviatura *rnd*, con el propósito de asignar control numérico, referido a la generación de coordenadas de puntos y variación en la posición de elementos geométricos en el espacio tridimensional.
- *ghpythonlib.components as gc:* incorporación de librería de los componentes propios de *Grasshopper*, mediante la abreviatura *gc* y con el propósito trabajar en conjunto con el lenguaje de programación *Python*.
- *Rhino.Geometry as rg:* incorporación de la librería de geometría propia del programa *Rhinoceros 5.0*, al representar a cada objeto geométrico en el espacio tridimensional; es importada ya que toda la geometría será representada en este entorno de trabajo. La abreviatura *gc* nos permitirá acceder a cada comando de control y generación de una manera más sencilla.
- *Grasshopper.Kernel as gk:* incorporación del núcleo de programación propio de *Grasshopper*, mediante la abreviatura *gk* y con el propósito de controlar, mediante una sintaxis de programación textual, cada parámetro de los componentes de producción de líneas de campo y vectores, en el entorno tridimensional de *Rhinoceros 5.0*.

02 – Variables de entrada e instancias de salida basadas en métodos

```
7   # --- SALIDAS DE CAMPO ---
8   field = gk.Types.GH_Field()
9   # ------------------
10  Decay = 2
```

Al importar librerías como abreviaturas de cada una, seremos capaces de acceder a todos sus componentes al escribir las letras correspondientes explicadas anteriormente. En este caso asignaremos y produciremos variables de inicio que relacionaremos como salidas a través de métodos de programación, representados como *()*.

La primera variable es *field* (campo), que representa a los datos de una línea de campo mediante un tipo de geometría particular de *Grasshopper*, es por eso que se define como *gk.Types.Gh_Field(),* que es un método que dependerá de atributos a asignar y desarrollar a lo largo de la sintaxis del código. Por otra parte, *Decay* representa el grado de curvatura de cada una de la línea de campo, mientras este valor sea mayor, las líneas o curvas serán más definidas o más “suaves”, mientras menor, las curvas serán más ortogonales, en este caso el valor es *2*, un valor intermedio que no demanda tantos recursos de hardware para su compilación y ejecución posterior.

Una vez definidas las variables de entrada, vamos a empezar a asignar atributos para generar a los datos de salida basados en instancias de un método.

03 – Generación de entradas-instancia y listas de puntos iniciales

```
12  # PUNTOS OBJETIVOS: PUNTOS /// ESFERAS /// GEOMETRIA POBLADA 3D /// PUNTOS CARGADORES
13
14  #---ENTRADAS---
15  TgPt
16  NumOfTgFL
17  TgPtChValue
18  Seed
19  #-----------
20
21  TgSpRadii = 3
22  TgPtList = []
23  TgSpList = []
24  TgSp_PG_List = []
```

Se definen las instancias de los componentes de control numérico exteriores (*sliders*) y *Point* en relación al código, para asignar las variables *TgPt* + *NumOfTgFL* + *TgPtChValue* + *Seed* (línea 15 – 18).

La variable *TgSpRadii* es definida para controlar el radio de las esferas a poblar a partir de los agentes iniciales, con un radio en cada una de ellas con *3* unidades. Las variables *TgPtList (Lista de agentes iniciales)* + *TgSpList (Lista de esferas)* + *TgSp_PG_List (Lista de Esferas poblada con puntos),* son definidas como listas *[]*, con el fin de asignar posteriormente información en cada una de ellas al generar atributos de control de datos y geométricos en líneas posteriores, mediante atributos de compactación *[X.x()].*

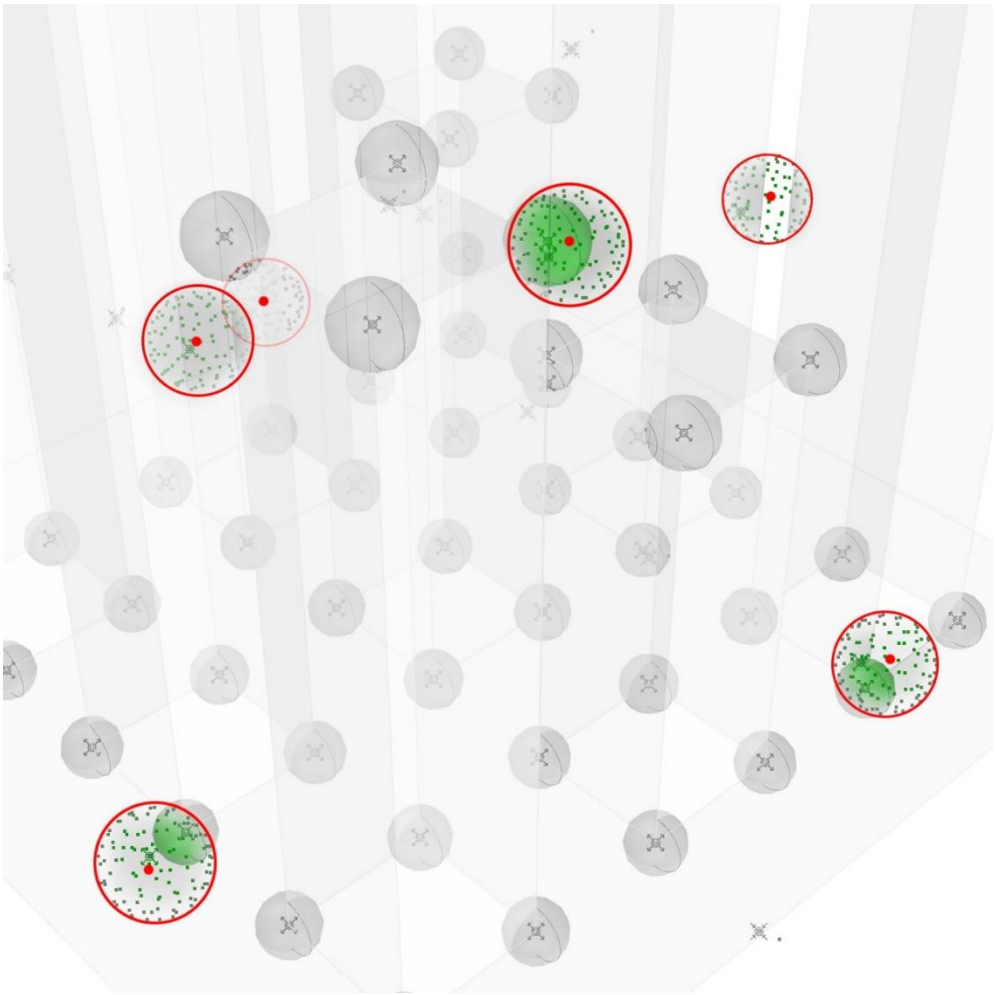
04 – Población de geometría inicial y generación vectorial inicial

```
26  for i in range(len(TgPt)):
27  ····TgPt_Li = gc.ListItem(TgPt,i)
28  ····print TgPt_Li
29  ····# ESFERAS
30  ····TgSp = gc.Sphere(TgPt_Li,TgSpRadii)
31  ····TgSpList.append(TgSp)
32  ····# GEOMETRIA POBLADA
33  ····TgSp_PG = gc.PopulateGeometry(TgSp,NumOfTgFL,Seed)
34  ····for i in range(NumOfTgFL):
35  ····· ····TgSp_PG_Li = gc.ListItem(TgSp_PG,i)
36  ····· ····TgSp_PG_List.append(TgSp_PG_Li)
37  ····# PUNTOS CARGADORES
38  ····TgPtCh = gc.PointCharge(TgPt_Li,TgPtChValue,Decay)
39  ····field.Elements.AddRange(TgPtCh)
40
41  # --- SALIDAS ---
42  TgSp_PG_List
43  #------------
```

Nuestro objetivo particular en este paso es generar esferas a partir de los agentes iniciales del primer Script, y controlar el radio de cada una ellas (*TgSpRadii = 3*). Por lo anterior se define un rango *for i in range(len(TgPt)*, que nos permite acceder a cada agente o punto inicial basado en el número de agentes; se define entonces la variable *TgPt_List = gc.ListItem(TgPt, i),* que depende de los puntos y del rango de los puntos, es decir, *6*; con lo anterior, obtenemos acceso a cada elemento de manera individual y no colectiva como en el rango. En las líneas 30 – 31 se definen las variables para producir la representación de las esferas a partir del rango de cada uno de los puntos y se reducen a esferas mediante *TgSpList.append(TgSp),* eliminando las coordenadas y datos de los puntos generadores.

En las líneas 33 – 36 se generan puntos en las caras exteriores de las esferas, a través de la variable *TgSp_PG = gc.PopulateGeometry(TgSp, NumOfTgFL, Seed)*, y que dependen del número de agentes iniciales y *Seed* para controlar la generación aleatoria de puntos. Posteriormente se asigna un rango a NumOfTgFL, que controla el número de datos de cada coordenada en una línea de campo, para posteriormente, ser instanciada como lista en base al valor índice establecido en el rango y acceder a cada uno de los puntos generados en las esferas. Se compacta la geometría a solo puntos que se localizan en las caras de todas las esferas mediante *TgSp_PG_List.append(TgSp_Li).*

Una vez definidos los puntos en las esferas, se generan las coordenadas de inicio de los vectores de repulsión (líneas 38 – 42), a través de la variable *TgPtCh = gc.PointCharge(TgPt_Li, TgPtChValue, Decay),* que depende de todos los puntos, de un valor de carga asignado anteriormente mediante un *slider* del mismo nombre y un valor de decaimiento (2) asignado en el principio del código. Se define, entonces, una variable de compactación de datos vectoriales de repulsión basada en puntos de carga que dependen de las esferas *field. Elements.AddRange(TgPtCh),* que será controlado a partir de un rango generado por el componente *gc.PointCharge*. Por último, se define el primer parámetro de salida (línea 42 - *TgSp_PG_List),* con el propósito de representar en el espacio tridimensional a las esferas y a los puntos iniciales vectoriales de repulsión.



04A. Esferas basadas en agentes iniciales y pobladas por puntos de repulsión vectorial [Cargadores].

05 - Generación de entradas-instancia y listas de puntos deflectores [Exclusión]

```
45  # DEFLECTORES: PUNTOS /// ESFERAS /// GEOMETRIA POBLADA /// PUNTOS CARGADORES
46
47  # --- ENTRADA ---
48
49  # DefPts ES DEFINIDO PARA ALMACENAR A TODOS LOS PUNTOS DEL SISTEMA URBANO
50  DefPts
51  NumOfDefPop
52  DefPtChValue
53  # -----------
54
55  DefSpRadii = 10
56  DefPtList = []
57  DefSpList = []
58  DefSpPG_List = []
```

Se definen las instancias de los componentes de control numérico exteriores (*sliders*) y los puntos resultantes de *DeBrep* + *CullDuplicates* (*DefPts*) en relación al código, para asignar las variables *NumOfDefPop* + *DefPtChValue* (línea 50 – 52).

La variable *DefSpRadii* es definida para controlar la magnitud de la fuerza de repulsión en cada uno de los puntos, que se localizarán en las caras exteriores las esferas, a partir de los vértices de exclusión en los edificios del sistema urbano virtual (*DefPts*). Las variables *DefPtList (Lista de agentes de exclusión)* + *DefSpList (Lista de esferas)* + *DefSp_PG_List (Lista de Esferas poblada por puntos de exclusión vectorial),* son definidas como listas *[]*, con el fin de asignar posteriormente información en cada una de ellas al generar atributos de control de datos y geométricos en líneas posteriores, mediante atributos de compactación *[Y.y()].*

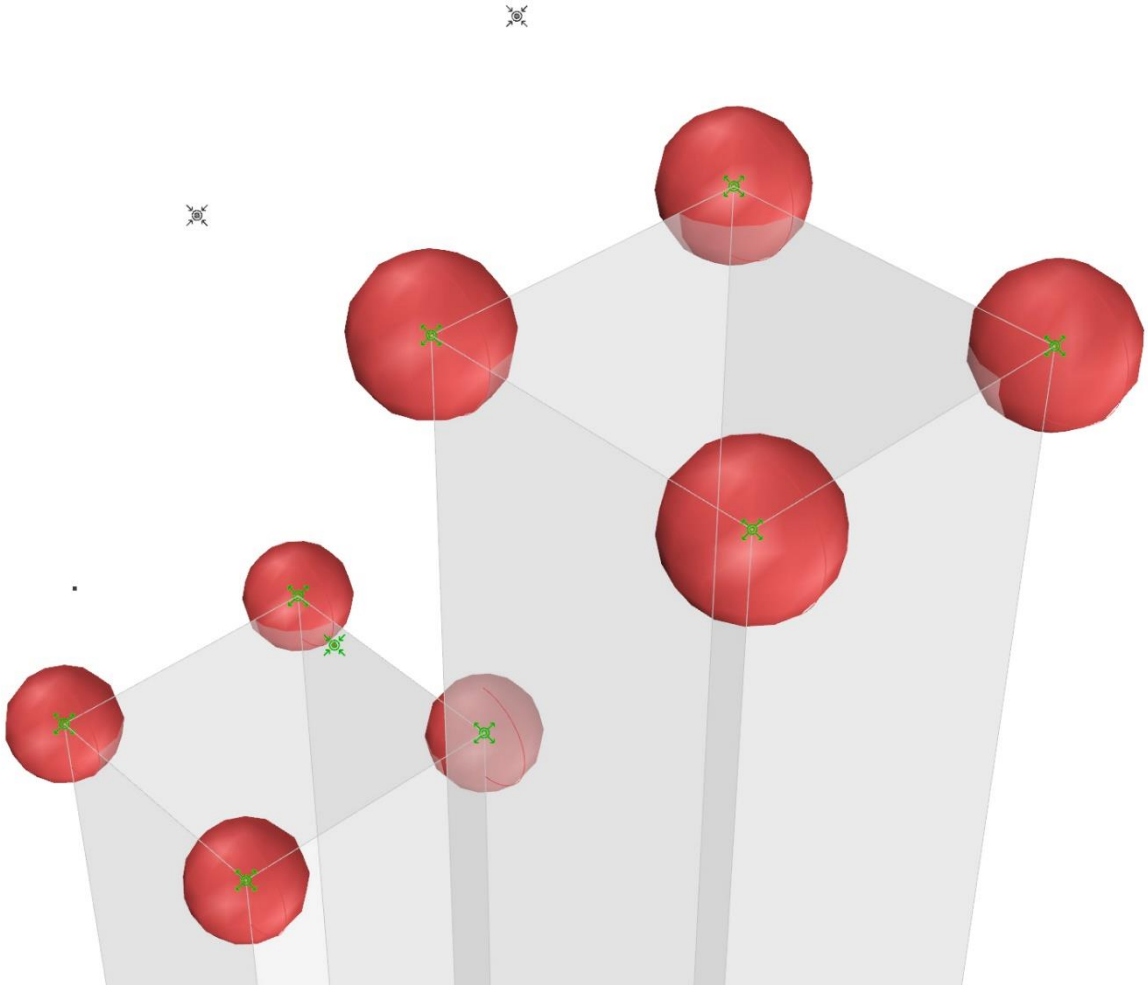
06 – Población de esferas y generación puntos deflectores de exclusión vectorial

```
60  # ESTE PASO CONVIERTE A TODOS LOS VERTICES DE LOS EDIFICIOS EN ESFERAS QUE SON POBLADAS POR PUNTOS
61  DefSp = gc.Sphere(DefPts, 2)
62
63  # PUNTOS CARGADORES
64  for i in range(len(DefPts)):
65      DefPts_Li = gc.ListItem(DefPts,i)
66      DefPtCh = gc.PointCharge(DefPts_Li,DefPtChValue,Decay)
67      field.Elements.AddRange(DefPtCh)
```

En la línea 61 se define la variable *DefSp = gc. Sphere(DefPts, 2)*, la cual produce esferas, tomando como punto central de generación a los puntos o vértices de los edificios (*DefPts*) definidos como entrada y método *Point3d* en el componente *GhPython*; el radio de cada esfera será de 2 unidades por lo que la magnitud de la fuerza de repulsión será de 8 unidades, al contemplar el límite máximo como 10 unidades – 2 unidades = 8 unidades.

Se definen entonces, los puntos como cargadores de datos vectoriales de repulsión, a través de la incorporación de un rango *for i in range(len(DefPts),* el cual genera un valor índice de control para todos los puntos, basado en la longitud existente entre el punto central de cada esfera y los puntos que localizan en el límite de generación a 10 unidades. Posteriormente, se desarrollan las variables *DefPts_Li = gc.ListItem(DefPts, i)* + *DefPtCh = gc.PointCharge(DefPts_Li, DefPtChValue, Decay)* para acceder a cada uno de los puntos de las esferas y asignar un valor vectorial de repulsión mediante un *slider* de entrada, además del mismo valor de decaimiento, con el fin de lograr mayor control en las curvaturas de las líneas de campo, al establecer un valor unitario. Por último, se compacta y resume

la geometría generada a un rango que representa a vectores de repulsión, por lo que habrá una exclusión de los edificios al generar las líneas de campo.



06A. Vectores de repulsión, a partir de todos los vértices de los edificios del sistema urbano virtual.

07 - Generación de entradas-instancia y listas de puntos [Atracción]

```
69  # ATRACTORES: PUNTOS /// PUNTOS CARGADORES
70
71  # --- ENTRADA ---
72  UrbanSystem
73  NumOfAtt
74  AttPtChValue
75  # -----------
76
77  AttPtList = []
78  PIB_List = []
```

Se definen las instancias de los componentes de control numérico exteriores (*sliders*) para las variables *NumOfAtt* + *AttPtChValue*, además de la instancia del sistema urbano (*componente Brep* = *UrbanSystem*) en relación al código (línea 72 – 74).

Las variables *AttPtList (Lista de agentes atractores)* + *PIB_List (Lista de coordenadas de puntos atractores),* son definidas como listas *[]*, con el fin de asignar posteriormente información en cada una de ellas al generar atributos de control de datos y geométricos en líneas posteriores, mediante atributos de compactación *[Z.z()].*

08 – Puntos atractores y coordenadas de líneas de campo

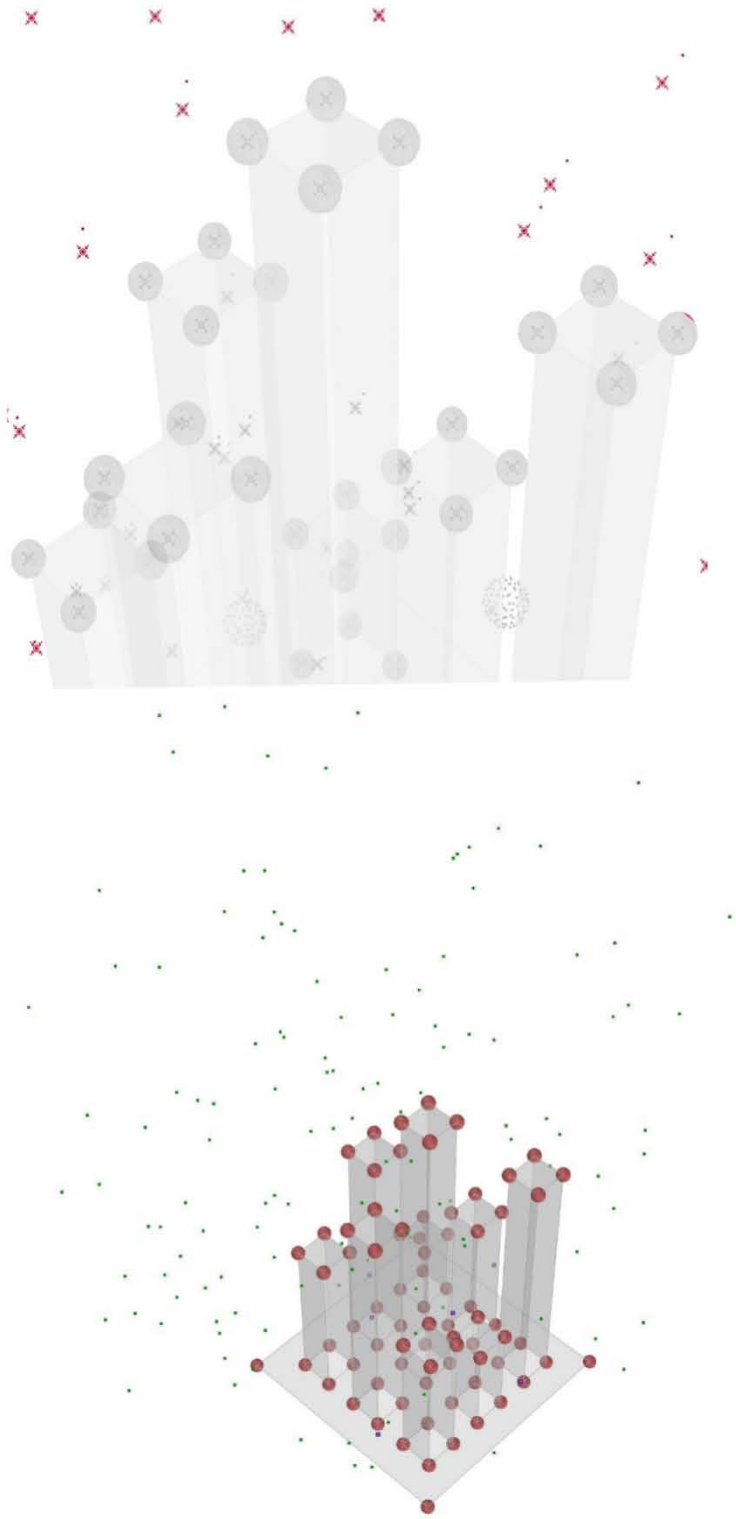
```
80  ····# PUNTOS ATRACTORES EN EL EXTERIOR DE LOS EDIFICIOS
81  for i in range(NumOfAtt):
82  ····Att_x = rnd.randint(-50,75)
83  ····Att_y = rnd.randint(-50,75)
84  ····Att_z = rnd.randint(0, 200)
85  ····AttPts = gc.ConstructPoint(Att_x,Att_y,Att_z)
86  ····PIB = gc.PointInBrep(UrbanSystem, AttPts)
87  ····if not PIB  :
88  ····· ····AttPtList.append(AttPts)
89  ····· ····PIB_List.append(PIB)
90  ····· ····AttPtCh = gc.PointCharge(AttPts,AttPtChValue,Decay)
91  ····· ····field.Elements.AddRange(AttPtCh)
```

La generación de puntos atractores dependerá de un número de control que está relacionado con el *slider* de entrada (*NumOfAtt*), por lo que a partir de este número, se podrán llevar a cabo iteraciones de producción de puntos aleatorios al importar *rnd* (random) y el comando *randit*, para asignar únicamente valores enteros.

En las líneas (81 – 86) se define un rango basado en el número de puntos atractores que deseamos incorporar, además de definir los límites de generación de puntos en relación a un control numérico para asignar las coordenadas en los ejes x + y + z. En este caso, la generación de puntos es definida por medio de las variables *Att_x* + *_y* + *_z,* que controlarán la generación de puntos en todo el sistema urbano de manera aleatoria. A partir de estas coordenadas, se define la variable *AttPts = gc.ContructPoint(Att_x, Att_y, Att_z),* para representar a todos los puntos en el espacio tridimensional. Posteriormente se define la variable *PIB = gc.PointInBrep (UrbanSystem, AttPts)* para eliminar y no generar puntos en el interior de los edificios.

Por último, en las líneas (87 – 91) se realiza una ejecución condicional, en dónde se especifica que, si los puntos se encuentran en el interior de los edificios, no se compacten para producir las coordenadas de las líneas de campo y, por otra parte, que, al generarse

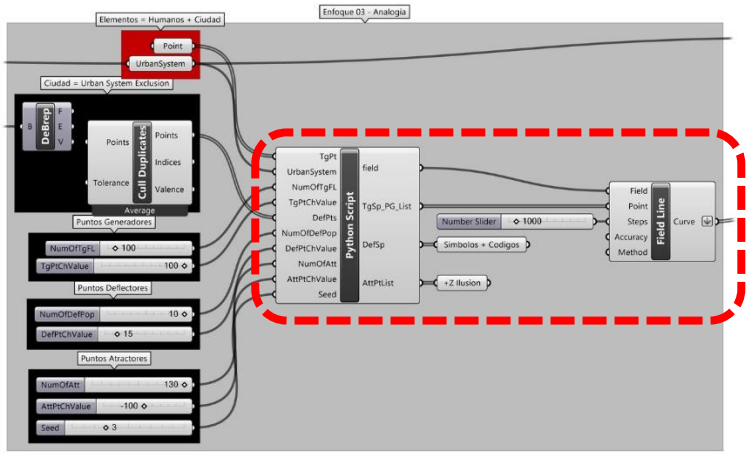
la exclusión de puntos en el exterior de los edificios, la compactación y producción de las coordenadas se lleve a cabo, al igual que los vectores de atracción mediante *PointCharge*.

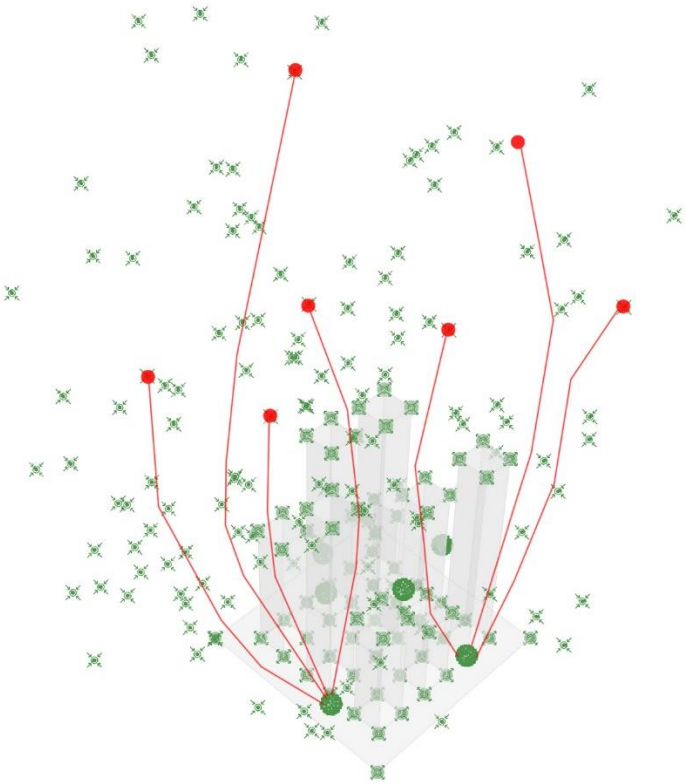


08A. Agentes de atracción vectorial en el exterior del sistema urbano y exclusión de edificios.

Al definir y representar a los puntos atractores, hemos colocado a los elementos principales para la generación de las líneas de campo. Por consiguiente, debemos relacionar a cada punto de repulsión, repulsión + exclusión y atracción con base en las coordenadas de cada punto de las líneas, así como de los puntos para producirlas.

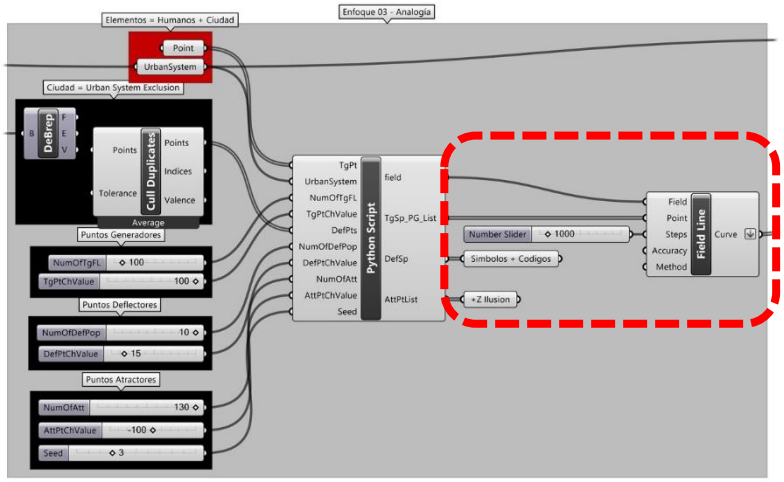
A tomar en consideración son las cuatro salidas que tenemos en el componente de *GhPython*, tales como *field*: coordenadas de los puntos o campos de fuerzas, *TgSp_PG_List:* como puntos generativos de las líneas de campo, *DefSp* que corresponden a los vértices de exclusión en los edificios, representados por medio de esferas y, por último, *AttPtList* que son todos los puntos atractores en el espacio tridimensional, en base a los límites establecidos en el código y localizados en el exterior de todo el sistema urbano.

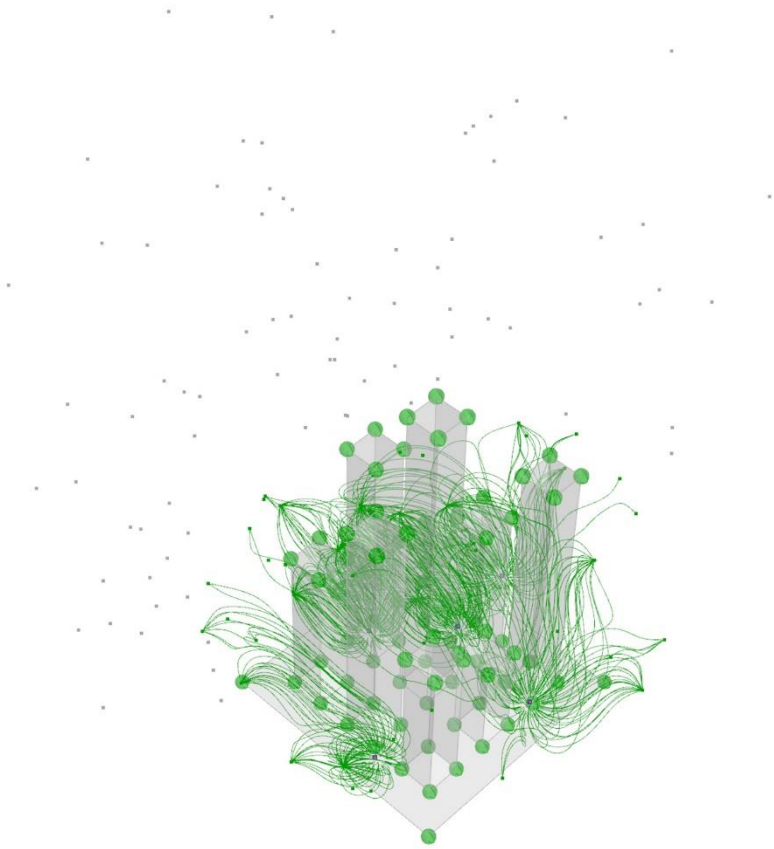




08B. Repulsión 01 + Repulsión 02 [Exclusión] + Atracción = Líneas de campo

Cuando relacionamos a las salidas del *Script* de *Python* (*GhPython)* con las entradas del componente de *Grasshopper – Field Line*, podremos representar y observar a las líneas de campo generadas en el entorno tridimensional de Rhinoceros 5.0.





08C. Líneas de campo iniciales, en base a agentes de fuerzas vectoriales.

El componente *Field Line* se ejecuta de manera satisfactoria al asignar los campos de fuerzas, los puntos de las líneas y un parámetro de control numérico de *1000*, referido a los pasos (*Step*) para definir el número de puntos generados a partir de las líneas y que proporcionan mayor definición en la representación tridimensional, además de activar la

condición de *Flatten*, con el fin de obtener un solo objeto conformado por entidades individuales.

Sin embargo, las líneas de campo se introducen en el interior de los edificios, a pesar de restringir la producción de puntos de fuerzas en el interior (*PointInBrep*), esto se debe a que tales puntos solo son necesarios para generar las fuerzas, pero no constituyen los vértices de las líneas (figura 124). Por lo anterior, se desarrollará otro *Script* basado en el lenguaje de programación *Python*, para excluir a las respectivas líneas de campo y representar a aquellas que queden exentas de todos los edificios del sistema urbano.

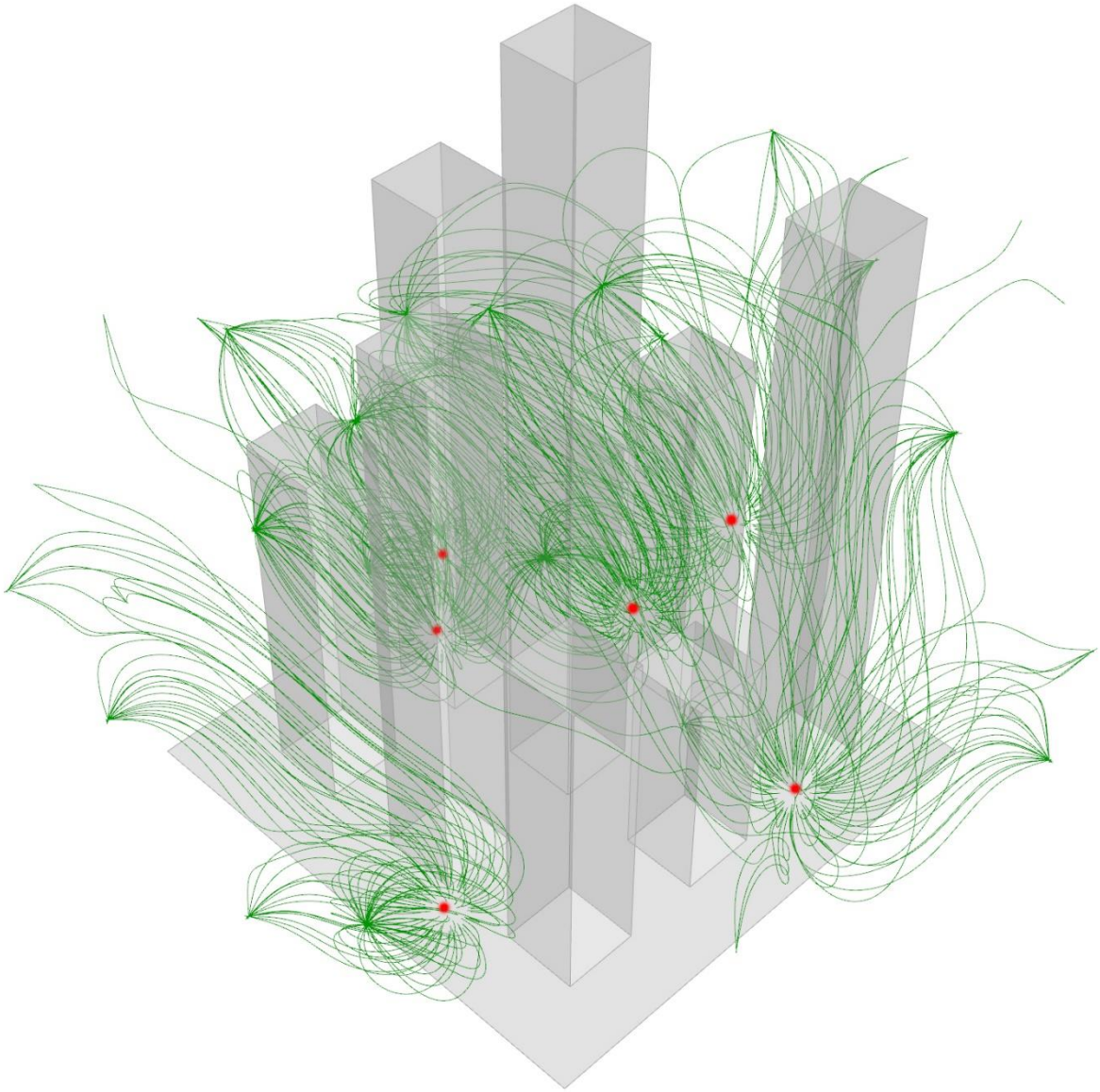
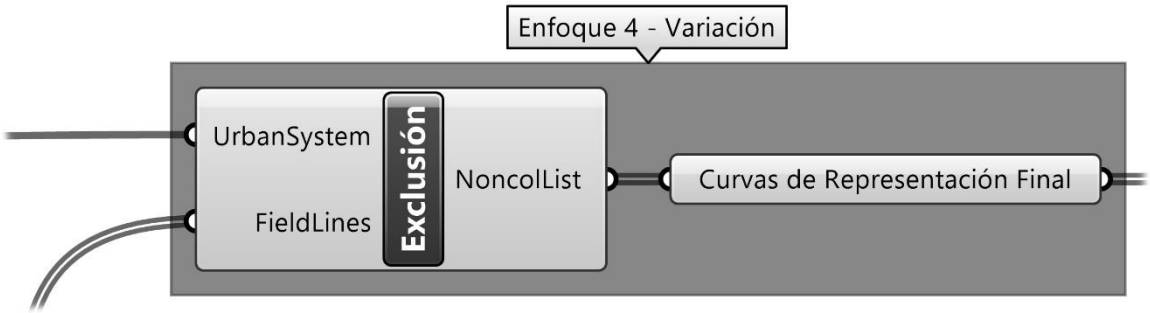


Figura 124. Representación de flujos en base a líneas de campo, sin exclusión de edificios en el sistema urbano.

5.6.3 – 3ra Etapa - Medio + Enfoque [4 – Variación]

Al programar otro *Script* basado *Python*, con el propósito de eliminar a las líneas en el interior de cada edificio, es necesario tomar en cuenta la importación de las librerías correspondientes, así como definir las entradas correspondientes en el componente *GhPython*, en este caso denominadas *UrbanSystem* (Geometría de exclusión) y *FieldLines* (Geometría a excluir); además de una salida relacionada con una lista de las curvas que se localizan en el exterior de los edificios (*NoncolList*).

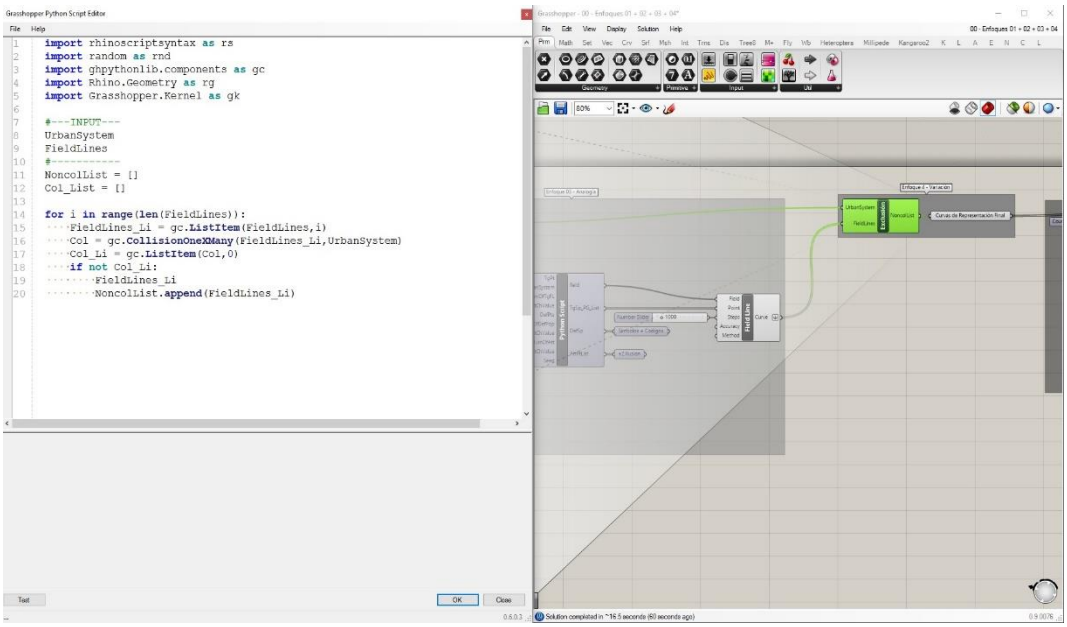


01A. El componente *GhPython* (Exclusión) y el componente *curve* (Curvas de Representación Final).

UrbanSystem corresponde a todo el sistema urbano en dónde se generaron todos los agentes iniciales mediante una simulación aleatoria, *FieldLines* son las líneas de campo producidas en el *Script* de *Python* anterior. Por otra parte, la salida *NoncolList* serán todas las curvas que se localizarán en el exterior de los edificios, propiciando por consecuencia, una variación en la representación de los flujos finales.

A] Eliminación de líneas de campo y generación de curvas

El tercer *Script* de *Python* excluirá a las líneas de campo dentro de los edificios y producirá curvas para representar a los flujos exteriores de manera más clara. La sintaxis se explica a continuación:



01B. *Script* basado en *Python* para excluir líneas de campo y representar una variación a través de curvas.

01 – Importación de librerías

```
1  import rhinoscriptsyntax as rs
2  import random as rnd
3  import ghpythonlib.components as gc
4  import Rhino.Geometry as rg
5  import Grasshopper.Kernel as gk
```

Los estatutos de importación (*import*) son los mismos que en anterior código, ya que, al formar parte de un sistema paramétrico, las instancias deberán permanecer de manera similar para evitar cualquier error de ejecución o generación de geometría.

02 - Generación de entradas-instancia y listas de líneas + curvas

```
7  #---INPUT---
8  UrbanSystem
9  FieldLines
10 #-----------
11 NoncolList = []
12 Col_List = []
```

Se definen las instancias de entrada a partir de los componentes exteriores (*UrbanSystem + FieldLines*), en relación al código (línea 08 – 09).

Las variables *NoncolList (Lista de curvas exteriores) + Col_List (Lista de curvas interiores en los edificios),* son definidas como listas *[]*, con el fin de asignar posteriormente información en cada una de ellas al generar atributos de control de datos y geométricos en líneas posteriores, mediante atributos de compactación *[A.a()].*

03 – Flujos finales

```
14 for i in range(len(FieldLines)):
15     FieldLines_Li = gc.ListItem(FieldLines,i)
16     Col = gc.CollisionOneXMany(FieldLines_Li,UrbanSystem)
17     Col_Li = gc.ListItem(Col,0)
18     if not Col_Li:
19         FieldLines_Li
20         NoncolList.append(FieldLines_Li)
```

Una vez realizadas las instancias de entrada y definir las variables como listas, el siguiente paso es definir un rango en base a la longitud de cada línea de campo, para ellos utilizamos *for i in range (len (FieldLines, i),* con el fin de generar un valor índice para cada línea. A partir de la línea 15 – 17, se definen las variables *FieldLines_Li = gc.ListItem(FieldLines, i)* que representa la selección de cada línea de campo en base a un valor índice; la variable *Col = gc.CollisionOneXMany(FieldLines_Li, UrbanSystem)* se refiere a la intersección de cada una de las líneas de los flujos en relación a sistema urbano virtual para, posteriormente, asignar la variable *Col_Li = gc.ListItem(Col, 0),* que identifica a aquellos elementos que se encuentran dentro de los edificios, en este caso el número *0* representa un valor booleano que significa verdadero, es decir, que las curvas se encuentran en el interior de la geometría.

A partir de la línea 18 – 20 se asigna una ejecución condicional, en dónde se especifica que, si las líneas, convertidas ahora en curvas, están en el interior de los edificios no se

compacten y representen; por el contrario, aquellas curvas que se encuentren en el exterior, serán compactadas y proporcionarán el medio generador geométrico a través de curvas.

Por último, se producen los elementos ajenos a los edificios en el sistema urbano, por medio del componente *curve* (Curvas de Representación Final), que propicia una consecuencia, referida a un cambio de líneas de campo a curvas que representan a los flujos y a una variación en cuanto al medio de expresión formal (figura 125).

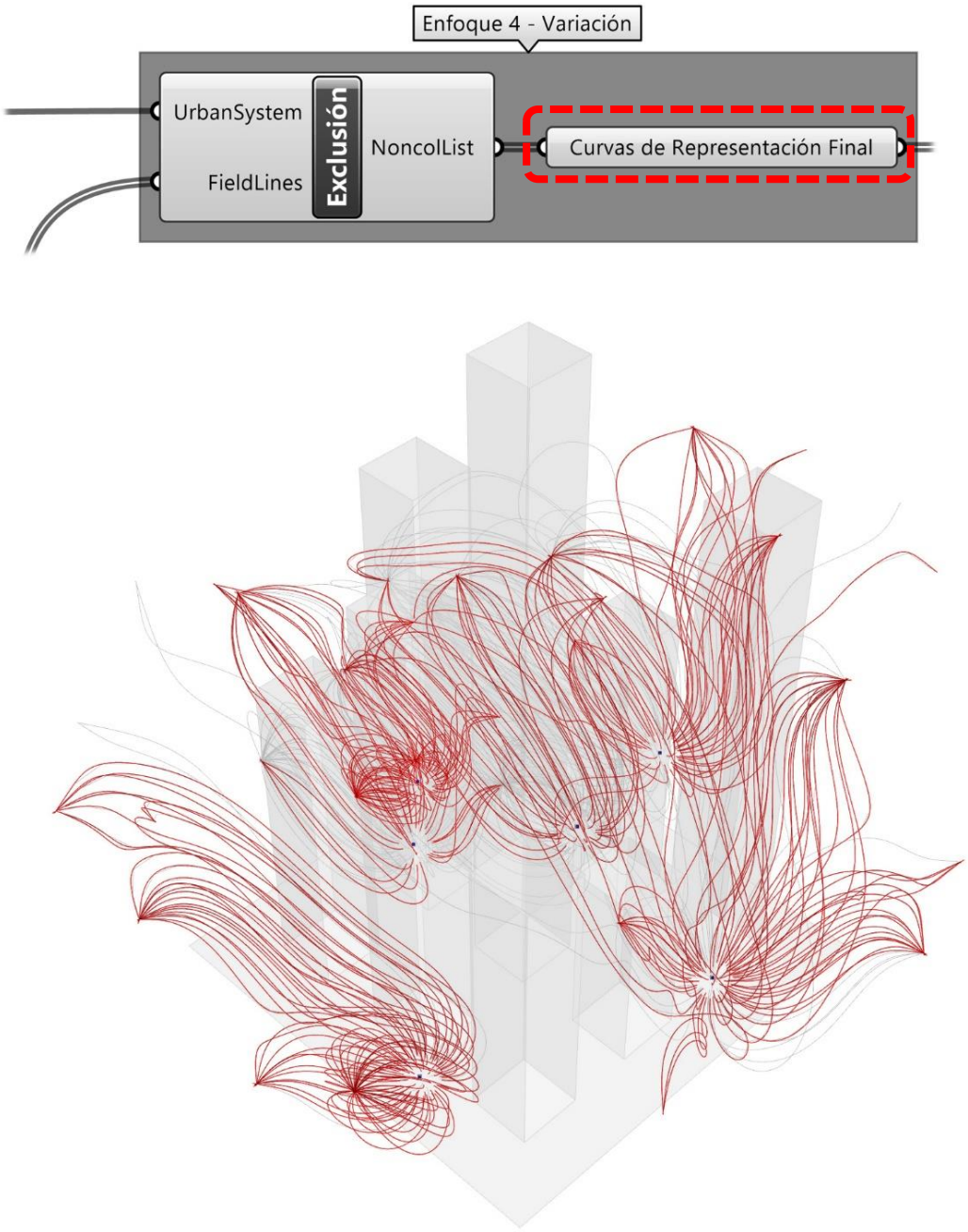


Figura 125. en negro: líneas de campo iniciales, en rojo: curvas de representación de flujos finales.

Este es el último medio de generación basado en código, continuaremos para describir el proceso de representación geométrica basado en mallas celulares, a través del uso del componente *Cocoon* para *Grasshopper*, explicado en el primer caso de estudio.

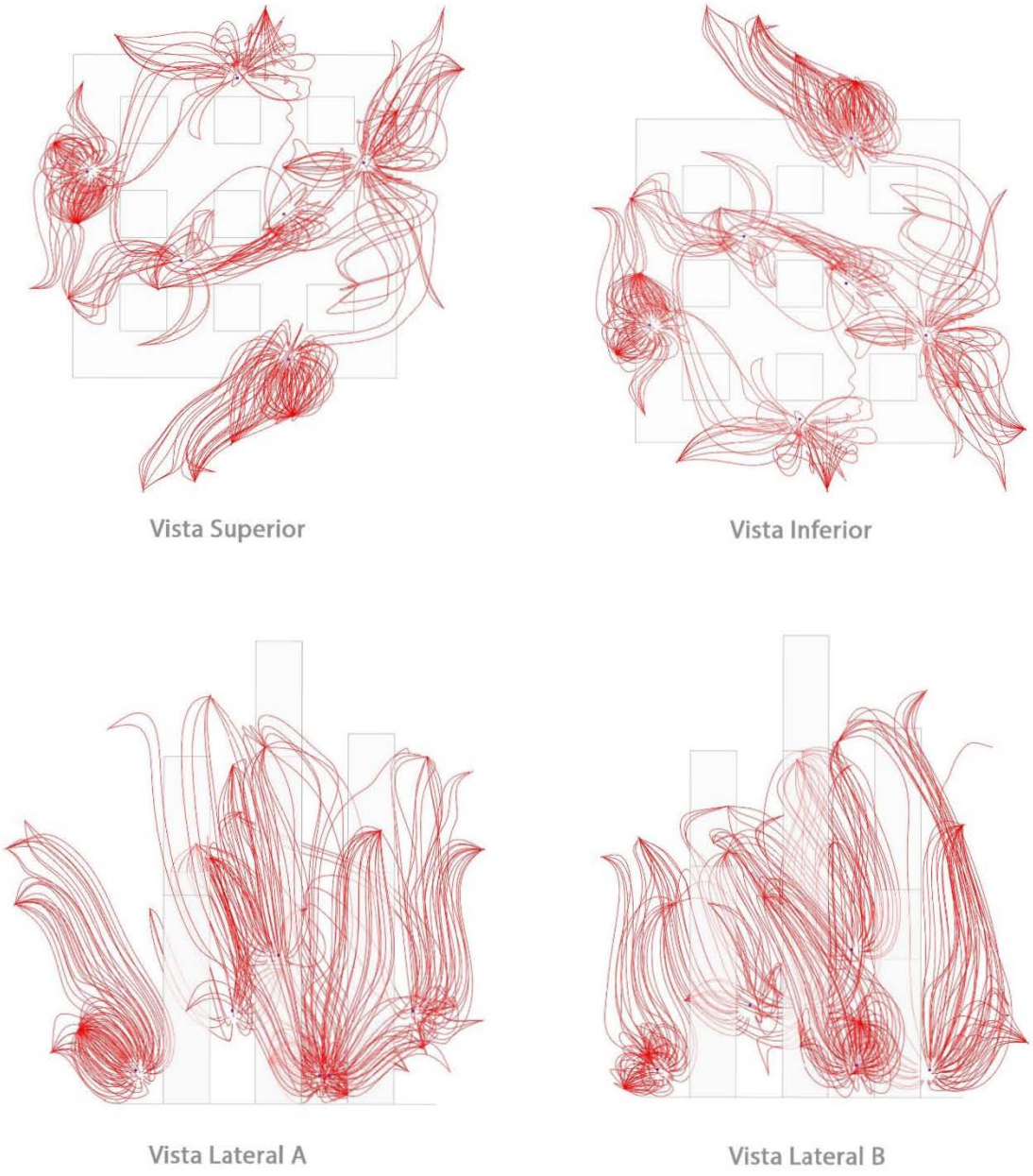


Figura 126. Flujos representados en cada una de las vistas en el sistema de coordenadas cartesiano.

5.6.4 – Representación geométrica final

Con el propósito de ofrecer una mejor representación de los flujos, generados a partir del movimiento aleatorio y simulado de los seres humanos en un entorno virtual, además de una analogía a la ilusión y deseos de cada uno, se desarrolló una gráfica *DAG* con *Grasshopper - Cocoon*, para visualizar de mejor manera al sistema.

Como primer paso se contempló convertir a las curvas finales anteriores, en puntos de generación de polilíneas, para producir mallas optimizadas basadas en comportamiento celular geométrico (figura 127).

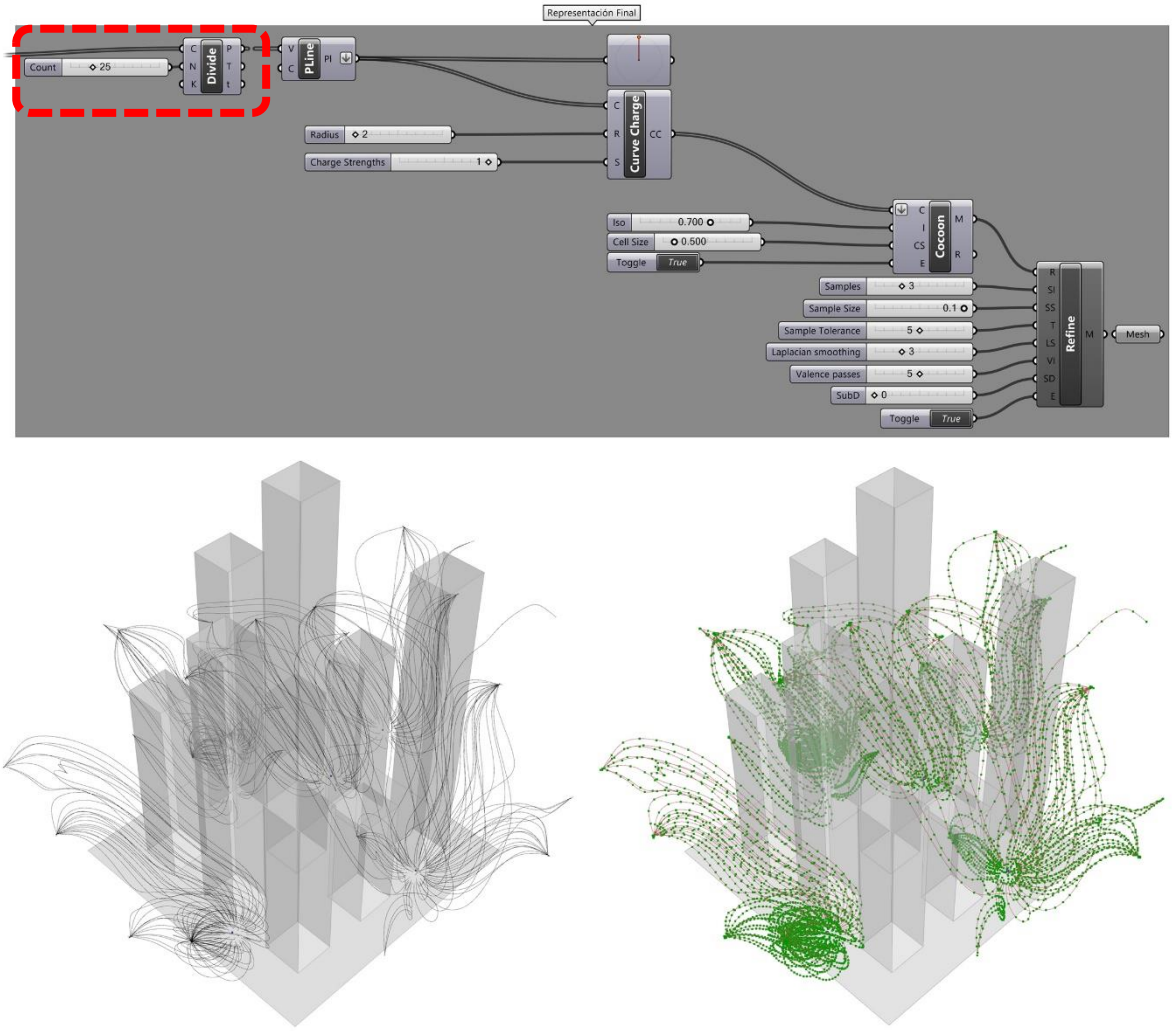


Figura 127. Transformación de curvas a puntos mediante el componente de *Grasshopper - Divide Curve*.

En particular, el *slider* de control numérico (*25*), asigna la cantidad de puntos en cada una de las curvas, mientras el valor sea mayor, la definición de las polilíneas basadas en los puntos de división será mayor, en este caso, este número proporcionará una representación adecuada sin contemplar el uso máximo de recursos de hardware en dónde se desarrolló (figura 128).

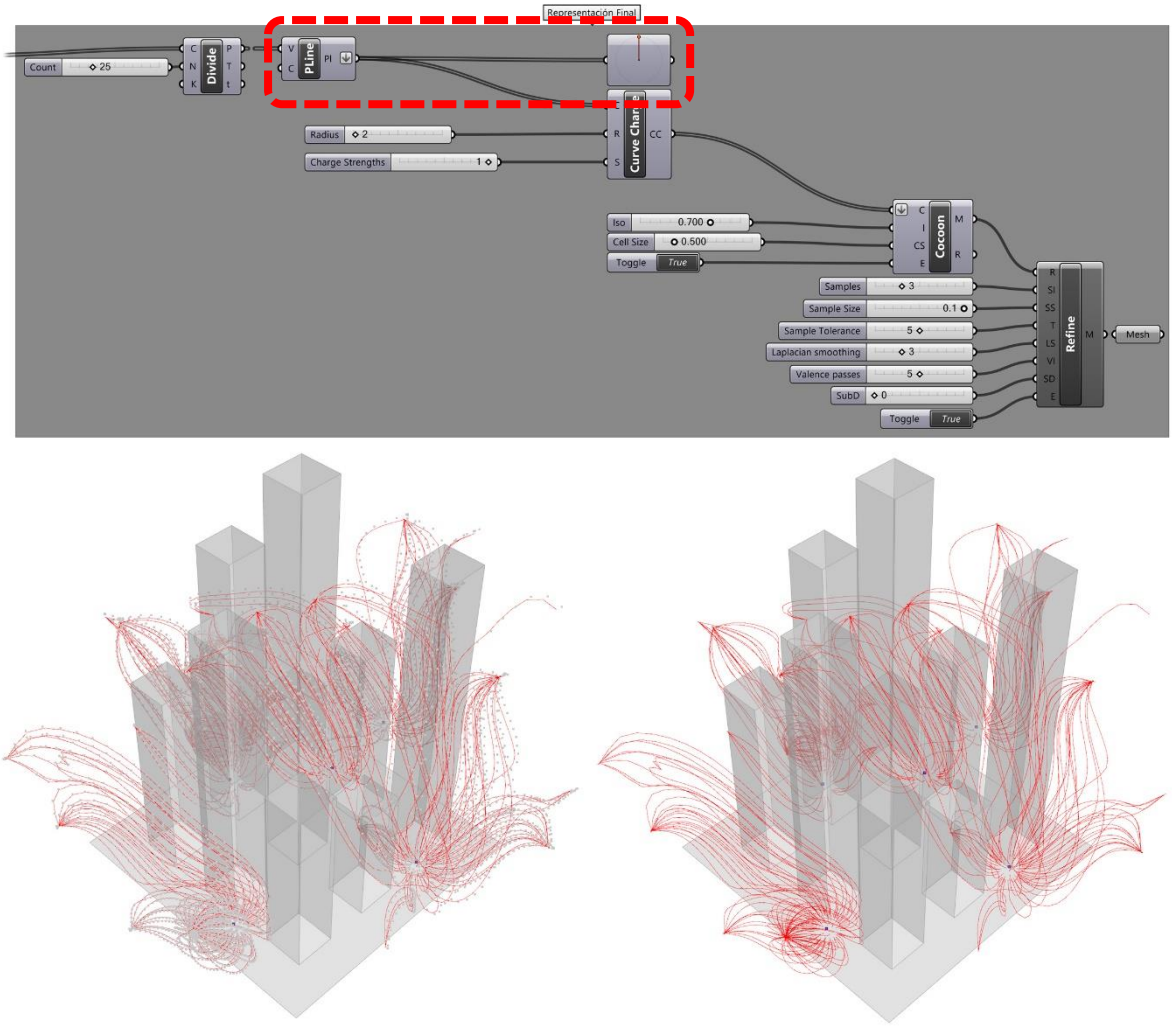


Figura 128. Generación de polilíneas basadas en puntos de control a través de componente de *Grasshopper – PLine.*

Una vez obtenidas las polilíneas, se prosigue a “aplanar” a la geometría final (condición de *Flatten* activa), con el propósito de generar a un objeto de manera unitaria en una lista, pero dependiente de entidades individuales, es decir, de varias polilíneas.

El siguiente y último paso en la representación de este sistema de flujos, consiste en incorporar a los elementos del componente *Grasshopper - Cocoon*, para generar mallas optimizadas, basadas en el principio de células geométricas (figura 129). Los parámetros y variables de control para la producción de las mallas se explican en la siguiente página.

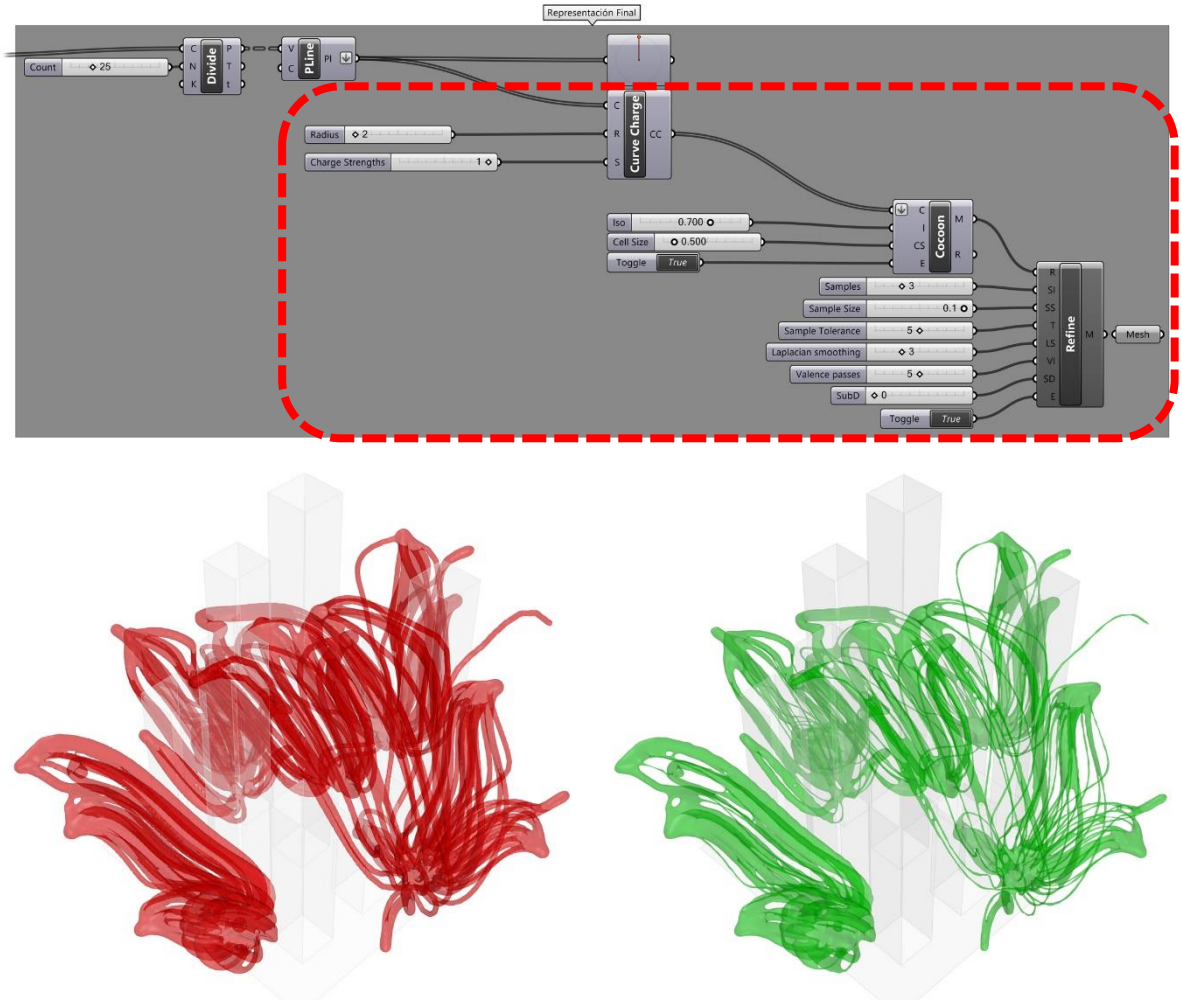


Figura 129. Generación de un sistema de mallas, basadas en el principio de geometría celular con el componente *Cocoon*.

Curve Charge (Curva de Carga), es un componente que nos permite relacionar a las directrices (polilíneas) con datos numéricos de control, referidos a la variación de escala, la cual puede ser llevada a cabo por medio de la relación de la salida *Radius* con la entrada *R* de *Curve Charge*. Con lo anterior, se obtendrá una relación de escala que depende del radio de cada célula. El parámetro de entrada *S* (*Charge Strengths*) controla la intensidad con la que se lleva a cabo está relación a partir de un *slider* y un valor numérico de *1*; para hacer que esta variación sea un poco notoria.

Una vez que tenemos las polilíneas con un parámetro de control generador variable en función del radio de escala, es representado por medio del parámetro de salida *CC* del componente, necesario para establecer la relación para la generación del primer sistema de mallas, a través del componente *Cocoon*, el cual posteriormente, será optimizado. Para generar las mallas primarias, es necesario relacionar las curvas directrices con parámetros de control variables (*CC* de *Curve Charge*) y la entrada *C* del componente *Cocoon*, y restringir la lectura de datos a un elemento unitario, por eso activamos nuevamente la condición de *Flatten* (flecha hacia abajo). Otros aspectos a considerar son los parámetros de entrada *I,* CS y *E* del componente, explicados a continuación:

- ***Iso [I]:*** controla la relación entre las células de generación, mientras este valor sea mayor, las células estarán más cercanas, propiciando la producción de mallas continuas, en este caso, necesitamos un valor alto, por eso lo hemos establecido en 0.70 unidades a través de la relación con un *slider*.
- ***Cell Size [CS]:*** controla el tamaño de las células que se relacionan para generar las mallas continuas. Mientras este valor sea pequeño, las células se podrán observar como esferas independientes a lo largo de las polilíneas. Si el valor es mayor, las células podrán conformar mallas continúas. El parámetro es variable en relación al tamaño del objeto, en este caso, se definió un valor de *0.50* unidades a través de un *slider*, adecuado al tamaño y escala del sistema urbano virtual.
- ***Execute [E]:*** lleva a cabo o no el cálculo para la generación de geometría. Es controlado a partir de un parámetro booleano, *True* es encendido y *False* apagado.

El cálculo y procesamiento de los datos, parámetros y geometría del componente *Cocoon-Grasshopper*, nos brindará la oportunidad de visualizar un resultado formal basado en mallas de generación a partir de células continuas. Este segundo acercamiento geométrico deberá de ser optimizado con otro componente, para variar la cantidad de materia en cada una de los flujos, es decir, mallas irregulares en relación a las polilíneas.

La optimización referida a la generación de geometría, es un proceso que nos permitirá producir mallas más complejas en relación de sus divisiones (teselaciones) y el control, a partir de parámetros de modificación generativos; independientemente de los recursos limitados o no del hardware en dónde se desarrolle. Para poder llevar a cabo la optimización de las mallas geométricas de generación primaria, utilizaremos el componente *Refine* de *Cocoon-Grasshopper*.

Con el propósito de representar y dotar de un aspecto morfológico al sistema de flujos virtual, hemos incorporado un elemento de optimización de geometría, representativo de la cantidad de agentes generadores y puntos de atracción en el sistema. El componente *Refine* optimizará la cantidad de materia a representar por medio de mallas geométricas. Los controles y parámetros necesarios del componente para generar la geometría se explican a continuación:

- ***Refinement Data [R]:*** datos geométricos a optimizar, es decir, la salida *R* del componente *Cocoon*, el cual representa el sistema primario de mallas.
- ***Samples [SI]:*** número de repeticiones en las cuales serán divididas las caras de las mallas en relación a las directrices; el valor asignado en el *slider* es de *3* divisiones máximas, con el fin de equilibrar procesamiento y la visualización geométrica en 3D.
- ***Sample Size [SS]:*** el tamaño de las muestras, se refiere al tamaño de las células de generación de mallas continúas, si el valor es bajo, la relación de cada célula con otra será muy cercana, propiciando una malla continúa; si el valor es grande, la separación será más mayor, generando un sistema de mallas no continuo.
- ***Sample Tolerance [T]:*** generación de continuidad en mallas a partir de la proximidad de las células, si el valor es grande, habrá mayor espacio para llevar a cabo la relación, si es bajo, no habrá espacio y ni continuidad. El valor definido en el *slider* es de *5*, de gran continuidad espacial.
- ***Laplacian Smoothing [LP]:*** control de la resolución de las mallas. El valor definido en el *slider* es de *3*, ya que se requiere una alta resolución en la visualización final.
- ***Valence Passes [V]:*** controla la sustracción de geometría en las mallas, en este caso, el valor definido a través de un *slider* es de *5*, con el fin de obtener una malla irregular, pero continúa en las teselaciones que la producen.

- ***SubD [SD]:*** incremento de las subdivisiones en las mallas, en este caso es definido en un *slider* con el valor de *0*, para evitar el lento procesamiento de geometría final.
- ***Execute [E]:*** lleva a cabo o no el cálculo para la generación de geometría. Es controlado a partir de un parámetro booleano, *True* es encendido y *False* apagado.

Figura 130. Representación Geométrica Final.

En esta última etapa hemos representado a un sistema de flujos generado a partir de agentes (seres humanos) y analogías, referidas a los deseos e ilusiones como medios de producción geométrica en el vacío de un entorno artificial y virtual de hábitat, además de hacer una relación de materia celular, con el fin de representar a un sistema de mallas optimizado según el número de agentes, mientras los agentes iniciales sean mayores, la cantidad de materia será mayor y viceversa (figura 131). Por otra parte, al ejecutar nuevamente cada uno de los *Scripts* basados en el lenguaje de programación textual *Python*, las configuraciones geométricas siempre serán variables, ya que cada situación es temporal y no se repite, con lo cual se procede al abandono de la simulación por la recreación y análisis visual de datos, cumpliendo con la condición de generación basada en el principio de espacio – tiempo (figura 132).

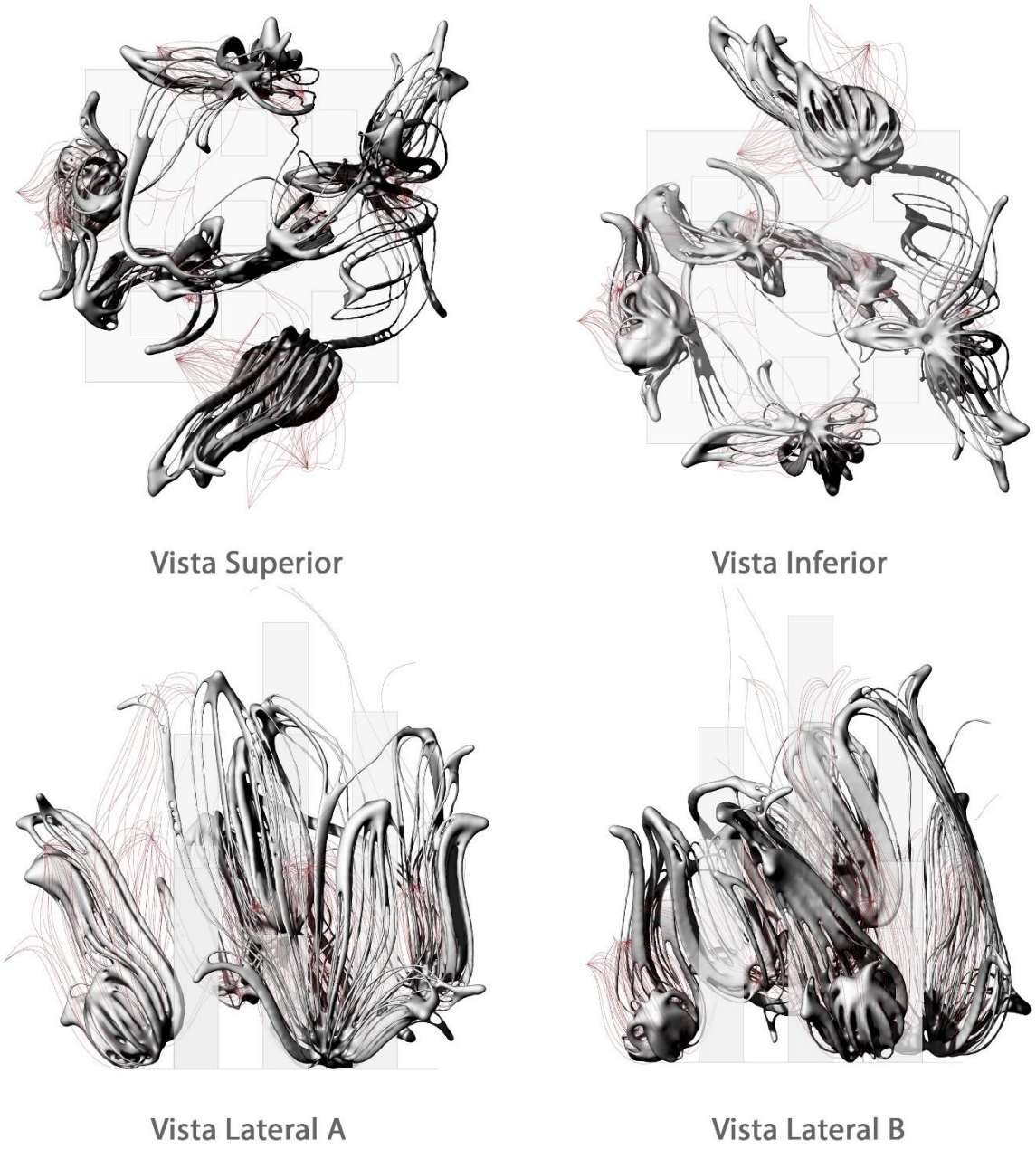


Figura 131. Representación geométrica final en cada una de las vistas de sistema de coordenadas cartesiano, basada en mallas de optimización celular, generadas mediante *Grasshopper + Python + Cocoon.*

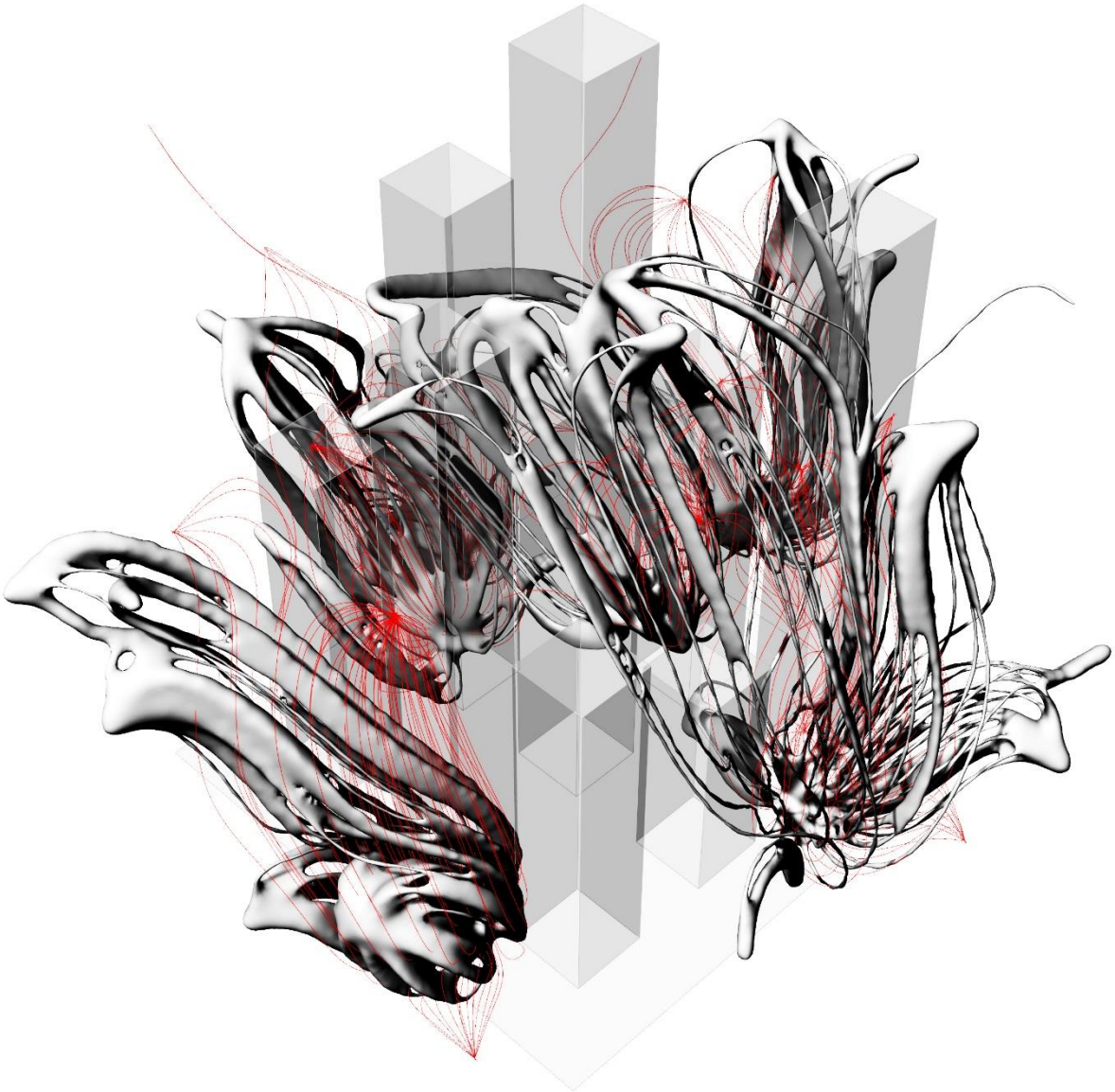


Figura 132. Representación geométrica final y de variación formal en un sistema de generación temporal, basada en situaciones específicas y no repetidas del espacio – tiempo; mediante *Grasshopper* + *Python* + *Cocoon.*

Figura 133. Vista Superior Final del Sistema

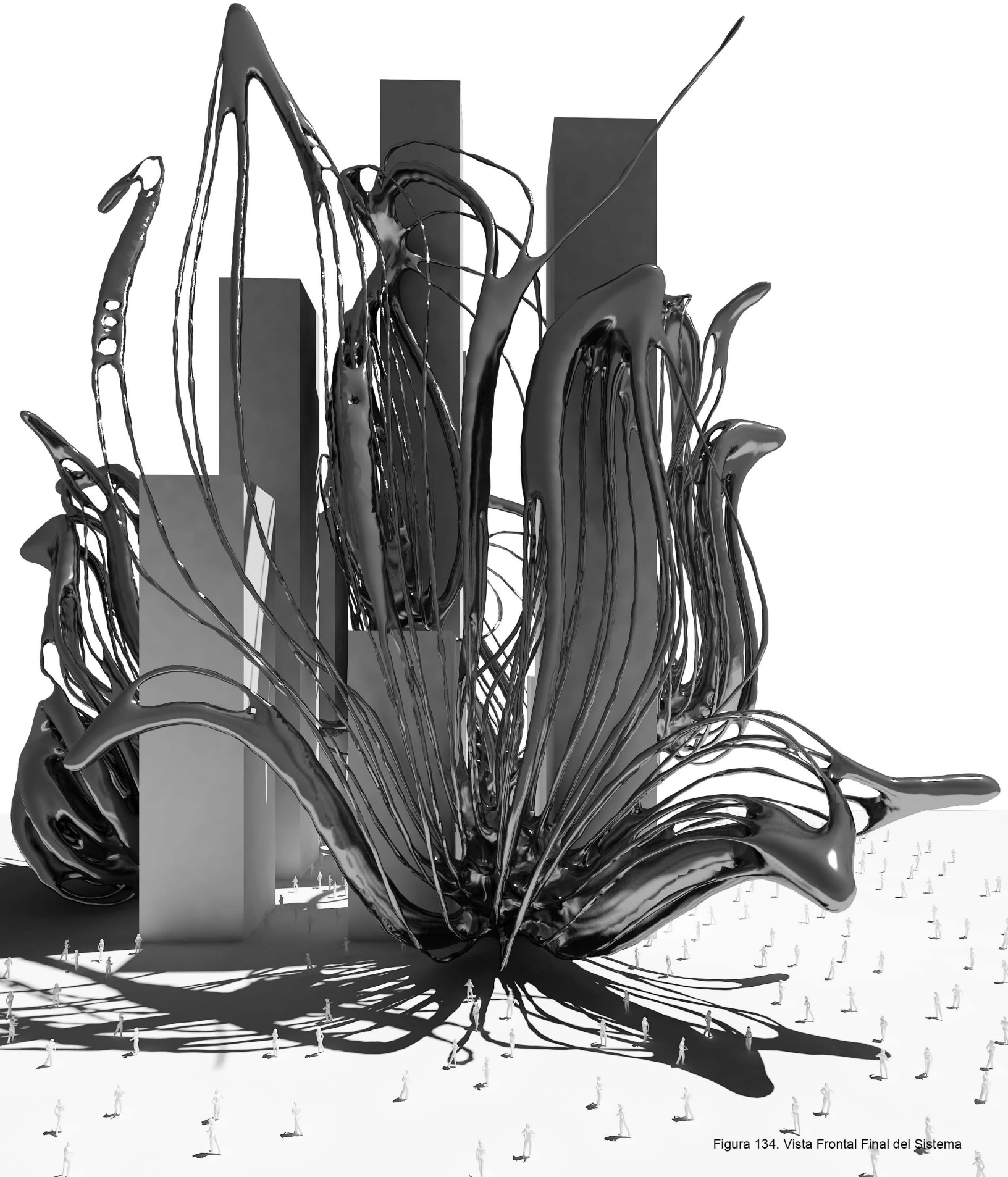


Figura 134. Vista Frontal Final del Sistema

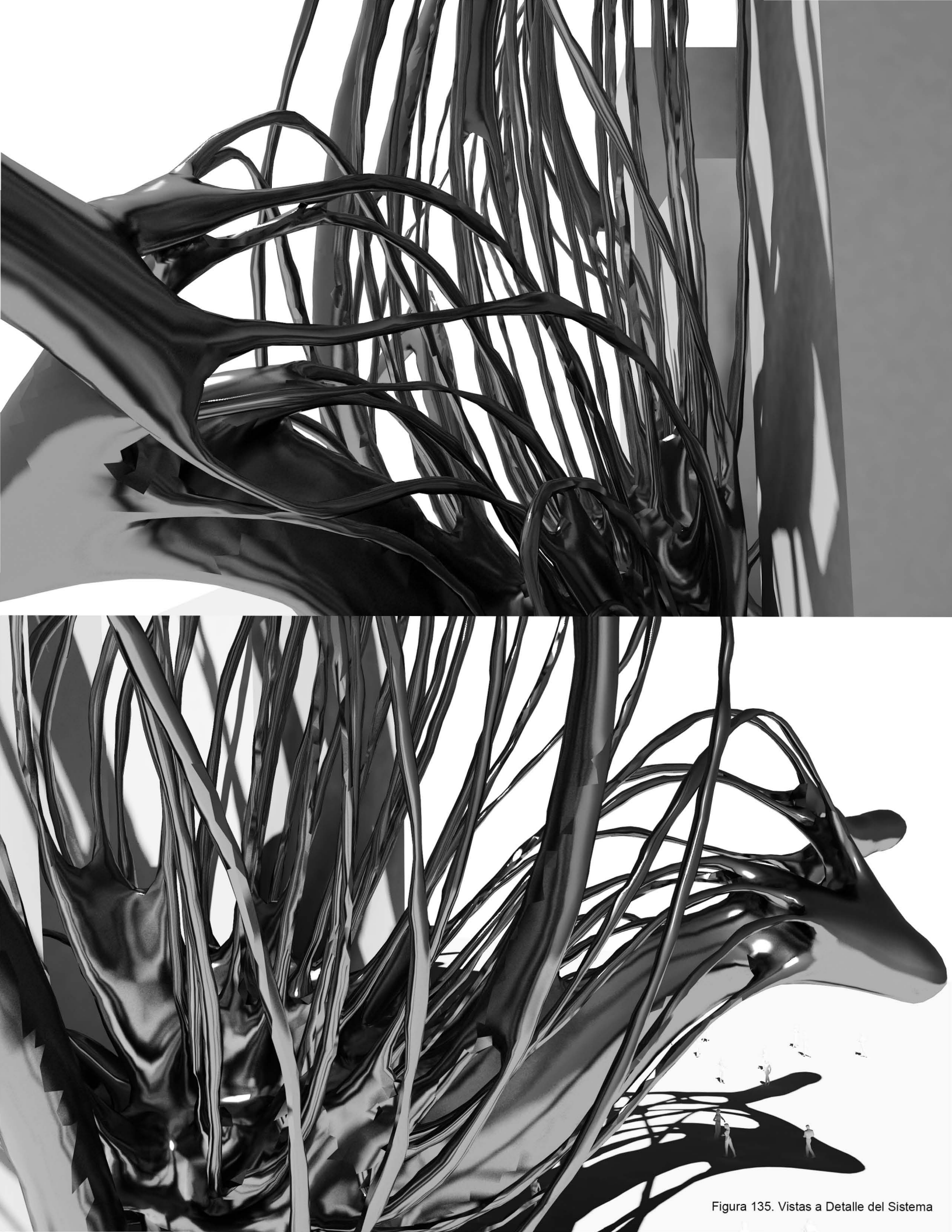


Figura 135. Vistas a Detalle del Sistema

5.7 - Conclusiones

Edsger Dijkstra es un ingeniero de software citado de manera continua en los primeros capítulos de esta tesis, el cual ha mencionado, que la ingeniería del software mantiene una relación “explícita” con las herramientas de trabajo actuales de los seres humanos:

Las computadoras son herramientas extremadamente flexibles y poderosas, que se pueden apreciar al aplicarlas en problemas específicos que cambian la forma de entender hasta las cosas más simples que nos rodean. Me atrevería a opinar que, mientras las consideramos como simples herramientas, se podría subestimar la importancia que tienen en el desarrollo de las actividades de los seres humanos en la actualidad. Su influencia como herramienta podría llegar a ser una gran ola de conocimiento en todas las culturas del mundo, mientras que yo espero que sean entendidas como entes de una profunda influencia en cuanto a capacidad de desafío intelectual que implica para la solución de cada problema humano.

Dijkstra 1970, 7

La distinción entre el uso de las computadoras como herramienta y medio generador, persiste en muchos de los discursos actuales relacionados con los modelos paramétricos. Este aspecto permanece explícitamente en la tesis doctoral de Roland Hudson *Strategies for Parametric Design in Architecture* del año 2010, dónde hace énfasis en la conceptualización compleja para producir “modelos paramétricos completos”, a partir de la identificación de problemas particulares en un sistema general para, posteriormente, definir enfoques variables de aplicación en las siguientes etapas de diseño, con el propósito de reducir el tiempo de generación y la complejidad de un logaritmo que describe al aspecto formal del modelo paramétrico.

Lo relevante es que la identificación y conceptualización de un problema complejo a variables o enfoques más sencillos y organizados, constituyen parámetros de inicio esenciales al desarrollar algoritmos basados en códigos, a partir de un lenguaje de programación textual, y que ayudan al arquitecto durante el proceso de diseño de cualquier objeto, mismo que depende de aspectos externos relacionados con la consecuencia formal.

A partir de un nivel de complejidad alto, en cuanto a conceptualización y enfoques de aplicación en este caso de este estudio, el desarrollo de la gráfica *DAG* y los componentes basados en el lenguaje de programación *Python*, produjeron un nivel medio de dificultad en el algoritmo de generación formal, que depende, al contrario, de aspectos complejos en cuanto a la incorporación de elementos generados por medio de una simulación aleatoria y relacionada con un entorno virtual; la ecuación de complejidad ciclomática de Thomas McCabe (páginas 31 – 31), nos ayudará a medir de manera cuantitativa este aspecto:

$$CC\ (G) = e - n + 2p + u - n$$

CC (Logaritmo Gráfico – Textual de Flujos basado en Agentes) $= e\ (\mathbf{51}) + i\ (\mathbf{55}) + u\ (\mathbf{50}) - n\ (\mathbf{52})$

Dónde:

- *i:* número de entradas.
- *u*: número de salidas.

CC (Logaritmo Gráfico – Textual de Flujos basado en Agentes) = 104

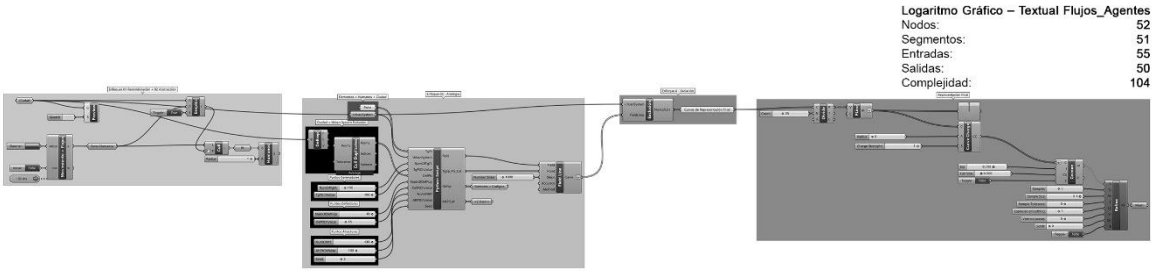


Figura 136. Gráfica *DAG* completa y final, la cual contempla a los tres componentes de *Python Script*, desarrollada a lo largo de este caso de estudio particular.

Tal y como lo muestra la ecuación anterior, la complejidad en el algoritmo *DAG* en este caso de estudio es media - baja (*104*); lo cual implica que, al incorporar componentes basados en código y definidos a partir de una sintaxis de programación textual, se reduce el nivel de complejidad en la gráfica, pero aumenta el nivel de complejidad en conceptualización y enfoques aplicados durante la etapa de diseño y representación final.

Por otra parte, este sistema formal de flujos, fue generado a partir de la incorporación del “modelado basado en agentes”, un tipo de enfoque en el diseño arquitectónico contemporáneo que implica conceptos de integración y materialización geométrica final de la relación de entidades. El principal aspecto, sin embargo, fue enlazar a cada una de estas entidades individuales en un sistema capaz de brindar una síntesis basada en la generación de la forma como consecuencia y en respuesta a variables externas cualitativas mediante variables de control cuantitativas. La investigación desarrollada a partir de este método puede permitir la constante retroalimentación, integración y construcción de modelos complejos en un proceso de diseño computacional particular, al permitir desarrollar componentes basados en códigos que solucionen problemas particulares a cada situación específica a la que se enfrentan los arquitectos en la práctica profesional.

Por lo anterior, el modelado basado en agentes y controlado a partir de código computacional, puede ser utilizado para realizar simulaciones de características externas que afecten al objeto arquitectónico, tales como la incidencia del movimiento de las personas en la estructura del edificio, análisis específicos de viento y cargas o radiación solar en las fachadas, de una manera muy específica; por otro lado, puede ser apto para optimizar y tomar decisiones en cada una las etapas de un proyecto que dependa de una serie de reglas dadas, estas incluyen a campos del conocimiento relacionados con la robótica, las finanzas, la economía, logística, juegos computacionales, biología y ciencias sociales. En el contexto arquitectónico y urbano, este método puede ser empleado, por ejemplo, al simular el tráfico vehicular de una ciudad, puede contribuir, además a optimizar la evacuación de un edificio durante un sismo al tener agentes dispersos y juntos en cada parte del edificio, generando espacios de seguridad y, en aplicaciones orientadas al diseño, la consideración del movimiento de las personas como medio de generación inicial y materialización de estructuras geométricas variables controladas por código, produciendo nuevos enfoques de aplicación en cualquier rama del diseño.

La generación de algoritmos basados en la escritura por medio de una sintaxis de programación computacional, permitirá, en cualquier caso que, el arquitecto desarrolle sus propias herramientas de trabajo, además de emplear cualquier método de conceptualización, independientemente del sistema de agentes; sin depender de aplicaciones externas y difíciles de incorporar en su trabajo diario; al producir cada parte del código, el arquitecto conocerá en profundidad todas las partes que conforman su

herramienta, propiciando la generación de nuevos enfoques aplicados al proceso de diseño arquitectónico.

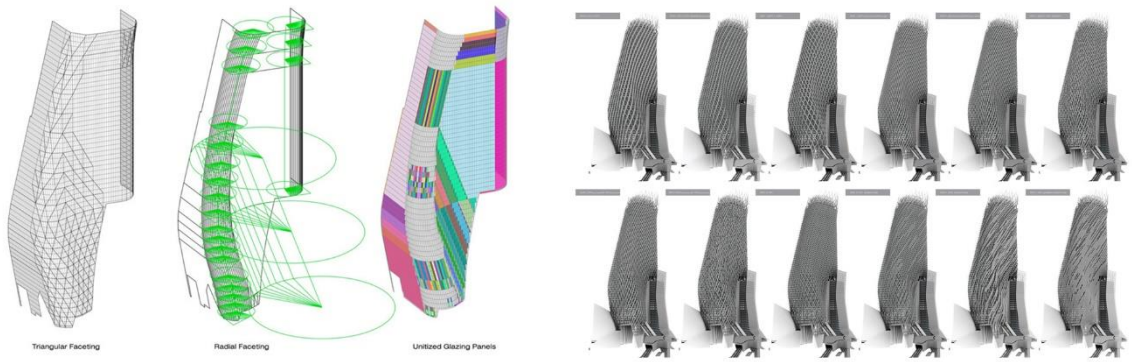


Figura 137. Estudio de generación y protección de fachadas en la *Torre Phare* de *Morphosis Architects*. Generado en el entorno de programación textual basado en el lenguaje Java iCode, desarrollado por el arquitecto Satoru Sugihara. París, Francia. 2014.

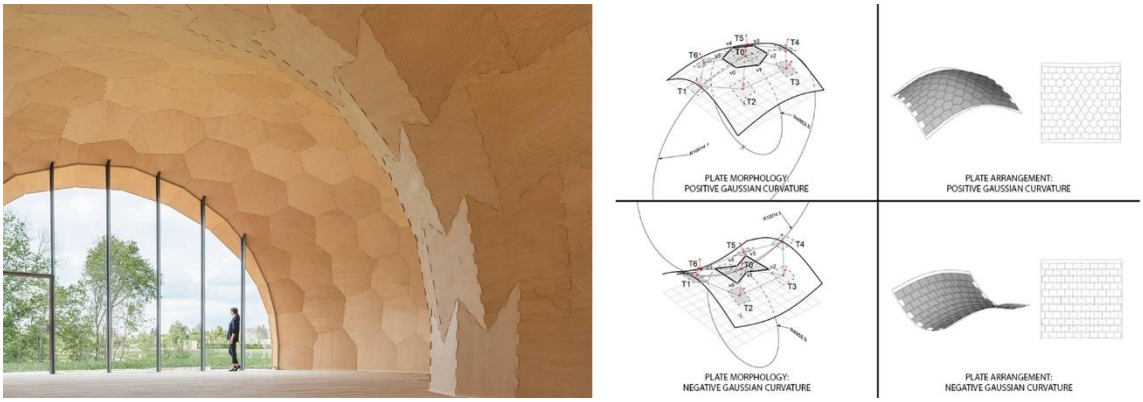


Figura 138. Landesgartenschau Exhibition Hall, Schwäbisch Gmünd, Alemania. Construido a partir del principio de modelado de agentes, como método principal de fabricación y panelización asistida computacional. ICD + ITKE. 2014.

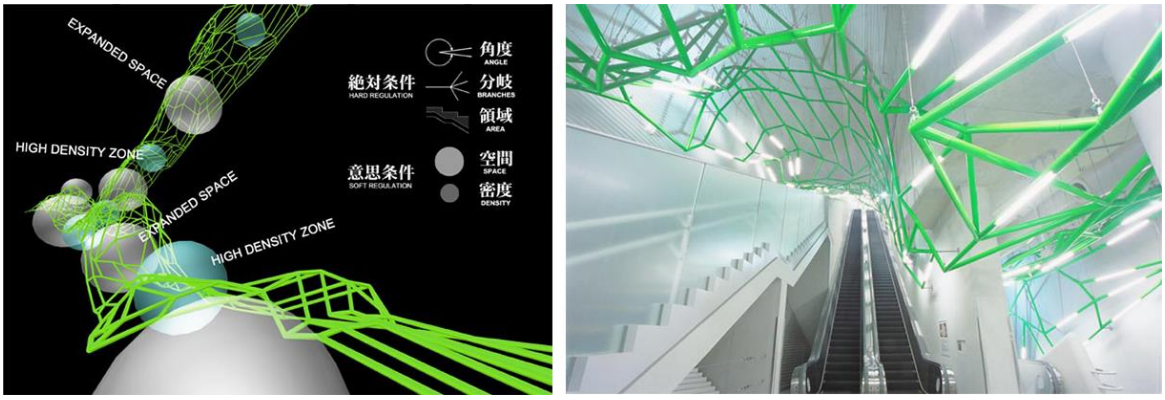


Figura 139. Estación de metro Iidabashi, Tokio, Japón. Las luminarias y la configuración interior espacial se generan a partir del principio de alta y baja densidad de humanos durante las horas de servicio. Arq. Makoto Sei Watanabe. 2000.

Conclusión General

Cuando Robert Woodbury describe al modelado paramétrico como una serie de herramientas para realizar adecuadamente las tareas del diseño, pero nunca siendo la tarea de diseño, menciona que “la tarea es más importante, la herramienta sólo debe ser adecuada a la misma” (2010, 9). Lo mismo persiste con cuando Benjamin Aranda y Chris Lash (2005) escriben en su libro *Tools*, que el trabajo de diseño comienza solamente una vez hecha la herramienta, ya sea desde un lápiz hasta el sistema de cómputo más potente y avanzado. Lo mismo sucede en las declaraciones, respecto a la enseñanza de la arquitectura, de John Frazer (2006), dónde afirma que “el diseño computacional es todavía visto por muchos, sólo como una herramienta de trabajo, alejada del negocio de las propiedades de diseño creativo”, que se debe principalmente a la falta interés de los profesores en las ciencias de la computación, como un tema ajeno a cualquier proceso de diseño. A menudo estás afirmaciones se mantienen cuando, por ejemplo, Patrik Schumacher (2009, 15) define al “parametricismo” en términos de resultados estilísticos y consecuencia de la “utilización herramientas de diseño paramétrico y *scripts*”. Y se vuelve a reafirmar cuando Mark Gage les dice a los arquitectos que “utilicen la computación, pero que dejen de investigar sobre eso” (2011, 1) y, más tarde, les propone contratar a ingenieros de software, porque esas herramientas son muy “nuevas” al aplicarlas en el diseño arquitectónico (2011, 133).

De una manera menos explícita, la separación de la tarea de *hacer* y *usar* una herramienta persiste en muchas definiciones contemporáneas del modelo paramétrico. Cuando diversos autores definen “paramétrico” como la totalidad de un objeto de diseño, o que los diseños pueden variar en relación a sus consecuencias formales o un estilo, implícitamente se centran únicamente en el resultado y no en el proceso. Al enfocarse en la tarea del *hacer*, muchas de las anteriores definiciones tienen características diferentes en relación al modelo o consecuencia formal del proceso en el modelado paramétrico, ya sea gráfico o textual; tales como la presencia de relaciones explícitas que unen parámetros y variables de comportamientos externos o internos con resultados geométricos, una de las características que más distinguen a este tipo de modelos generados logarítmicamente, de otras herramientas manuales o digitales de mera representación.

Existe algo diferente, sin precedentes y extraordinario en las computadoras cuando se comparan con herramientas de trabajo manuales y tradicionales. Esta diferencia radica en la incapacidad del ser humano para razonar inmediatamente acerca de todos los procesos computacionales, tales como el modelado paramétrico. Arquitectos como Neil Denari, Greg Lynn o Peter Eisenman utilizan este tipo de herramientas para describir únicamente procesos computacionales referidos a la consecuencia formal, sin embargo, ninguno de ellos tiene ningún tipo de educación formal en ciencias de la computación. De hecho, las discusiones acerca de la informática e ingeniería del software son ausentes en los debates alrededor del modelado paramétrico. A los estudiantes de arquitectura en general, no se les enseña nada acerca de la ingeniería del software, por lo que hay “poca conexión explícita”. En la literatura académica arquitectónica, existen muchos comentaristas prominentes que opinan del modelado paramétrico (varios de los cuales, no utilizan o han utilizado alguna vez en su práctica profesional modelos paramétricos, ni siquiera BIM), que parecen atrapados en la incógnita de si el modelado paramétrico constituye un nuevo movimiento en la arquitectura, o es solamente un desafío temporal de los arquitectos para que sean reconocidos al utilizar modelos paramétricos al resolver alguna problemática en particular.

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

La investigación desarrollada a lo largo de esa tesis ha revelado tres conexiones importantes entre el modelado paramétrico y la ingeniería del software; dichas conexiones enlazan desafíos mutuos, métodos de investigación compartidos y prácticas de diseño afines y comunes. Este tipo de desafíos compartidos son analogías en cuanto a la fabricación de herramientas digitales particulares que comienzan a carecer de importancia para la mayoría de los arquitectos; además, de ser eclipsados por el gran número de programas computacionales comerciales y genéricos de las grandes compañías de software. Sin embargo, en las grietas de los límites comerciales, queda la racionalización particular de cada ser humano capaz de generar modelos paramétricos gráficos o textuales, independientemente del lenguaje de programación, para dar oportunidad al cambio de paradigmas y enfoques en los procesos de diseño arquitectónico.

A menudo, existen fallas en los modelos paramétricos que obligan a cambiar la lógica y los enfoques de programación de un problema, con el propósito de representar a un proyecto completo. Ser capaz de cambiar, modificar o regresar en un modelo paramétrico es una característica distintiva mucho más que cualquier semejanza exterior, como en las herramientas CAD. En estos cambios y variaciones, los diseñadores son provocados para resolver problemas específicos, con herramientas específicas; pero también, a partir de estas modificaciones, del modelado paramétrico deriva su utilidad y la ingeniería del software adquiere la notabilidad correspondiente para la incorporación en la generación de los modelos paramétricos.

Esta tesis es un poco diferente, ya que narra las crónicas de lo que sucede con una gama de modelos paramétricos algorítmicos (Gráfico – *Grasshopper* + Textual – *Python*), en dos casos de estudio diferentes. Estos dos casos de estudio muestran que el diseño no es algo que un arquitecto hace con una herramienta paramétrica, sino más bien, algo que ocurre y se desarrolla de forma iterativa a lo largo de un proceso colectivo o particular. Lo anterior sugiere que, en una práctica, mediante la cual el usuario y el fabricante de la herramienta son indistinguibles y, por lo tanto, capaces de sintonizar una variedad de flexibilidades en un modelo para investigar y explotar algunos aspectos y enfoques de diseño, mientras que la reconstrucción o importación de las secciones del modelo logarítmico puede dar cabida para la generación de nuevos paradigmas de diseño. Todavía existe un vacío en nuestro conocimiento acerca de cómo ocurre el proceso anterior. Mientras que otras formas de representación arquitectónica tienen una rica historia de investigación crítica para aprovechar, la presente investigación indica que otras disciplinas, y en específico, la ingeniería del software puede ofrecer un fundamento similar para la práctica del modelado logarítmico paramétrico. Pero para ello, es necesario cambiar nuestro enfoque más allá de la fabricación o la producción de herramientas digitales hechas por el mismo arquitecto.

En muchos sentidos, la conclusión de esta tesis es simple: los ingenieros de software producen programas computacionales y, los arquitectos que diseñan a partir de modelos paramétricos logarítmicos, comparten algunos retos similares, que a menudo pueden ser interpretados como métodos de investigación similares, así como en prácticas de diseño. Pero esta aparente simplicidad puede ser difícil de discernir. Hace cincuenta años, cuando Timothy Johnson arrastró un lápiz electrónico por una pantalla y generó unos dibujos, parecía que estaba dibujando. Hoy en día muchas personas dirían que Johnson fabricó la herramienta, casi como si estuviera elaborando una regla T, que es de alguna manera, un precedente análogo al relacionar las conexiones de redes de funciones explícitas en un modelo paramétrico. Sin embargo, a diferencia de la regla T, o de cualquier otra herramienta de representación en el diseño, los modelos paramétricos logarítmicos fusionan los conceptos del *hacer* y *usar* hasta un punto de indistinción.

Esto representa retos aún desconocidos por los arquitectos y diseñadores; retos que han causado numerosos contratiempos en la entrega de proyectos de arquitectura. Este tipo de problemas se asemejan mucho a los problemas que enfrenta la ingeniería del software en la actualidad. Esta investigación sugiere que esta relación y semejanza ofrece un método para la experimentación y realización de posteriores documentos e investigaciones en el modelado paramétrico basado en algoritmos, tomando en consideración conceptos de las gráficas SWEBOK. Es cierto que existe una contradicción respecto a la idea de lo que los ingenieros y programadores pueden enseñar a los arquitectos en relación a la representación de un diseño o proyecto contemporáneo, pero puede ser difícil de comprender en algunos aspectos, sobre todo en aquellos profesionales que muestren poco interés en lo comentado. Lo relevante es que, en la actual práctica de la arquitectura, hay una relación más común con la ingeniería del software, que con los blocs de dibujo utilizados por las generaciones anteriores de arquitectos.

Bibliografía

Trabajos citados

A

Addadi, L., Raz, S., y Weiner, S., *"Taking advantage of disorder: amorphous calcium carbonate and its roles in biomineralization,"* Advanced Materials, Volúmen 15, No. 12, Páginas: 959–970. 2003.

Addington, D. and Schodek, D., *"Smart materials and new technologies: for the architecture and design professions",* Architectural Press, 2005.

ACM (Association for Computing Machinery). 2000*. "A Summary of the ACM Position on Software Engineering as a Licensed Engineering Profession".* Web report, consulta: 17.09.2015.

Abran, Alain, and James Moore. *Guide to the Software Engineering Body of Knowledge: Version 2004*. Los Alamitos: Institute of Electrical and Electronic Engineers Computer Society. USA. 2004.

Abrahams, Paul. *"Systematic Programming: An Introduction by Niklaus Wirth". Mathematics of Computation* (American Mathematical Society). USA. 1977.

Aish, Robert. 2005. *"From Intuition to Precision".* In 23rd eCAADe Conference Proceedings, 62–63. Lisbon: Technical University of Lisbon.

Aish, Robert, Benjamin Barnes, Mehdi Sheikholeslami, and Ben Doherty. 2012. *"Multi-modal Manipulation of a Geometric Model".* US Patent application 13/306,730.

Aish, Robert, and Robert Woodbury. 2005. *"Multi-level Interaction in Parametric Design".* In Smart Graphics: 5th International Symposium, edited by Andreas Butz, Brian Fisher, Antonio Krüger, and Patrick Olivier, 151–162. Frauenwörth Cloister: Springer.

Anker, P., *"Bauhaus at the zoo,"* Nature, Vol. 439, 2006, página 7079.

Antonelli, P*., "Design and the elastic mind",* Museum of Modern Art, 2008.

Antonelli, P*., "Mutant materials in contemporary design",* HNA Books, 1995.

Antonelli, P. L., "*Mathematical essays on growth and the emergence of form",* University of Alberta. Press Edmonton, Alta., Canada, 1985.

Appleby, Doris, and Julius VandeKoppel. *Programming Languages: Paradigm and Practice*. Segunda edición. Massachusetts, USA: McGraw-Hill. 1997.

Ashby, M., *"Materials selection in mechanical design"*, Butterworth-Heinemann, 2005.

Ashby, M. F., G. L. J. W. U. and Olive, R., "*The mechanical properties of natural materials. I. Material property charts,*" Proc. London. AR. Soc., Vol. 1, 1995, Páginas: 123–140.

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Atallah, M., Fox, S., and Lassandro, S., *"Algorithms and theory of computation handbook,"* Crs Press Boca Raton, FL, 1999.

B

Balmond, C., Smith, J., y Brensing, C., *"Fractile in Informal*", Prestel, Munich; New York, 2002.

Barrie, Donald, and Boyd Paulson. 1991. *Professional Construction Management: Including C.M., Design-construct, and General Contracting.* Tercera Edición. McGraw Hill.

Benyus, J., *"Biomimicry: Innovation inspired by nature",* Perennial, 2002.

Bishop, C., "*Neural networks for pattern recognition*", Oxford University Press, USA, 1995.

Brooks, Frederick. 1975. *The Mythical Man-month: Essays on Software Engineering.* Edición de Aniversario. Addison-Wesley. Boston.

Burry M., 2011*, scripting cultures: Architectural design and programming*. John Wiley & Sons.

C

Cross, Nigel. *The Automated Architect*. Pion Limited. 1997.

D

Dana, James. 1837. *"On the Drawing of Figures of Crystals."* The American Journal of Science and Arts 32: 30-50.

Danielson, K. and Noor, A., *"Three-dimensional finite element analysis in cylindrical coordinates for nonlinear solid mechanics problems,"* Finite Elements in Analysis & Design, Vol. 27, No. 3, 1997, pp. 225–249.

De Castro, L. N., "*Fundamentals of natural computing: basic concepts, algorithms, and applications*", Chapman & Hall/CRC, Boca Raton, 2006.

Dijkstra, Edsger. *"Go to Statement Considered Harmful".* Communications of the Association for Computing Machinery 11 Volúmen 3. USA. 147–148. 1968.

———. "*Notes on Structured Programming"*. Segunda Edición. Eindhoven: Technological University of Eindhoven. Nederland. 1970.
———. *"The Humble Programmer."* Communications of the Association for Computing Machinery 15, Volúmen 10. USA. 859-866. 1972.
———. *"Foreword."* In Communicating Sequential Processes, by Charles Hoare, iii. Upper Saddle River: Prentice Hall. USA. 1985.
———. *"The Tide, Not the Waves".* In Beyond Calculation: The Next Fifty Years of Computing, editado por Peter Denning y Robert Metcalfe, 59–64. New York: Springer. USA. 1997.

E

Edsger W. Dijkstra. *Cooperating sequential processes*. In: F. Genuys: *Programming Languages: NATO Advanced Study Institute*. Academic Press, 1968.

F

Fausett, L.*, "Fundamentals of neural networks: architectures, algorithms, and applications",* Prentice Hall Englewood Cliffs, NJ, 1994.

Flake G.W., 1998, "*The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation"*. MIT Press.

Forbes, N., *"Imitation of life: how biology is inspiring computing,"* Imitation of life: how biology is inspiring computing, Volúmen 1, página 1. 2004.

Frampton, K. and Futagawa, Y., *"Modern architecture",* Thames and Hudson, 1980.

Fudos, Ioannis. *"Constraint Solving for Computer Aided Design".* Tesis Doctoral, Purdue University. USA. 1995.

G

Gage, Mark. "Project Mayhem." Fulcrum, 8 June 2011.

———. *"Software Monocultures".* In Composites, Surfaces, and Software: High Performance Architecture, editado por Greg Lynn y Mark Gage, 107–120. New Haven: Yale School of Architecture.

Gibson, L., *"Biomechanics of cellular solids,"* Journal of biomechanics, Vol. 38, No. 3, 2005, p. 377–399.

Gibson, L. J., *The mechanical behavior of cancellous bone, Massachusetts Institute of Technology, School of Engineering, Department of Civil Engineering,* Cambridge, Mass., 1984.

Gordon, J. E., *"The new science of strong materials: or, why you don't fall through the floor, Penguin, Harmondsworth",* Middlesex, 2nd ed., 1976.

Gusev, A., *"Representative volume element size for elastic composites: a numerical study,"* Journal of the Mechanics and Physics of Solids, Volúmen. 45, No. 9, Páginas 1449–1459. 1997.

H

Hanna, S. and Mahdavi, S. H., *"Modularity and flexibility at the small scale: evolving continuous material variation with stereolethography,"* ACADIA Fabrication: Examining the Digital Practice of Architecture, Vol. 1, 2004, pp. 76–87.

Haykin, S., *"Neural networks: a comprehensive foundation "*, Prentice Hall PTR Upper Saddle River, NJ, USA, 1994.

Helm, P., Beg, M., Miller, M., and Winslow, R., “Measuring and mapping cardiac fiber and laminar architecture using diffusion tensor MR imaging,” ANNALS-NEW YORK ACADEMY OF SCIENCES, Vol. 1047, 2005, pp. 296.

Helmut, Balzert, *,,Lehrbuch der Software - Technik; Software-Management, Software-Qualitätssicherung*, Unternehmensmodellierung”. Páginas. 481–482.

Henderson-Sellers, Brian, y David Tegarden. *“The Theoretical Extension of Two Versions of Cyclomatic Complexity to Multiple Entry/exit Modules”*. Páginas 253–269. USA. 1994.

HOK. 2012*. “HOK History + Lore.”* Acceso 10.09.2015. http://www.hok.com/lore/.

I

ISO (International Organization for Standards). *Information Technology: Software product quality – Part 1: Quality mode. ISO/IEC 9126*. New York: American National Standards Institute. 2000.

J

Jencks, C., “*The language of post-modern architecture*”, Rizzoli Intl Pubns, 1984.

K

Keller, Sean. *“Fenland Tech: Architectural Science in Postwar Cambridge”.* 23 Abril 2013: 40-65. USA. 2006.

Kilian, A., *“Linking digital hanging chain models to fabrication,”* ACADIA Fabrication: Examining the Digital Practice of Architecture [Proceedings of the 23rd Annual Conference of the Association for Computer Aided Design in Architecture and the 2004 Conference of the AIA Technology. In Architectural Practice Knowledge Community / ISBN 0-9696665-2-7], Vol. 1, 2004, pp. 110 – 185.

Klein, L., *“A phenomenological interpretation of Biomimicry and its potential value for sustainable Design,”* A phenomenological interpretation of Biomimicry and its potential value for sustainable Design, Volúmen 1, página 01. 2009.

Kohonen, T., *“An introduction to neural computing,”* Neural networks, Vol. 1, No. 1, pp. 3– 16. 1988.

Kolarevic, B., 2003. *Digital morphogenesis*. In: B. Kolarevic, ed. *Architecture in the digital age.* Design and manufacturing. New York: Taylor & Francis.

Kowalski, Robert*. “The Early Years of Logic Programming.”* Communications of the Association for Computing Machinery 31. Volúmen 1: 38-43. 1998.

Kumar, S. and Bentley, P., *“On growth, form and computers”,* Elsevier Academic Press, Amsterdam; Boston, 2003.

Kwinter, S., 1993. *Soft Systems*. In: B. Boigon, ed. Culture Lab 1. New York: Princeton Architectural Press.

L

Leach, Neil. Xu Wei-Guo (curators), *(Im) material Processes: New Digital Techniques for Architecture"*, Architecture Biennial Beijing. China. 2008.

Lincke, Rudiger, and Welf Lowe. *Compendium of Software Quality Standards and Metrics. Version 1.0.* Publicado el 03 de abril de 2007. http://www.arisa.se/compendium/. Consulta 19.09.2015.

Livingston, Mike. 2002. "*Watergate: The name that branded more than a building."* Washington Business Journal, 09.09.2015.

M

MacLeamy, Patrick. 2010*. "Bim-Bam-Boom! How to Build Greener, High-performance Buildings."* HOK Renew. Consulta: 11.09.2015. http://hokrenew.com

McQuaid, M. y Beesley, P., *"Extreme textiles: designing for high performance",* Princeton Architectural Press, 2005.

Mattheck, C., *"Design in nature: learning from trees",* Springer, 1998.

Maturana, Humberto. *Der Baum der Erkenntnis. Die biologischen Wurzeln menschlichen Erkennens*. S. Fischer, Frankfurt am Main, 2009.

Makris, Dimitrios, Ioannis Havoutis, Georges Miaoulis, and Dimitri Plemenos. *"MultiCAD: MOGA A System for Conceptual Style Design of Buildings".* Editado por Dimitri Plemenos, 73-84. USA. 2006.

McCabe, Thomas. *"A Complexity Measure".* IEEE Transactions on Software Engineering: Páginas: 308-320. 1976.

McConnell, Steven. *"Software Estimation: Demystifying the Black Art".* Redmond: Microsoft Press. 2006.

McCulloch, W. and Pitts, W., "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biology, Vol. 5, No. 4, Páginas: 115–133. 1943.

Menges A., Ahlquist S., eds., 2011, *Computational Design Thinking*. John Wiley & Sons, London.

Meyer, Bertrand. Object-Oriented Software Construction. Segunda Edición. Upper Saddle River: Prentice Hall. Germany. 1997.

Miller, Robert. *"Response Time in Man-Computer Conversational Transactions".* In Proceedings of the AFIPS Fall Joint Computer Conference, Páginas: 267–277. Washington, DC: Thompson. 1998.

Mitchell, W. J. and McCullough, M.*, "Digital design media, Van Nostrand Reinhold*," New York, Segunda Edición. 1995.

Moretti, Luigi. 1957. *"Forma Come Structtura."* Spazio June-July. Republished in: Federico Bucci and Marco Mulazzani. 2000*. Luigi Moretti: Works and Writings.* New York: Princeton Architectural Press.

N

Naur, Peter and Randell, Brian. NATO Software Engineering Conference. Garmisch, Germany. 1968.

Neville, A., *"Biology of fibrous composites: development beyond the cell membrane",* Cambridge University Press, 1993.

Niklas, K. J., *"Plant biomechanics: an engineering approach to plant form and function"*, University of Chicago Press, Chicago, 1992.

O

Otto, F., Trostel, R., y Schleyer, F. K., "*Tensile structures; design, structure, and calculation of buildings of cables, nets, and membranes"*, The MIT Press, Cambridge, Mass., 1973.

Otto, F., Glaeser, L., and of Modern Art (New York N.Y.), M., *"The work of Frei Otto, Museum of Modern Art";* distributed by New York Graphic Society, Greenwich, New York, 1972.

Otto, F., Herzog, T., and Schneider, M., *"Der umgekehrte weg: Frei Otto zum 65. Geburtstag*, R. Müller, Klein, 1990.

Oxman, N., *"Rapid Craft: Material Experiments towards an Integrated Sensing Skin System." Proceedings of the 27th Annual Conference of the Association for Computer Aided Design in Architecture: Expanding Bodies: Art Cities Environment.* October 2007; Halifax, Nova Scotia, Vol. 1, Páginas: 182–191. 2007.

Oxman, N., *"Forbidden Form: Towards a Post Materialist paradigm in Architecture,"* Yale Architectural Press, Volúmen 1, Página: 43. 2010.

P

Paton, R., *"Towards a metaphorical biology,"* Biology and Philosophy, Vol. 7, No. 3, Páginas: 279–294. 1992.

Parnas, David. *"On the Criteria to Be Used in Decomposing Systems into Modules".* Communications of the Association for Computing Machinery 15, Volúmen 12, Páginas: 1053-1058. USA. 1972.

Pearce, P., *"Structure of Nature Is a Strategy for Design,"* Leonardo, Volúmen 14, No. 2, página. 175– 176. 1981.

Python Primer. "Python for Rhinoceros". Robert McNeel and Associates. USA. Review 3. Consulta 04 de octubre de 2015. http://www.rhino3d.com/download/IronPython/5.0/RhinoPython101.

R

Ray, T., “An evolutionary approach to synthetic biology,” Artificial Life, Vol. 1, No. 1/2, Páginas: 179–209. 1994.

Resnick, M., “Learning about life,” Artificial Life, Vol. 1, Páginas: 229–241. 1993.

Reynolds, C., *“An evolved, vision-based behavioral model of coordinated group motion,”* An evolved, vision-based behavioral model of coordinated group motion, Páginas: 245–262. 1993.

Rogers, C., *“Intelligent material systems the dawn of a new materials age”,* J. Intell. Mater. Syst. Struct, Vol. 4, No. 1, 1993, pp. 4–12.

S

Sánchez Vázquez, Adolfo. *La Filosofía de la Praxis*. (1967), Grijalbo, México 1980.

Sei Watanabe, Makoto. *ALGODEX. Theory + Work + Program: on the site.* Maruzen-publishing Japan. 2012.

Sennett, R., *“The craftsman”,* Yale University Press, 2008.

Sharp, John. *“A Brief Introduction to Data Flow.” In Dataflow Computing: Theory and Practice*, editado por John Sharp. Norwood: Ablex. Norway. 1992.

Shiffman, Daniel. *The nature of code: Simulating Natural Systems with Processing.* First Edition. 2012.

Shimada, K. and Gossard, D*., “Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis 1,”* Computer Aided Geometric Design, Vol. 15, No. 3, 1998, pp. 199–222.

Smith, A., *“Stereom microstructure of the echinoid test”,* Palaeontological Association, 1980.

Smith, C. S., *“A search for structure: selected essays on science, art, and history”,* MIT Press, Cambridge, Mass., 1981.

Smith, Rick. *“Technical Notes from Experiences and Studies in Using Parametric and BIM Architectural Software”.* Publicado el 4 de marzo de 2007. http://www.vbtllc.com/images/ VBTTechnicalNotes.pdf. Acceso 09 de septiembre de 2015. USA. 2007.

Sterling, Leon, and Ehud Shapiro. *The Art of Prolog: Advanced Programming Techniques.* Segunda edición. Massachusetts: MIT Press. USA. 1994.

Stiles, Robert. 2006. *“Aggregation Strategies”.* Masters dissertation, University of Bath.

Swinson, Peter. *“Logic Programming: A Computing Tool for the Architect of the Future”.* Computer-Aided Design 14 – Volúmen 2: 97-104. USA. 1982.

T

Takashi Oshima, Ken. *Arata Isozaki*. Phaidon Press. First Edition. 2009.

Tedeschi, Arturo. ADD – *Algorithms Aided Design. Parametric strategies using Grasshopper.* Le Penseur Publisher. Italy. 2014.

Teresko, John. *"Parametric Technology Corp.: Changing the way Products are designed".* Semana de la industria de 1993. USA. 1994.

Terzidis, K., 2003. "*Expressive form. A conceptual approach to computational design*." London: Spon.

Terzidis, K., 2006. *Algorithmic architecture.* 1st ed. Amsterdam, London: Architectural Press.

TIOBE Software. *"TIOBE Programming Community Index for May 2012".* Publicado en mayo 2012 en: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. USA. 2012.

Trummer, P., 2011. *Associative Design. From Type to Population*. In: A. Menges and S. Ahlquist, eds. Computational Design Thinking. London: John Wiley & Sons.

The Standish Group. *"The CHAOS Report 2012".* White-paper. Boston: The Standish Group. USA. 2012.

Thompson, D. W., *"On magnitude*," University Press, Cambridge [Versión en inglés], Segunda Edición., 1952.

Thompson, D.W., *"On growth and form*," The University Press, Cambridge, a new ed., 1942.

Thomson, N., J, "*The natural language for analytic computing*", Research Studies Press, Baldock, Hertfordshire, England; Philadelphia, PA, 2001.

V

Van Roy, Peter, and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming.* Massachusetts: MIT Press. USA. 2004.

Victor, Bret. *"Inventing on Principle."* Presentation at Turing Complete: Canadian University Software Engineering Conference, Montreal, 20 January 2012. Presentación digital de video. Acceso 10 de septiembre de 2015. http://vimeo.com/36579366. Canada. 2012.

Vincent, J., *"Structural biomaterials"*, Princeton University Press, 1990.

Vincent, J. F. V., *"Structural biomaterials",* Macmillan, London, 1982.

Vogel, S., "*Comparative biomechanics: life's physical world"*, Princeton University Press, Princeton, NJ, 2003.

W

Wallner, Johannes, and Helmut Pottmann. 2011. *"Geometric Computing for Freeform Architecture".* Journal of Mathematics in Industry. 1-11. USA. 2011.

Watson, Arthur, and Thomas McCabe*. "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric"*. Gaithersburg: National Institute of Standards and Technology. USA. 1996.

Weiner, S. and Wagner, H., *"The material bone: structure-mechanical function relations,"* Annual Review of Materials Science, Volúmen 28, No. 1, Páginas: 271–298. 1998.

Weisberg, David. 2008. *"The Engineering Design Revolution: The People, Companies and Computer Systems that Changed Forever the Practice of Engineering."* Acceso 10.09.2015. http://www.cadhistory.net

Weisstein, Eric. 2003. *CRC Concise Encyclopedia of Mathematics.* Second edition. Boca Raton: Chapman & Hall/CRC.

Wong, Yuk Kui, and John Sharp. *"A Specification and Design Methodology Based on Data Flow Principles."* In Dataflow computing: Theory and Practice, editado por John Sharp, Páginas: 37-79. Norwood: Ablex. USA. 1992.

Woodbury, Robert. *Elements of Parametric Design.* Taylor & Francis Ltd. USA. 2010.

Índice de ilustraciones

Figura 1 – Laboratorio Lincoln MIT, 1964 4

Figura 2 – Laboratorio Lincoln MIT, 1964 7

Figura 3 – Laboratorio Lincoln MIT, 1964 7

Figura 4 – Laboratorio Lincoln MIT, 1964 7

Figura 5 – Barry Boehm, 1981, página 40 11

Figura 6 – Kent Beck, 1999, página 26 11

Figura 7 – James Dana, 1837, páginas 41 – 43 14

Figura 8 – Robert Woodbury, 2010, página 11 16

Figura 9 – Federico Bucci y Marco Mulazzani, 2000, página 114 17

Figura 10 – Boyd Paulson, 1976, página 588 20

Figura 11 – Daniel Davis, 2013 20

Figura 12 – Daniel Davis basado en The Standish Group, 1994 – 2012 22

Figura 13 – Alan Rodríguez Carrillo, 2016 25

Figura 14 – Appleby y VandeKoppel, 1997 27

Figura 15 – Appleby y VandeKoppel, 1997 – 2016 27

Figura 16 – Alan Rodríguez Carrillo, 2016 32

Figura 17 – Alan Rodríguez Carrillo, 2016 32

Figura 18 – Alan Rodríguez Carrillo, 2002 – 2016 34

Figura 19 – Alan Rodríguez Carrillo, 2002 – 2016 34

Figura 20 – Alan Rodríguez Carrillo, 2002 – 2016 34

Figura 21 – Alan Rodríguez Carrillo, 2016 35

Figura 22 – Alan Rodríguez Carrillo, 2016 36

Figura 23 – Alan Rodríguez Carrillo, 2016 36

Figura 24 – Alan Rodríguez Carrillo, 2016 36

Figura 25 – Alan Rodríguez Carrillo, 2016 37

Figura 26 – Alan Rodríguez Carrillo, 2016 37

Figura 27 – Alan Rodríguez Carrillo, 2016 37

Figura 28 – Makoto Sei Watanabe, 2012, página 46 44

Figura 29 – Alan Rodríguez Carrillo, 2016 45

Figura 30 – Alan Rodríguez Carrillo, 2016 46

Figura 31 – Alan Rodríguez Carrillo, 2016 46

Figura 32 – Alan Rodríguez Carrillo, 2016 49

Figura 33 – Alan Rodríguez Carrillo, 2016 51

Figura 34 – Alan Rodríguez Carrillo, 2016 51

Figura 35 – Alan Rodríguez Carrillo, 2016 51

Figura 36 – Alan Rodríguez Carrillo, 2016 52

Figura 37 – Corrado Boehm y Giuseppe Jacopini, 1966 59

Figura 38 – Alan Rodríguez Carrillo, 2016 59

Figura 39 – Alan Rodríguez Carrillo, 2016 61

Figura 40 – Alan Rodríguez Carrillo, 2016 62

Figura 41 – *Python for Rhinoceros 5*, 2011, página 2 66

Figura 42 – Alan Rodríguez Carrillo, 2016 68

Figura 43 – *Python for Rhinoceros 5*, 2011, página 11 71

Figura 44 – Alan Rodríguez Carrillo, 2016 72

Figura 45 – Alan Rodríguez Carrillo, 2016 73

Figura 46 – Alan Rodríguez Carrillo, 2016 75

Figura 47 – Alan Rodríguez Carrillo, 2016 75

Figura 48 – Alan Rodríguez Carrillo, 2016 77

Figura 49 – Alan Rodríguez Carrillo, 2016 81

Figura 50 – Fábrica de Ford, Modelo A, 1928 85

Figura 51 – *La Ville Radieuse*, Le Corbusier 1992 85

Figura 52 – *Aristolochia Clematitis*, Karl Blossfeldt 1982 87

Figura 53 – *Orthocarpus Luteus*, Robert Kesseler, 2006 90

Figura 54 – *Juncus Effesus*, Roger Neville Hughes 1993 92

Figura 55 – Fémur humano, Klaus Bach 1995 94

Figura 56 – Escaneo de fémur humano, Thompson D. W. 1992 95

Figura 57 – Fibras de madera, Thompson D. W. 1992 95

Figura 58 – Análisis CFD, Ludwig Boltzmann, 2009 99

Figura 59 – Frei Otto, 1990 101

Figura 60 – Frei Otto, 1990 102

Figura 61 – Madera en anisotropía, Thompson D. W. 1992 105

Figura 62 – *Mountain Range*, Otto y Herzog, 1990 107

Figura 63 – Tensión en madera, Thompson D. W. 1992 110

Figura 64 – Espectro *Juniperus*, Jane Sper, 2011 112

Figura 65 – Clair y Thibaut, 2001 – 2005 112

Figura 66 – Roger Neville Hughes, 1993 114

Figura 67 – Weiner, Addadi y Wagner, 1998 118

Figura 68 – Gibson y Ashby, 1997 119

Figura 69 – Michael Ashby, 1998 125

Figura 70 – Smith y Carnevali, 1991 127

Figura 71 – Neri Oxman, 2014 133

Figura 72 – Alan Rodríguez Carrillo, 2016 135

Figura 73 – Neri Oxman, 2011 135

Figura 74 – *MRI DT*, Helm, 2005 136

Figura 75 – Alan Rodríguez Carrillo, 2016 142

Figura 76 – Alan Rodríguez Carrillo, 2016 143

Figura 77 – Alan Rodríguez Carrillo, 2016 144

Figura 78 – Alan Rodríguez Carrillo, 2016 145

Figura 79 – *Acropora Cervicornis*, Bonaire, 2007 146

Figura 80 – Alan Rodríguez Carrillo, 2016 148

Figura 81 – B. Sapoval, 2014 149

Figura 82 – Dave Fleetham, 2010 150

Figura 83 – Gregory Bard, 2011 151

Figura 84 – Alan Rodríguez Carrillo, 2016 153

Figura 85 – Alan Rodríguez Carrillo, 2016 154

Figura 86 – Alan Rodríguez Carrillo, 2016 156

Figura 87 – Dr. Diego Liman, 2012 158

Figura 88 – Alan Rodríguez Carrillo, 2016 172

Figura 89 – Alan Rodríguez Carrillo, 2016 173

Figura 90 – Alan Rodríguez Carrillo, 2016 190

Figura 91 – Alan Rodríguez Carrillo, 2016 191

Figura 92 – Alan Rodríguez Carrillo, 2016 200

Figura 93 – Alan Rodríguez Carrillo, 2016 201

Figura 94 – Dr. Ray Berkelmans, 2009 202

Figura 95 – Alan Rodríguez Carrillo, 2016 203

Figura 96 – Alan Rodríguez Carrillo, 2016 209

Figura 97 – Alan Rodríguez Carrillo, 2016 210

Figura 98 – Alan Rodríguez Carrillo, 2016 211

Figura 99 – Alan Rodríguez Carrillo, 2016 211

Figura 100 – Alan Rodríguez Carrillo, 2016 212

Figura 101 – Alan Rodríguez Carrillo, 2016 212

Figura 102 – Alan Rodríguez Carrillo, 2016 212

Figura 103 – Alan Rodríguez Carrillo, 2016 212

Figura 104 – Alan Rodríguez Carrillo, 2016 213

Figura 105 – Alan Rodríguez Carrillo, 2016 214

Figura 106 – Alan Rodríguez Carrillo, 2016 215

Figura 107 – Frei Otto y Bodo Rasch, 1960 218

Figura 108 – Joerg Schlaich y Bergermann, 2004 218

Figura 109 – Arata Isozaki, 2011 218

Figura 110 – Arata Isozaki, 1970 221

Figura 111 – Alan Rodríguez Carrillo, 2016 224

Figura 112 – Peter Cook, 2003 228

Figura 113 – Kenchiku Bunka, 1972 234

Figura 114 – Alan Rodríguez Carrillo, 2016 241

Figura 115 – Alan Rodríguez Carrillo, 2016 242

Figura 116 – Alan Rodríguez Carrillo, 2016 250

Figura 117 – Alan Rodríguez Carrillo, 2016 252

Figura 118 – Alan Rodríguez Carrillo, 2016 253

Figura 119 – Alan Rodríguez Carrillo y Philip Siedler, 2016 254

Figura 120 – Alan Rodríguez Carrillo y Philip Siedler, 2016 255

Figura 121 – Alan Rodríguez Carrillo y Philip Siedler, 2016 256

Figura 122 – Alan Rodríguez Carrillo, 2016 256

Figura 123 – Alan Rodríguez Carrillo, 2016 257

Figura 124 – Alan Rodríguez Carrillo, 2016 269

Figura 125 – Alan Rodríguez Carrillo, 2016 272

Figura 126 – Alan Rodríguez Carrillo, 2016 273

Figura 127 – Alan Rodríguez Carrillo, 2016 274

Figura 128 – Alan Rodríguez Carrillo, 2016 275

Figura 129 – Alan Rodríguez Carrillo, 2016 276

Figura 130 – Alan Rodríguez Carrillo, 2016 278

Figura 131 – Alan Rodríguez Carrillo, 2016 279

Figura 132 – Alan Rodríguez Carrillo, 2016 280

Figura 133 – Alan Rodríguez Carrillo, 2016 281

Figura 134 – Alan Rodríguez Carrillo, 2016 282

Figura 135 – Alan Rodríguez Carrillo, 2016 283

Figura 136 – Alan Rodríguez Carrillo, 2016 285

Figura 137 – Satoru Sugihara, 2014 286

Figura 138 – Achim Menges y Jan Knippers, 2014 286

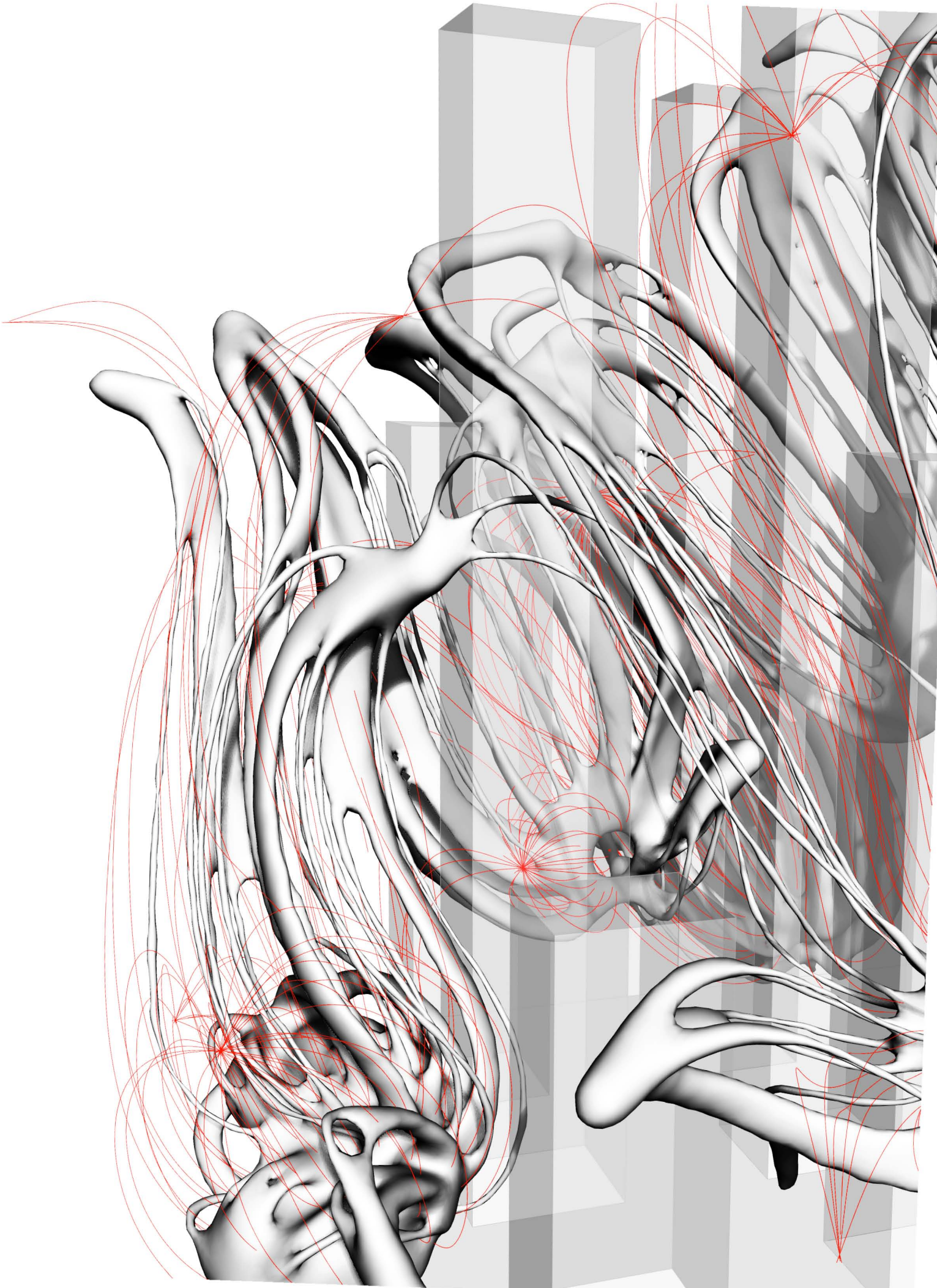



Figura 139 – Makoto Sei Watanabe, 2000 286