Universidad Nacional Autónoma de México

Facultad de Ciencias

Generación Procedural de Ambientes Urbanos Virtuales con gramáticas y un análisis de su calidad

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
Licenciado en Ciencias de la Computación

PRESENTA:
JULIO EZEQUIEL GONZÁLEZ ROJAS

DIRECTORA DE TESIS:
MARÍA CONCEPCIÓN ANA LUISA SOLÍS GONZÁLEZ COSÍO

DIRECTORA DE TESIS:
MARÍA CONCEPCIÓN ANA LUISA SOLÍS GONZÁLEZ COSÍO

Ciudad Universitaria, CD. MX
2022

Agradecimientos

A mi asesora Ana Luisa Solís González Cosío, por su enorme paciencia, su continuo apoyo y las oportunidades que me ha brindado al trabajar con ella.

A mis sinodales: Dr. Jesús Savage Carmona, Dra. Verónica Esther Arriola Ríos, Dr. Homero Vladimir Ríos Figueroa y MCIC Daniel Ruelas Milanés, por el tiempo que invirtieron en leer mi trabajo, y por la ayuda prestada en sus comentarios.

A las y los profesores de Facultad de Ciencias, por formar mi mente, templar mi carácter, y mostrarme la alegría de entender.

Al profesor César Hernández, por proporcionar valiosos recursos de escritura al iniciar este trabajo.

A mi familia, por su apoyo e incansable diligencia.

A esos científicos, artistas, escritores y mentes creativas, por darnos ejemplos a seguir, e incontables formas de ver el mundo.

Índice

Introducción

En su afán por entretenerse, el ser humano ha buscado desarrollar nuevas tecnologías que le permitan llevar su visión a la realidad: Videojuegos, películas, animaciones. Hoy en día, estas experiencias forman parte importante de la cultura. Son un reflejo de cómo se percibe el mundo en ese momento.

Con un par de clicks y unos ojos atentos a una pantalla, podemos adentrarnos en la capilla sixtina, ver un concierto virtual desde la comodidad de nuestro sillón o saltar en paracaídas sin mover un dedo; o incluso más allá, a parajes inexistentes que cobran vida ante nuestros ojos, todo gracias a una industria de entretenimiento y simulación siempre creciente, siempre cambiante, ansiosa de dar vida a nuevas experiencias.

Pero todo eso tiene un precio, y para que un desarrollador independiente pueda competir en la industria, en ocasiones tiene que hacer uso de tecnologías que reduzcan costos y tiempos, sin perder control ni calidad en el contenido creado.

Ahí es donde entra la generación procedural de contenido (PCG por sus siglas en inglés), la cual consiste en usar un algoritmo para crear contenido que sería generalmente creado por un humano: Desde modelos 3D y cortas historias hasta paisajes enteros, la generación procedural pone una gran cantidad de contenido al alcance del desarrollador independiente sin acarrear costes millonarios.

Aunque no todo son ventajas. Basta dar un paso en falso y el uso de PCG puede ser incluso contraproducente, tal es el caso de Spore o No Man's Sky, videojuegos que vieron sus costos de desarrollo incrementados o tuvieron defectos en el contenido generado. Deben hacerse pruebas exhaustivas para garantizar la calidad del contenido.

El ambiente generado en ocasiones es una malla de modelo 3D de una ciudad, y

con este fin en mente, y el de explorar el proceso de prueba del algoritmo, es que este trabajo fue creado.

El capítulo 1 define la generación procedural de contenido, explora los distintos enfoques que existen de ella y expone los problemas de su uso. Al final, se plantea la problemática principal y se define la hipótesis con que se trabajará.

Al inicio del capítulo 2 se describe el contenido a generar, explorando las definiciones teóricas de ciudad, contenido virtual y evaluación de contenido PCG aplicado a ciudades. Más tarde se exploran las alternativas existentes a la generación PCG de ciudades y los aportes que puede realizar este trabajo.

El capítulo 3 propondrá un diseño para el algoritmo PCG, tanto en la estructura del programa como en las técnicas utilizadas. También define la entrada que toma y los parámetros que permiten controlarlo.

La implementación del algoritmo es explicada a detalle en el capítulo 4, desglosando las técnicas propuestas anteriormente y corrigiendo cualquier aspecto que en la práctica no sea tan factible.

La prueba del algoritmo se ve en el capítulo 5, que comparará los resultados de evaluar los ambientes usando funciones de evaluación con los de la opinión directa del usuario, encontrando posibles errores.

Al final, se recopilan resultados y se determina si los ambientes generados tuvieron la calidad suficiente, se discuten posibles explicaciones y se dan medios para solicitar el código fuente del proyecto.

Capítulo 1

Generación Procedural de Contenido (PCG)

1.1 Visión General

1.1.1 Generación Procedural de Contenido

La Generación Procedural de Contenido (abreviado PCG - Procedural Content Generation en Inglés), es el uso de un algoritmo formal para generar contenido de forma automatizada que sería comúnmente generado por un humano.

Suele tener la forma de una función parametrizada, cuya entrada (Puntos en el espacio, números, etc...) puede ser modificada por el usuario e introducida a un algoritmo que proporciona como salida el contenido requerido (Modelos 3D, vectores, etc...).

En años recientes se han desarrollado distintos métodos para producir una gran cantidad de contenido, siendo una alternativa a la que poco a poco se va recurriendo más, en formato visual, escrito e incluso auditivo. En la siguiente sección se dará un panorama general de la PCG y sus diversas aplicaciones y dificultades.

1.1.2 Historia y Aplicaciones

La generación procedural surgió en al año 1980 con el lanzamiento del juego Rogue[1], que utilizaba dicha técnica para generar calabozos que el jugador era capaz de ex-

[1]Rogue, 1980. Juego de aventuras que usa PCG.

plorar.

En primera instancia, la PCG fue usada para lidiar con la limitada capacidad de memoria de los dispositivos de esa época. Un ejemplo de ésto se presentó en 1984 con Elite[2], que era capaz de generar 8 galaxias con 256 planetas distintos y sus respectivas características.

Al aumentar la capacidad de memoria y procesamiento para dispositivos multimedia, los desarrolladores dejaron de lado la PCG para empezar a crear ellos mismos el contenido. Ejemplos de ésto, son juegos como Halo[3] y películas como Zootopia[4], conocidos ejemplares de la industria del entretenimiento. Asímismo, en el área de producción de contenido multimedia, surgieron diversas herramientas para su creación, como Maya[5], Photoshop[6] y Audacity[7], software ampliamente usado y que se ha convertido en el estándar de la industria.

Sin embargo, en tiempos recientes, la PCG es cada vez más utilizada debido a su flexibilidad y a su inmensa capacidad para ahorrar trabajo a gran escala. Nuevas herramientas son creadas para reemplazar roles que no necesitan el nivel de destreza de un ser humano para otorgar resultados aceptables. Ejemplos de ésto son Minecraft[8], Darkest Dungeon[9] y Pokémon Go[10] en videojuegos, mientras que en el desarrollo de herramientas se vieron avances como SpeedTree[11], City Engine[12] y Mixamo [13].

Debido a que muchos de los ambientes tienen patrones en sus componentes o pueden ser descritos con estructuras matemáticas, es relativamente sencillo transcribir su naturaleza a un algoritmo.

---

[2]Elite, 1984. Juego de exploración y comercio que usa generación procedural para no almacenar todo el entorno en disco.

[3]Juego de disparos en primera persona con temática de ciencia ficción, 2001

[4]2016, película sobre una aspirante a policía que es un conejo, en la metrópolis de Zootopia.

[5]Herramienta de modelado, esculpido y animación 3D.

[6]Herramienta de manipulación y creación de imágenes.

[7]Herramienta de manipulación y creación de audio.

[8]Juego de exploración y mundo abierto que genera terrenos y estructuras de forma procedural

[9]Juego de exploración de mazmorras y combate que genera ambientes de forma procedural

[10]Juego de realidad aumentada que genera proceduralmente puntos geográficos importantes y entidades dentro del mundo

[11]Programa para generación procedural de vegetación en ambientes virtuales

[12]Programa de generación de ambientes urbanos de forma procedural

[13]Programa de creación de animaciones y rigging de forma automática.

Figura 1.1: Ejemplos de contenido hecho por humanos, de izquierda a derecha: Halo 1, Zootopia y una fotomanipulación usando Photoshop

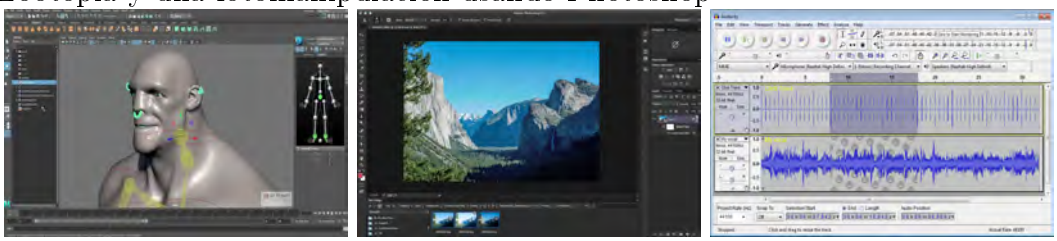


Figura 1.2: Herramientas de creación de contenido. De izquierda a derecha: Maya, Photoshop y Audacity

1.2 Técnicas de generación

Se discutirán brevemente algunas de las técnicas de generación procedural más usadas.

1.2.1 Fractales

Los fractales son estructuras geométricas con un alto nivel de replicación [1]. Un ejemplo de ellos es el copo de Koch, que puede verse en la figura 1.6.

Se puede generar: Vegetación (Fig. 1.5); terreno [18]; copos [2]; símbolos, etc... Objetos cuyos bloques de construcción sean similares al objeto son ideales para modelarse con fractales.

Los bloques que conforman los fractales no siempre son evidentes a primera vista, por ende, desde su concepción, los fractales han sido construidos por algoritmos. Gracias a que están limitados a estructuras autocontenidas, son sencillamente implementados con algoritmos recursivos.

Figura 1.3: Ejemplos de contenido generado proceduralmente, de izquierda a derecha: Minecraft, The Darkest Dungeon y Pokémon Go

Figura 1.4: Herramientas de creación de contenido. De izquierda a derecha: SpeedTree, City Engine y Mixamo Auto Rigging

Esta misma 'ventaja' es también su factor limitante, pues no todos los objetos se prestan a ser modelados con estas estructuras, o pueden necesitar percibir su ambiente para decidir cómo 'crecer'.

1.2.2 Sistemas L

Los Sistemas de Lindenmayer, o Sistemas L, son sistemas de re-escritura de cadenas (Fig. 1.7) que operan sobre cadenas de caracteres. Son muy similares a una gramática de Chomsky, sólo que, a diferencia de estas, la escritura de cada carácter se realiza de forma paralela [3].

Los sistemas L fueron concebidos como una teoría matemática para simular el crecimiento de plantas [3], y eran incapaces de simular cualquier tipo de comunicación entre la planta y el ambiente [4].

El primer paso para solucionar ésto fueron los sistemas L estocásticos y sistemas L parametrizados, que permitían elegir una producción con cierta probabilidad o el cambio en la derivación base al ambiente de manera cuantitativa o cualitativa.

Figura 1.5: Ejemplos de contenido generado con fractales

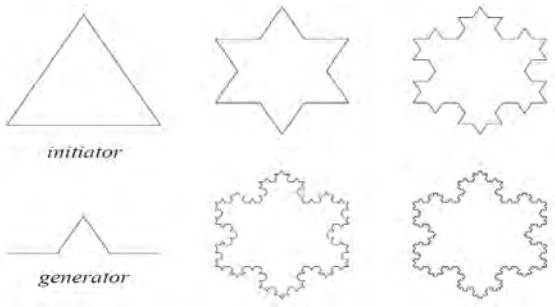


Figura 1.6: Ejemplo de un algoritmo de fractal: El Copo de Koch.

Luego llegaron los sistema (k,l), que permitían definir un sistema L dependiente del contexto con una contexto izquierdo de longitud k y un contexto derecho de longitud l, permitiendo una percepción casi absoluta del ambiente. Luego llegaron los sistemas L abiertos, que utilizan un símbolo que permite una comunicación bilateral con el ambiente [4].

En general, se definen como una tripleta ordenada <V,w,P>, en la que V es el alfabeto del sistema (Los caracteres que lo conforman), w $\in P^+$ es el axioma, y P un conjunto de reglas [3].

Donde las reglas, para los sistemas L cerrados, se definen de la siguiente forma:

id : lc <pred> rc: cond -> suc : prob

Donde id es el identificador de la producción, lc, pred y rc son el contexto

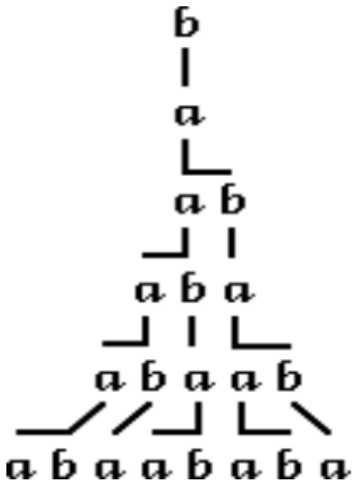


Figura 1.7: Ejemplos de derivación del primer sistema L, con las reglas b → a, a → ab.

izquierdo, el predecesor y el contexto derecho respectivamente, cond es la condición para que el sucesor sea escogido, suc es el sucesor, y prob es la probabilidad con la que ese elige esa regla [4], todos ellos están contenidos en V.

Cada extensión agrega nuevos elementos a esta definición que permiten ampliar su funcionalidad. Una clasificación de los principales puede apreciarse en la figura 1.8.

La capacidad de los sistemas L de comunicarse con el ambiente los hace idóneos para simular crecimiento, y se pueden aplicar en plantas, árboles, y urbanismo.

Se controla el contenido generado mediante el nivel de profundidad del sistema y las condiciones con que las reglas se derivan.

1.2.3 Funciones de Ruido, Interpolación y Turbulencia

Las funciones de ruido son funciones diseñadas para generar números pseudo aleatorios. Suelen ser funciones de estado, de tal forma que, al evaluarse, un nuevo número sea devuelto calculado a partir de una operación realizada al número anterior, resultando en datos aparentemente aleatorios, pero que se pueden recuperar de poseer el número inicial.

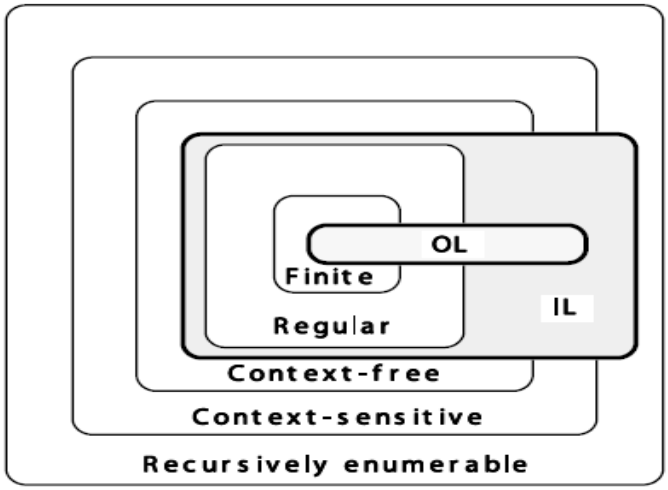


Figura 1.8: Ubicación de los sistemas L independientes (OL) y dependientes del contexto (IL) en la jerarquía de Chomsky.

Una de las primeras funciones de este tipo es la función de ruido de Perlin (Fig. 1.10), que genera números aleatorios en una matriz que luego son interpolados. Ésto nos permite usar dicha función para generar texturas con un alto nivel de detalle [5], o terreno.

Existen implementaciones en generación de ambientes urbanos virtuales usando esta técnica que muestran resultados aceptables [7].

La siguiente evolución de las funciones de ruido son las funciones de interpolación (Fig. 1.11), éstas 'unen' los números generados, colocando valores en medio usando curvas. Aunque puede ser un proceso tardado, son muy útiles cuando se dispone de suficiente tiempo para realizar la interpolación.

Sin embargo, las funciones de interpolación en ocasiones no logran igualar la estructura caótica que encontramos en algunos objetos de la naturaleza.

Para solucionar ésto, se usan las llamadas funciones de turbulencia, que consiste en introducir datos semi-aleatorios en medio de los valores interpolados, generando así una ilusión de caos que puede ser aplicada a pequeños detalles, como la simulación de olas oceánicas [10].

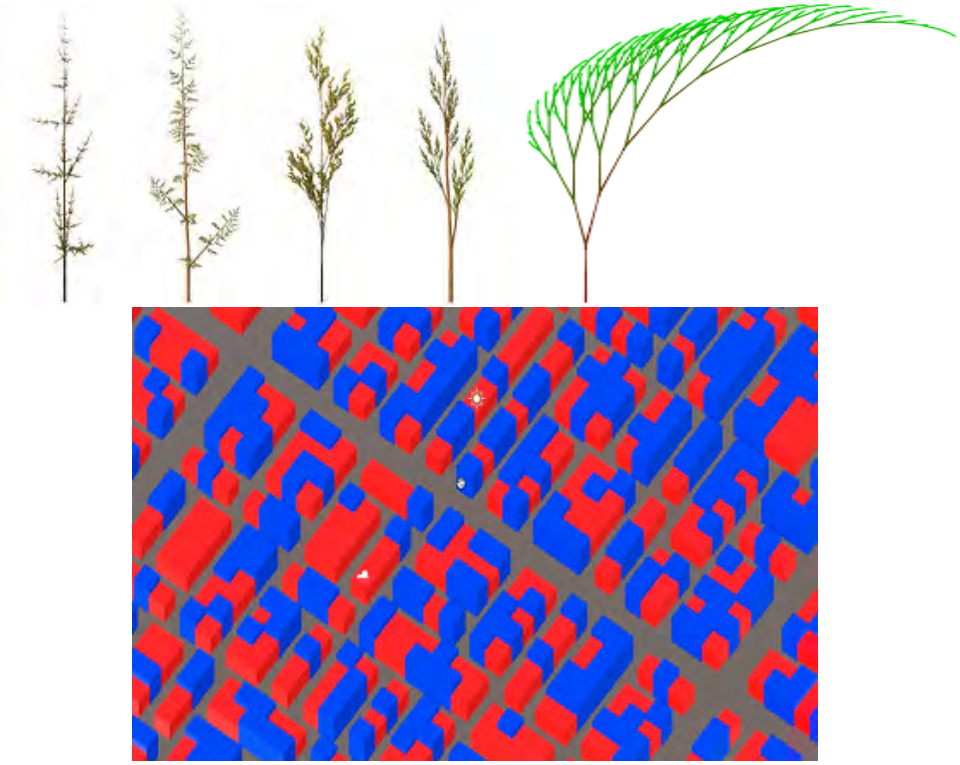


Figura 1.9: Sistemas L aplicados a la creación de diverso contenido

1.2.4 Semillas Aleatorias

Los algoritmos PCG deben ser diseñados como funciones inyectivas, para que el usuario pueda replicar los resultados dada una entrada. Ya que los algoritmos PCG hacen uso extensivo de generadores de números aleatorios, se debe diseñar de tal forma que genere siempre el mismo flujo de números 'aleatorios' a partir de las mismas condiciones iniciales.

El parámetro que da unicidad a esas condiciones iniciales se denomina semilla. Si se desean resultados replicables, se debe usar esta semilla y recordarla para reproducir la misma secuencia aleatoria más tarde [11].

Se puede decir que dichos números son sólo aleatorios si el observador desconoce dichas condiciones, pues una vez conocida la semilla, se puede predecir la salida, por lo que es una forma muy útil de garantizar la inyectividad del algoritmo de PCG.

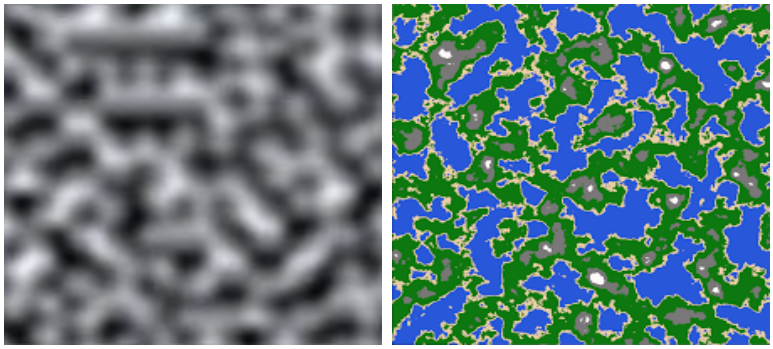


Figura 1.10: Ruido de Perlin y su aplicación en generación de terreno

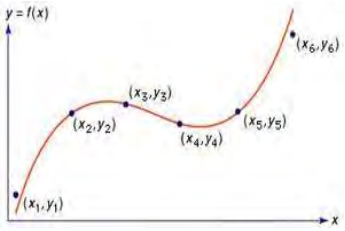


Figura 1.11: Ejemplo de función de interpolación

1.3 Enfoques y distinciones de la Generación Procedural

Los distintos enfoques de la PCG[12] la dividen según cómo se generará el contenido, bajo qué condiciones se genera, qué tipos de parámetros recibe o bien el tipo de estrategias utilizadas para elegir el contenido que mejor se ajusta a las necesidades del proyecto.

1.3.1 *Online* versus *Offline*

Se refiere a si el contenido es generado en tiempo real (*online*) o durante una fase de producción (*offline*).

Un ejemplo de PCG *online* sería un juego que genera el interior de un edificio en cuanto el jugador entra a él. Uno de PCG *offline* sería la generación de un paisaje para una película en producción.

Al generar contenido *offline*, se dispone de más tiempo para generar y ajustar el contenido, por lo que suele ser de mayor calidad. Una aproximación muy utilizada es generar un contenido que luego es refinado por un humano, para luego ser colocado en el producto final. Aunque se desvía un poco del propósito de utilizar PCG en un proyecto, es una buena opción para garantizar la calidad del contenido.

1.3.2 Contenido necesario contra contenido opcional

Podemos separar el contenido generado en si es necesario u opcional. El necesario es contenido esencial para la credibilidad del contenido a generar, mientras que el opcional aumenta la credibilidad sin perderse por completo si es omitido.

Un ejemplo de ésto es: En la generación de una ciudad necesitan colocarse caminos (esencial) para vehículos y peatones. Por otro lado, edificar rascacielos aumentará la inmersión del ambiente, pero su ausencia es tolerable.

Por otro lado, un error en el contenido necesario será más fácilmente percibido que un error en el contenido opcional, por lo que esta característica puede afectar profundamente el nivel de inmersión que el usuario percibe al presentarle el contenido.

1.3.3 Semillas aleatorias contra vectores de parámetros

Se debe planear también hasta qué punto el algoritmo es parametrizable, ya que nos otorga control sobre la salida del algoritmo al expandir su contenido a partir de estos parámetros.

Pueden ir desde un sólo parámetro, como una semilla aleatoria, hasta un vector de parámetros que describan a detalle las características del contenido deseado.

A más parámetros tenga el algoritmo PCG, más control se tendrá sobre el contenido, pero también será más complicado de usar, ya que buscar los parámetros óptimos es también tardado. Por el contrario, una baja cantidad de parámetros puede hacernos perder control sobre el algoritmo.

1.3.4 Generación estocástica contra generación determinística

Esta distinción aborda un problema de diseño, ya que algunos algoritmos son concevidos para generar aleatoriamente su contenido, sin oportunidad de parametrizarlo ni recuperarlo.

Una generación estocástica genera todo al azar, lo que nos permite simplificar u omitir la parametrización, mientras que una generación determinística se puede ver más bien como una compresión de datos.

Si no se necesita recuperar los datos, será preferible la generación estocástica, mientras que si se desea trabajar sobre un mismo contenido variando algunos parámetros o se quiere recuperar para compartirlo, se preferirá la generación determinística.

1.3.5 Algoritmos Constructivos contra Algoritmos de Generación y Prueba

Una distinción final se relaciona a cómo se acepta el contenido. Un algoritmo constructivo genera el contenido una única vez asegurándose de que cumpla cierta característica.

Por otro lado, los algoritmos de generación y prueba generan el contenido y comprueban su calidad, de ser correcto, se conserva, de lo contrario, se vuelve a generar o se corrige.

1.3.6 Generación Procedural Basada en Búsquedas

Esta característica yace dentro de la PCG de Generación y Prueba que, en lugar de aceptar o rechazar un contenido, lo evalúa mediante una función de calificación y lo procesa, dando cabida a implemetaciones basadas en algoritmos genéticos u otras metaheurísticas para encontrar el candidato adecuado.

1.4 Dificultades del uso de PCG

Uno de los principales problemas al usar PCG radica en el hecho de garantizar que éste tenga la mayor calidad posible, por lo que debe ser probado para identificar errores o puntos a mejorar.

La calidad es la adecuación que tiene un producto o servicio al satisfacer la característica o necesidad para el que fue creado.

En el caso de un ambiente virtual, la calidad sería qué tan cercano es el aspecto visual del ambiente a lo que intenta emular, sin embargo, ésto no basta, ya que también debe aportar un valor de entretenimiento o estético al producto, por lo que necesita ser interesante.

Los ambientes virtuales tienen algunas características que le ayudan a mantener esa inmersión, las cuales se verán a continuación.

1.4.1 Corrección

La corrección del contenido sería qué tan bien cumple con su función: Un laberinto es correcto siempre que tenga una salida, mientras que una ciudad debe garantizar que haya semáforos, edificios, banquetas, etc...

En el caso del aspecto visual de un ambiente, se tienen reglas básicas: No tener objetos sobrepuestos entre sí, evitar que estén flotando o que desafíen las leyes de la física, etc...

Se podría considerar además que el ambiente es correcto si su aspecto visual es muy cercano a su contraparte de la vida real. Sin embargo, esto está ligado a la percepción de cada individuo, por lo que es una característica poco precisa de evaluar.

Puede pasar también que no exista una definición de 'realismo' para el contenido, como sería el caso del nivel de un videojuego (Fig. 1.12). Tan sólo se puede trabajar apegándose a una teoría, a las necesidades del proyecto o imitando la realidad. He aquí algunos ejemplos:

- En el caso de un bosque, se busca que sea muy similar a un bosque real.

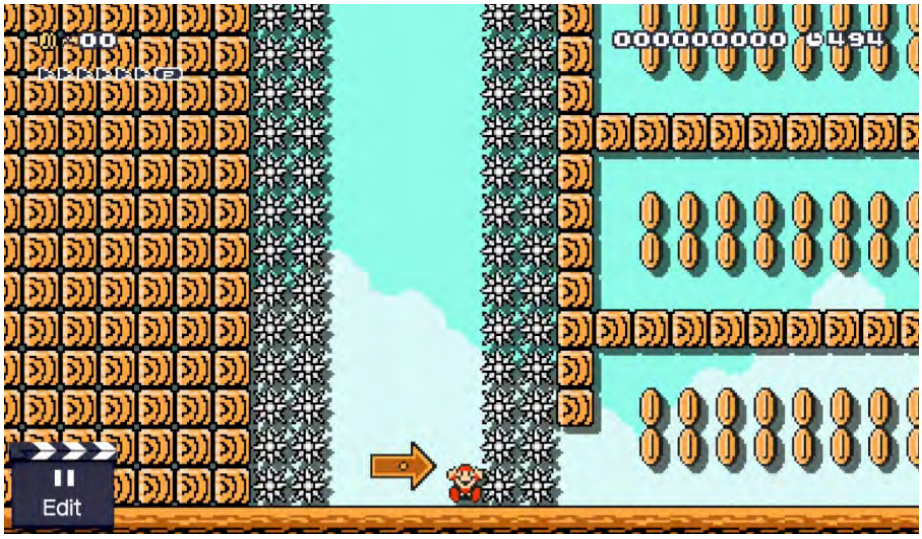


Figura 1.12: Nivel de un videojuego sin posibilidad de término, Super Mario Maker, Nintendo

- En el caso de un personaje, se busca que tenga proporciones adecuadas y anatomía correcta.
- En el caso del nivel de un videojuego de plataformas, se busca que el nivel pueda ser terminado.

En ocasiones, resulta mejor que el contenido sea generado correctamente aunque no sea tan interesante, que intentar crear algo interesante que tenga errores, pues los fallos son más percibidos que los aciertos.

Se debe estudiar el contenido necesario y opcional, y evaluar qué elementos son de vital importancia para que el contenido luzca auténtico. La retroalimentación del usuario es muy útil en este caso para corregir errores o malinterpretaciones.

1.4.2 Predictibilidad

RAE: Cualidad de predictible o predecible. Ya sea en sus características, comportamiento, etc...

Nos interesa que los ambientes no sean predecibles: Un entorno altamente repetitivo acaba rápidamente con el interés del usuario, sobre todo si se trata de un ambiente visual.

Figura 1.13: Ejemplo de una ciudad con alto grado de predictibilidad:

Sin embargo, que el ambiente sea impredecible no implica que sea interesante. Un entorno muy "variado" puede tomarse por caótico, reduciendo la inmersión y calidad del ambiente.

Esta es quizá la característica más subjetiva y compleja de evaluar, ya que requiere un proceso de prueba con usuarios largo. No sirve que nosotros consideremos que es interesante, pues al final, es el usuario quien consumirá el producto.

1.4.3 Replicabilidad

La replicabilidad distingue si podemos recuperar el ambiente generado.

La forma más sencilla de garantizarla es usar una función de ruido ligada a una semilla inicial, y:

1. Modificar las características del contenido según los números aleatorios generados por la función.

2. Diseñar el algoritmo de modo que repita las mismas operaciones dados los mismos parámetros.

Si garantizamos 2, la secuencia que genera el algoritmo es la misma (Pues es una función de estado). Asímismo, por 1, el contenido tendrá las mismas características si se repite la secuencia, garantizando así que el contenido que obtenemos es la misma.

1.5 Evaluación de los ambientes

El contenido debe evaluarse para determinar qué combinaciones de parámetros generan mejores resultados, identificar posibles fallos y aspectos a ser mejorados.

El cómo probar los ambientes recae en manos del desarrollador, y las aproximaciones más comúnes se exponen a continuación.

1.5.1 Retroalimentación por el usuario

Quizá la aproximación más utilizada para evaluar la calidad de nuestros ambientes es preguntar directamente al usuario por su opinión respecto al contenido generado, ya que éste último es el que va a consumir el contenido.

En [7] se propuso una forma de evaluación mediante encuestas a usuarios con alta afinidad con los videojuegos.

Sin embargo, las evaluaciones mediante cualidades como interés, belleza o realismo suelen ser subjetivas y carentes de explicación concreta, por lo que pueden no dar información suficiente sobre qué componentes del algoritmo fallan o qué parámetros son los más adecuados para que el contenido alcance una calidad plena.

1.6 Funciones de evaluación

Sea X el conjunto de posibles elementos representando una solución al problema. Una función de evaluación o calificación es una función f : X -> $\mathbb{R}$ o f : X -> $\mathbb{R}^{\kappa}$ que evalúa una característica. En Ciencias de la Computación, suele usarse en conjunto

con diversas técnicas (Como cómputo evolutivo o metaheurísticas) para buscar el candidato que optimice dicha función.

El diseño de estas funciones no siempre es una tarea trivial, pues el desarrollador tiene que decidir qué es exactamente lo que le interesa evaluar del contenido y formalizarlo. Incluso teniéndola, puede que no se hayan contemplado algunos detalles que los usuarios sí notarán, volviendo largo el proceso de prueba.

Por otro lado, si consideramos la calidad del contenido como qué tan inmersos están los usuarios en él, surgen nuevos problemas. La inmersión conlleva un estado emocional en un usuario que no se puede formalizar con facilidad, y ésto lleva al desarrollador a trabajar asumiendo una definición que no siempre puede ser la más adecuada.

En [12], se identifican tres distintas clases de funciones de evaluación para calificar la experiencia generada por el contenido:

1.6.1 Funciones de evaluación directa

En este tipo de funciones, algunas características del contenido son mapeadas directamente a un valor de aptitud. i.e. se podría usar el número de salidas de un edificio para evaluar qué tan seguro es, o evaluar la dificultad de un laberinto según cuántas salidas tenga.

Este mapeo depende en gran medida del problema tratado y de la interpretación del desarrollador. De estas funciones, es importante distinguir entre dos subtipos:

Guiadas por teorías:

En las que el desarrollador se guía por intuición o por algún tipo de teoría cualitativa a aplicar sobre su contenido para derivar la función de evaluación.

Guiadas por datos:

Aquellas en las que se recaba una gran cantidad de información obtenida de cuestionarios o pruebas al usuario sobre el contenido, luego es introducida a la función, la cual produce una salida según los datos.

1.6.2 Basadas en simulaciones

En ocasiones, y en especial con contenido interactivo, es necesario evaluar la experiencia [17] que el contenido genera en el usuario. En las funciones basadas en simulaciones, se da el contenido generado a un agente de IA para que lo 'experimente'.

El rendimiento es entonces usado para evaluar qué tan apto es el contenido según la experiencia que se esté diseñando. i.e., se podría crear un laberinto, y calificar su dificultad según cuánto tiempo en promedio se tarda el agente en resolverlo.

1.6.3 Funciones de evaluación interactivas

Estas funciones son similares a las basadas en simulaciones, sólo que utiliza como agentes de experimentación a los usuarios objetivo del contenido. Durante o después de la interacción con el contenido, se recaba información directamente mediante cuestionarios o reacciones, o implícitamente, evaluando algunas características de la interacción resultante. i.e. ¿Cuánto se tardó un jugador en salir de un laberinto?.

Sin embargo, no siempre es factible esta forma de evaluación, pues medir las reacciones de un jugador puede requerir de aparatos o estudios especializados que no siempre están al alcance de un desarrollador independiente.

1.7 Problemática planteada

Una de las principales ventajas del uso de PCG es la disminución de la carga de trabajo en la creación de proyectos. Con la proliferación de herramientas de trabajo gratuitas como Blender [14], Unity[15] o Unreal[16], se ha extendido la capacidad de creación de contenido multimedia como videojuegos, animaciones y piezas de arte digital. Como consecuencia directa, el desarrollo de proyectos es cada vez más asequible y los equipos de trabajo independientes, que carecen de un presupuesto lo

[14] Programa de animación y modelado 3D gratuito

[15] Motor de juegos que permite la programación gráfica y de entornos dinámicos usando el lenguaje C#

[16] Motor de juegos que permite la programación gráfica y de entornos dinámicos usando el lenguaje C++

suficientemente alto para la producción de contenido hecho por profesionales, necesitan alternativas para la creación de su contenido que les permitan competer con efectividad en la industria.

Un candidato potencial en varios ámbitos de esta industria es la generación procedural de contenido (PCG), que permite al desarrollador crear contenido mediante un algoritmo, dándole una herramienta adaptable con la que puede generar una cantidad muy alta de contenido.

En algunos casos el contenido requerido es un ambiente urbano virtual, por lo que tener un algoritmo que sea capaz de generar dichos ambientes puede ser muy útil en el desarrollo de, por ejemplo, un videojuego, una animación o una simulación que tomen lugar en una ciudad.

La primer parte del problema es crear un algoritmo que cumpla tres aspectos teóricos del contenido PCG de calidad: Corrección, impredicibilidad y replicabilidad.

La segunda parte es evaluar la calidad de los ambientes generados por el algoritmo. El uso de funciones de evaluación y la retroalimentación directa del usuario nos da perspectivas sobre qué tanta calidad tiene el contenido generado por el algoritmo según el público objetivo y según el desarrollador.

La creación de un ambiente virtual que cumpla esas características y el como analizar su calidad sigue siendo un problema ampliamente tratado y las soluciones se relacionan estrechamente con las áreas de inteligencia artificial y cómputo gráfico.

1.7.1 Hipótesis

Varias de las fuentes encontradas [7],[8],[9] no ofrecen suficientes recursos de visualización para evaluar cuál puede ser el mejor. Las gramáticas no son el método elegido en muchas de ellas, aún cuando su percepción del entorno las hace buenas candidatas como técnica de generación. Las fuentes encontradas que sí las usan [6], no proporcionan ningún estudio ni forma de visualizar en detalle los resultados.

Al usar un algoritmo PCG, es importante probarlo para identificar posibles errores y mejorar la calidad del contenido generado. El tener una función que califique teóricamente el contenido no es suficiente, ya que carece de la retroalimentación del

público objetivo. Asímismo, a ésta última le suele faltar algún tipo de justificación formal al momento de emitir su juicio. Al tener ambas ventajas y desventajas, la conjunción de éstas puede probar ser más útil.

Teniendo ésto en cuenta, la siguiente hipótesis es formulada:

La generación procedural de ambientes urbanos virtuales usando gramáticas es capaz de generar ciudades de buena calidad para un público familiarizado con ciudades, ofreciendo un alto grado de control sobre el contenido generado y siendo fácil de usar.

En contraste, la hipótesis nula es la siguiente:

El uso de gramáticas no es capaz de generar ciudades de calidad suficiente, o el control que ofrece el usar esta aproximación resulta en un software complicado de usar, para el que es mejor modelar manualmente la ciudad.

1.7.2 Objetivos

Se creará un programa que permita generar, mediante un algoritmo PCG, ambientes urbanos virtuales que asemejen una ciudad o metrópolis sobre terreno semi llano. El algoritmo debe ser capaz de conseguir un nivel aceptable de calidad en el ambiente generado para su uso en animaciones, videojuegos o simulaciones. Dicha calidad se evaluará de 2 formas: La teórica y la práctica.

La teórica evaluará los ambientes con funciones de evaluación que midan la corrección, impredecibilidad y replicabilidad del contenido.

La práctica se evaluará midiendo el interés generado por las ciudades mediante encuestas y comentarios, luego de haber mostrado parte del contenido que el algoritmo es capaz de generar.

Comparar ambas calidades evalúa el entendimiento del desarrollador sobre el contenido, y permite identificar errores y puntos a mejorar.

El algoritmo también debe ser fácil de usar, con opciones para personalizar la creación del contenido, sin tener que realizar búsquedas exahustivas de los parámetros óptimos de generación.

Este proyecto pretende demostrar el potencial de la generación procedural en el área de cómputo visual y dar una visión general al uso de las Ciencias de la Computación en la generación de ambientes virtuales.

Capítulo 2

Generación Procedural de Ambientes urbanos virtuales

2.1 Caracterización de los elementos de la solución

Como primer aproximación, se debe comenzar definiendo de qué se conforman los objetos que dan solución al problema.

2.1.1 Ciudades

El conjunto de posibles soluciones que generará el algoritmo PCG son, entonces, ciudades virtuales. Más en concreto, el aspecto visual de estas. Sea C nuestro conjunto de ciudades virtuales.

En [14] se proporciona una caracterización de ciudades analizando el potencial de un ambiente urbano para ser sustentable e 'inteligente', y enlista los siguientes componentes:

- Nivel de desempleo en jóvenes
- Nivel de educación
- Áreas verdes
- Ocupación del terreno
- Servicios Públicos

- Medios de transporte
- Uso de estrategias inteligentes de crecimiento
- Disponibilidad de incentivos públicos
- Fuentes de energía
- Esperanza de vida promedio
- Medios de Comunicación

De estos, se extraerán aquellos que puedan ser representados gráficamente y se excluirán los que necesiten una simulación para mostrarse (Vehículos, peatones, etc...):

- Áreas Verdes
- Servicios Públicos
- Ocupación del terreno
- Medios de transporte
- Fuentes de energía

Cada uno de estos elementos tiene su propio conjunto de objetos que se verán individualmente.

Se necesita también evidencia visual que respalde la 'apariencia urbana' del paisaje, esta se logra a través de la distribución que tienen en relación al núcleo central [15], así como los imponderables que van unidos a estos, como la gradual ausencia de elementos naturales, la atmósfera vivencial y el tránsito.

Así, mientras más nos alejemos del centro de la ciudad, menor altura tendrán los edificios y más áreas verdes y residenciales habrá.

Con ello, llegamos a la siguiente definición. Una malla 3D de ciudad $c \in C$, consta de lo siguiente:

- Un terreno sobre el que la ciudad se generará.

- Un conjunto de distritos que dividen la ciudad por zonas.
- Una red de caminos o calles, simbolizando cruces y conexiones entre estos.
- Una red de manzanas, obtenida a partir de ciclos en los cruces de la red de caminos.
- Un conjunto de parcelas que dividen la manzana.
- Un conjunto de edificios colocados en las parcelas.

Los distritos ayudan a crear una diferenciación por áreas en los edificios y servicios. Se generarán ciudades con alta densidad de población, por lo que el contenido generado estará ligado a ciudades con éstas características (por ejemplo, su crecimiento seguirá un plan hipodámico, habrá poco espacio entre edificios y estos serán altos, etc..).

La generación de cada componente se divide por capas, donde cada capa es colocada encima de la anterior (Figura 2.1).

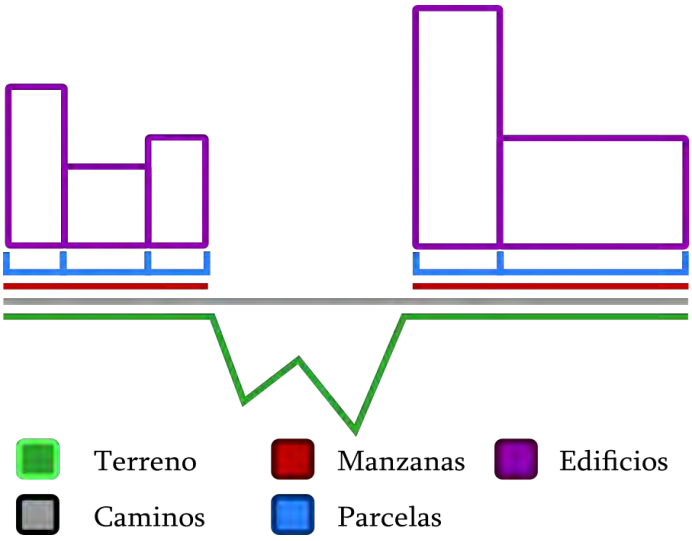


Figura 2.1: Distintas capas de la ciudad.

Por ejemplo: Los edificios son colocados encima de las parcelas, los caminos encima del terreno, etc...

2.2 Generación de contenido a partir de elementos característicos

A continuación se describe qué tipo de contenido genera el algoritmo, y se caracterizan según los elementos que componen cada capa.

2.2.1 Contenido a generar

Gran parte del contenido generado es gráfico, por lo que es de particular interés la construcción de mallas 3D.

Una malla 3D (Fig. 2.2) es un conjunto de vértices en un espacio, generalmente $\mathbb{R}^3$, unidos mediante aristas y rellenadas por una superficie sólida denominada polígono. La construcción de éstas mallas da como resultado un modelo 3D que puede servir para representar un objeto.

A cada uno de éstos vértices corresponde un vértice 'uv' que es mapeado a un espacio E, $E \in \mathbb{R}^2$ y que los algoritmos de graficación utilizan para consultar de qué color será pintado dicho polígono de ser visible por una cámara virtual.

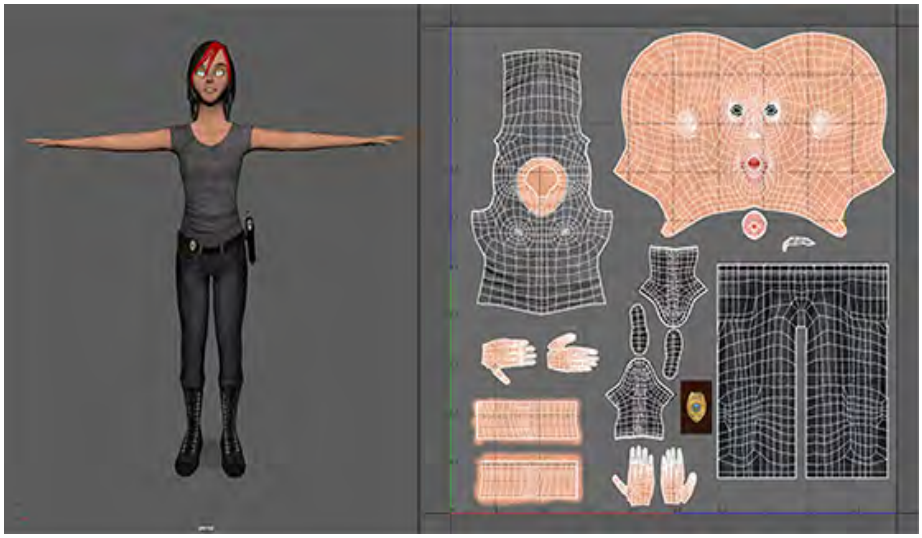


Figura 2.2: De izquierda a derecha: Malla de modelo 3D texturizada y su respectivo mapeo

Cada vértice y polígono tienen además un conjunto de vectores que indican en qué dirección están 'orientados'. Dicho vector se utiliza para cálculos de iluminación y puede ser utilizado para simular relieves.

Para determinar el aspecto final del objeto, la malla hace uso de 'materiales', que codifican el algoritmo de renderizado o 'shader', que utiliza las texturas, mapas de normales, etc...

El contenido a generar consiste entonces de un conjunto de éstas mallas de modelo representando todos los elementos que se encontrarían en un ambiente urbano real.

A continuación, se enumeran qué mallas de modelo forman parte de qué capas.

2.2.2 Superficie de terreno

El terreno es una extensión de tierra delimitada por una frontera, generalmente física.

En una ciudad existen elementos no perceptibles del todo, como redes subterráneas (agua, drenaje, gas, etc..) que yacen por debajo de la superficie. Puesto que no son visibles en su mayoría, no se generarán.

El terreno será la superficie sobre la cual se colocarán las demás capas que componen la ciudad. Abarcará también el agua y la vegetación. Consta de lo siguiente:

- Una malla 3D representando la superficie del terreno.
- Una malla 3D representando el agua.
- Un conjunto de mallas 3D representando la vegetación.

2.2.3 Red de caminos

Camino o calle

Un camino o calle es una franja de terreno usada para trasladarse de un punto a otro. Tiene un principio y un fin, ambos denominados cruces.

Cada camino estará formado por los siguientes valores:

Nombre del valor	Tipo del Valor	Utilidad
Nodo inicial	v $\in$ $Nodos(G)$	Cruce que marca el inicio del camino.
Nodo final	v $\in$ $Nodos(G)$	Cruce que marca el final del camino.
Atributo	v $\in \mathbb{R}^3$	Tupla de valores simbolizando el ángulo, el largo y el ancho del camino.

Dadas las características de las ciudades que se busca emular, los materiales de las calles simularán asfalto, y los de las banquetas, cemento o concreto.

Red de caminos

Al conjunto de caminos y cruces que sirven para recorrer el terreno se le denomina red de caminos.

Ésta tiene una clara similitud con los grafos, siendo los nodos equivalentes a los cruces y las aristas a las conexiones entre éstos. Internamente, se usará esta estructura para representar la red de caminos.

Las mallas de modelo generadas serán las siguientes:

- Un conjunto de mallas 3D de puntos iniciales, finales y cruces de los caminos.
- Un conjunto de mallas 3D representando de las conexiones entre los cruces.

2.2.4 Manzanas

Una manzana es un espacio urbano delimitado por caminos o calles por todos lados. Está representada por el siguiente conjunto de datos:

Nombre del valor	Tipo del Valor	Utilidad
Tipo primario	n $\in \mathbb{Q}$	Determina el tipo de parcelamiento a realizar sobre la manzana.

Una manzana es fácilmente representada como un polígono preferiblemente regular, ya que la división en partes iguales de ángulos permite modelar fácilmente el

contenido dentro de esta.

Los tipos de las manzanas serán determinados según la densidad de población y el distrito en que la manzana caiga.

La red de manzanas constará de lo siguiente como contenido final:

- Una malla 3D representando las banquetas de cada manzana.
- Una malla 3D representando el patio interno de cada manzana.
- Una malla 3D representando los callejones (En caso de existir).
- Un conjunto de materiales texturizados aplicados a éstos.

2.2.5 Parcelamiento de la ciudad

Una parcela es un área destinada para un uso (ya sea vivienda, comercio, etc.) Las parcelas (o lotes) deben tener acceso a una acera, banqueta, o vialidad.

Las parcelas tendrán los siguientes atributos:

Nombre de valor	Tipo de Valor	Utilidad
Polígono	Arreglo de vectores v $\in \mathbb{R}^3$	Puntos representando la base de la parcela.
Altura	n $\in \mathbb{R}$	Altura del edificio que irá dentro de la parcela.
Tipo	n $\in \mathbb{Q}$	Tipo de parcela (Edificio que aloja).

Las parcelas son un concepto meramente abstracto. Similares a una frontera, no tienen representación física en el mundo real y por consiguiente no generarán mallas de modelo.

2.2.6 Edificios

Un edificio es una construcción firmemente unida al piso que provee refugio parcial o total a actividades humanas, máquinas u equipamento [16].

Los edificios serán caracterizados con los siguientes valores:

Nombre del valor	Tipo del Valor	Utilidad
Posición	$v \in \mathbb{R}^3$	Posición del edificio.
Alcance	$v \in \mathbb{R}^3$	Región ocupada por el edificio.
Rotación	$v \in \mathbb{R}^3$	Rotación del edificio en cada uno de los ejes.

Cada edificio consta de lo siguiente como producto final:

- Un conjunto de mallas 3D representando el edificio.
- Una malla 3D representando el patio.

2.3 Calidad en ambientes urbanos virtuales

La calidad de los ambientes generados será determinada por el grado de corrección, impredecibilidad y replicabilidad del mismo.

2.3.1 Evaluando la corrección

RAE : 2. f. Cualidad de correcto. Escribir, expresarse, comportarse con corrección.

La corrección se evalúa dependiendo de para qué será usado el contenido generado; Ya que se está evaluando el aspecto gráfico de una ciudad, el 'realismo' o 'qué tanto se parece a una ciudad real' es lo más adecuado al medir su corrección. Aquí se define teóricamente, y en el capítulo de diseño se entra en detalles sobre la evaluación.

Los componentes de la ciudad (edificios, parcelas y caminos) simulan todos un espacio físico, y por ello deben tener las siguientes características:

- Un componente no debe ocupar el mismo espacio que otro (i.e. un edificio atravesando otro o un semáforo incrustado en un árbol).
- La colocación de los componentes debe tener coherencia física (i.e. no tendría sentido ver un edificio flotando o apoyado en una esquina)

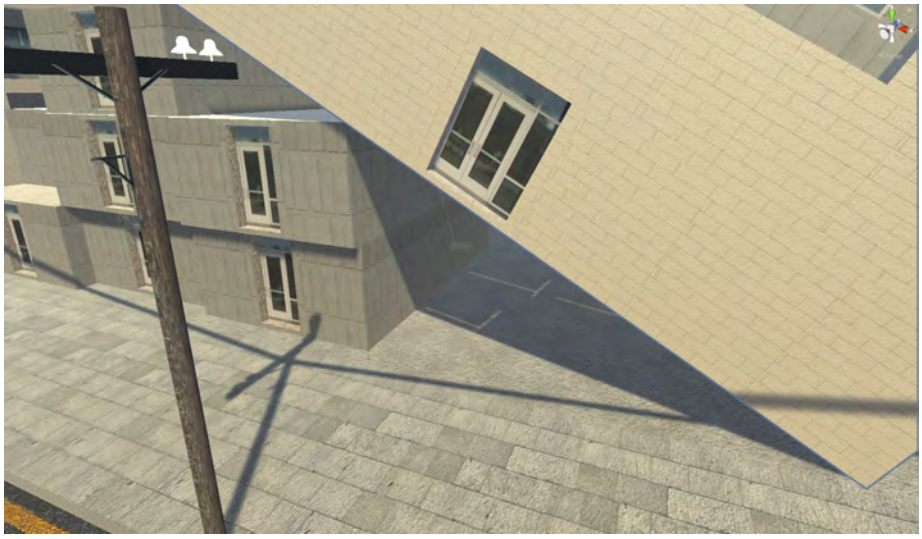


Figura 2.3: Ejemplo de edificio mal colocado.

Los componentes de la ciudad deben estar correctamente colocados: Sería extraño ver un semáforo colocado en un edificio, o un árbol en medio de la calle. Para ello, se usarán las distintas capas de la ciudad, por lo que la colocación debe cumplir lo siguiente:

- Un componente no puede estar en una capa que no le corresponde.
- Un edificio particular a un distrito no puede estar en otro (Un rascacielos no puede estar en un distrito industrial)

De no cumplir con estas cualidades, la corrección total del ambiente se verá afectada.

Finalmente, un poco menos perceptible pero no menos importante, es el aplicar elementos teóricos a como crecerá la ciudad, cómo serán colocados los edificios, etc...

Una definición [13] nos plantea una caracterización basada en un 'centro' urbano y en su interacción con celdas contiguas a él. A medida que nos alejemos del centro de la ciudad, los edificios, distritos y áreas verdes se verán afectados. Estas características se simplifican, quedando las siguientes:

Figura 2.4: Ejemplo de postes de poste de luz en medio de la calle o incrustados en edificios.

- Coherencia 'económica': Las ciudades tienen presupuesto limitado para ir expandiéndose, por lo que no tiene sentido colocar muchos elementos componentes costosos ni baratos, pues no coincide con cómo crece una ciudad. Deben colocarse de manera equilibrada, y esto se ve reflejado en distintas características de los componentes de la ciudad como la altura de edificios, ancho de los caminos, materiales usados, etc...

- Coherencia en escala: Dependiendo del tamaño de algunos componentes, algunas reglas tienen que aplicarse: i.e. una manzana muy pequeña no puede alojar edificios, un edificio no puede ser demasiado alto.

- Coherencia distributiva: Como hemos visto, las características de los componentes varían en función de qué tan lejos están del centro de la ciudad. El algoritmo PCG deberá poder emular correctamente esta propiedad.

- Coherencia de forma: Algunos componentes pueden cumplir con las propiedades anteriores pero no tener cabida en el mundo real: i.e., una calle que vaya girando en espiral, un edificio en forma de zig zag, un poste con forma de tornillo, etc...

Estas características determinan qué parámetros incluir en el algoritmo para que el desarrollador tenga control sobre el contenido generado.

2.3.2 Replicabilidad en ambientes urbanos virtuales

Para garantizar la replicabilidad del contenido, se diseñará una función de ruido de modo que cumpla las características expuestas en 1.4.3.

Sea E el conjunto de todas las posibles parametrizaciones del algoritmo (entradas, semillas y parámetros), sea e∈E, sea $s_0 \in \mathbb{N}$ la semilla de e.

Sea f una función de ruido escalonada cuyo estado inicial es $s_1 = f(s_0,e)$. Sean g_1, g_2,..., g_n tal que:

$$f(s_j,e) \begin{cases} g_1(s_j,e) & \text{Caso 1.} \\ g_2(s_j,e) & \text{Caso 2.} \\ ... & ... \\ g_n(s_j,e) & \text{Caso n.} \end{cases} \tag{2.1}$$

Cada que se utilice f en el algoritmo, se guarda la 'sub' función g_k utilizada. Sea $Seq(e)=\{g_{k_1}, g_{k_2}, ..., g_{k_j}\}$ la secuencia de operaciones generada por el algoritmo dado e. El algoritmo es replicable si:

- Para cualquier e_i, $e_j \in E$. Si $e_i = e_j \rightarrow Seq(e_i) = Seq(e_j)$

2.3.3 Predictibilidad de una ciudad

Aunque el contenido debe tener variedad, también debe conservar la corrección, lo que limita el tipo de contenido que pueden generar los módulos: Un edificio cuyos pisos tengan un número aleatorio de paredes es impredecible, pero no correcto.

Incluso si es correcto, éste no siempre será 'interesante'. Se debe procurar que el ambiente generado no sea repetitivo y ofrezca algún tipo de valor para el usuario.

Dependiendo del tipo de ciudad generada, la inclusión de patrones podría ser obligatoria. Por ejemplo, el plan hipodámico (Manzanas cuadradas/rectangulares, caminos crecen siempre en 90°) provoca una baja impredecibilidad en la ciudad, pero aún así no afecta el realismo de la ciudad, ya que es considerado normal.

El problema surge al momento de colocar elementos con alto nivel de unicidad, en particular, los elementos colocados dentro de las manzanas, como edificios, parques

y demás, necesitan distinción de otros para no juzgar el ambiente como repetitivo.

Los algoritmos PCG hacen un uso extensivo de generadores de números aleatorios, por lo que la mayoría de contenido es teóricamente distinto, pero esto no garantiza que dicho contenido no pueda ser juzgado como tal. Un edificio diferente a otro puede ser considerado repetitivo si las diferencias son mínimas y se muestran con mucha frecuencia.

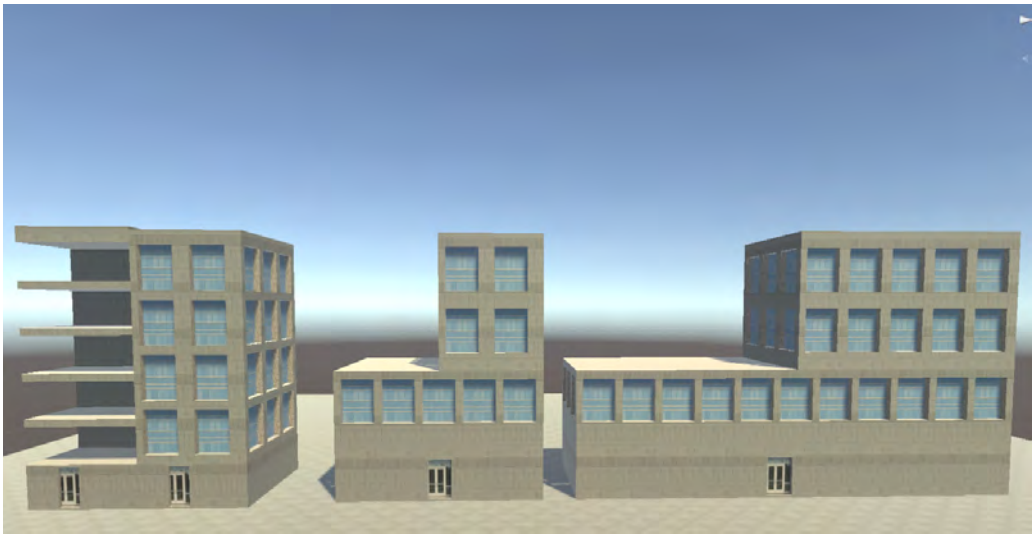


Figura 2.5: Ejemplo de edificios que, si bien no son iguales, quizá puedan considerarse repetitivos.

Puede que un edificio se repita, pero si están colocados muy lejos el uno del otro, es muy probable que pase desapercibido.

Esto nos lleva a definir un índice de parentezco, que determina qué tan similares son ciertos componentes de las capas en base a qué tan similares son en forma, tamaño, ubicación, distancia entre ellos, etc...

Se toma entonces el promedio del índice de parentezco de todos los componentes de la ciudad c_i, $\mathrm{Ip}(c_i) \in \mathrm{R}$. Mientras menor sea su valor, más impredecible se considerará la ciudad i. No se calificará el interés generado de manera teórica ya que sería muy subjetivo.

2.4 Trabajos Previos y Herramientas

Al estar creando un programa para su uso en proyectos de ambientes virtuales, es necesario tener una visión general de lo existente teórica y comercialmente, para evitar redundar o fallar al diseñar el algoritmo.

Este análisis nos permitirá también ofrecer otra implementación que tenga elementos que las demás no tengan, ofreciendo a los desarrolladores alternativas a la creación de ciudades usando PCG.

2.4.1 PCG ciudades

En [6] se proporciona una implementación basada en gramáticas y sistemas L para la generación de ciudades. Sin embargo, sólo se pueden ver algunas imágenes y el programa no está disponible públicamente, sumando que no hay ningún tipo de retroalimentación del usuario, se vuelve complicado saber qué tan bueno es el contenido generado.

Aún así, es un trabajo completo, donde se exploran varios aspectos del uso de PCG aplicado a ciudades en terreno llano.

Junto con [5], se puede obtener un panorama general de las técnicas para generación de ambientes urbanos virtuales más usadas en tiempos recientes.

2.4.2 Urbis Terram

[8] sugirió el uso de funciones de aptitud aplicadas a ciudades, aunque sólo de forma teórica. Al no poseer una implementación completa, carece de detalles en las técnicas y parámetros usados.

2.4.3 City Generation Perlin Noise

Aquí se utiliza una función de ruido para determinar globalmente distintos aspectos de la ciudad, como la altura de edificios, la división de distritos, etc...

Si bien este trabajo consigue su propósito, no ofrece una prueba del trabajo y el detalle de las ciudades generadas no es muy complejo. Sin embargo, sí analiza la calidad de los ambientes generados mediante encuestas a jugadores de videojuegos.

2.4.4 CityEngine

El marco de trabajo más potente de generación procedural de ciudades, CityEngine ofrece una interfaz de usuario para la creación de ciudades altamente parametrizable, siendo capaz de generar ciudades de aspecto realista, además de poder usar datos proporcionados por Sistemas de Información Geográfica (SIG's).

Sin embargo, el usuario tiene que realizar bastantes modificaciones e introducir muchos datos para generarla, por lo que buscar los parámetros adecuados para la ciudad puede ser complicado, y claro, sólo disponible con todas las funciones si se paga la licencia de $4,000 dólares por equipo.

El proyecto expuesto en esta tesis es creado como un punto medio para un desarrollador que desee generar ciudades usando PCG: Altamente parametrizable sin necesitar formación ni conocimiento especializado, analiza la calidad desde los puntos de vista del usuario y del desarrollador, y ahonda en detalles sobre las técnicas y parámetros utilizados a lo largo del desarrollo, todo esto siendo asequible para el desarrollador independiente.

Capítulo 3

El diseño del algoritmo

Ya definidos todos los elementos necesarios para la implementación, se procede a diseñar el algoritmo y el sistema que harán uso de las técnicas y enfoques expuestos para construir las ciudades virtuales.

Esto no sólo evita contratiempos y errores, sino que permite formalizar las ideas para no desviarse del objetivo inicial.

3.1 Entrada del algoritmo

La entrada es la parte de la parametrización del algoritmo que tiene el impacto más notable en el contenido generado, y forma parte importante del control que se tiene en el algoritmo.

La entrada al algoritmo estará dada por mapas (imágenes) [19], similares a los mencionados en [6], los cuales determinan distintas características que tendrán los componentes de cada capa. Éstos son:

- Mapa de densidad de población
- Mapa de distritos
- Mapa de terreno
- Mapa de ruido

Los mapas determinan también la dimensión que tendrá el ambiente generado en metros, con cada píxel representando una región de un metro cuadrado, así, un mapa de 1200x1200 píxeles generará una ciudad de 1200x1200 m^2.

Como entrada se da también un conjunto de distritos que determinan las características del contenido generado en ellos.

3.1.1 Mapa de densidad de población

La densidad de población se proporcionará con una imágen de n×m en escala de grises donde el valor promedio de los componentes RGB de cada píxel ($\frac{R+G+B}{3}$) determinan la densidad de población en la región representada por el pixel (Fig. 3.1).

Mientras más cercano al color blanco (255) esté el valor promedio del pixel, más población tendrá la región de terreno resultante, mientras que si es más oscuro (0), menos tendrá.

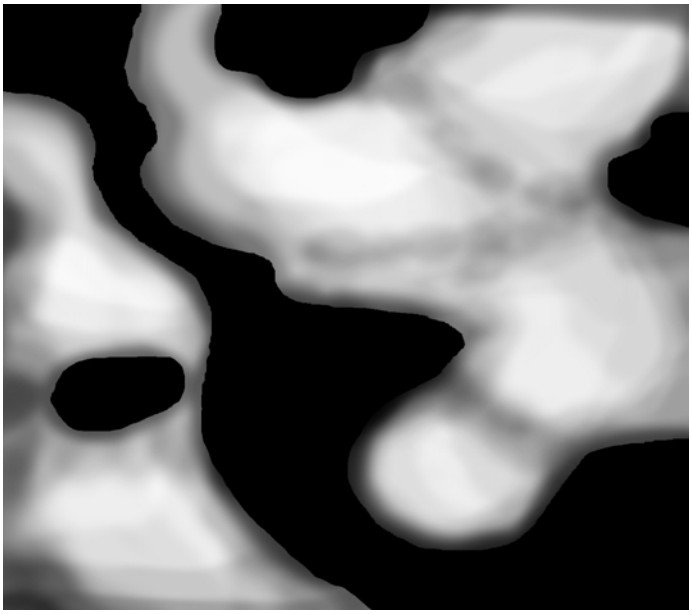


Figura 3.1: Ejemplo de mapa de densidad de población

.

Impacto de la densidad de población en la generación de la ciudad

La densidad de población está estrechamente relacionada con el tráfico de vehículos y peatones en una región, lo cual se extrapola a un incremento directo en la magnitud de los elementos en cada capa: Algunos caminos se ensancharan para permitir el paso de más vehículos, y algunos edificios, como los rascacielos, sólo aparecerán en zonas de alta densidad poblacional.

Se planea que la densidad de población afecte las siguientes capas de la ciudad:

- Caminos : Los caminos tendrán probabilidad de eliminarse o reducir su grosor debajo de cierta densidad de población, debido a un flujo de tráfico menor.
- Manzanas: La densidad afecta el tipo de parcelamiento que se realizará en la manzana y la probabilidad de aparición de los tipos de edificios que contendrá la manzana, debido a que es mejor colocar ciertos edificios donde pase más gente.
- Parcelas: La densidad de población afecta la altura de los edificios creados. A mayor densidad, más espacio se necesita para llevar a cabo actividades humanas.

3.1.2 Mapa de terreno

El mapa de terreno es una imagen en escala de grises donde el valor promedio de los componentes RGB de cada píxel ($\frac{R+G+B}{3}$) determina la altitud del terreno en la región representada por el pixel (Fig. 3.2).

Mientras más cercano al blanco (255) sea el valor promedio del pixel, mayor altitud tendrá el terreno generado en la región, mientras que a más oscuro (0), menor altitud tendrá. Debido a que el algoritmo será diseñado para generar ciudades sobre terreno llano, sólo se colocarán edificios en la altura máxima (255).

Se pueden tomar distintas aproximaciones para modelar el crecimiento de la ciudad según el terreno, como crecer hacia el sitio con pendiente menos pronunciada [6], debido a que acarrea menos costos en el aplanado y colocación de edificios y redes de agua/electricidad.

Figura 3.2: Ejemplo de mapa de terreno

Por simplicidad, el terreno afectará unicamente el relieve en las zonas verdes o acuáticas, mientras que será llano en los demás distritos.

3.1.3 Mapa de división en distritos

El mapa de división de distritos permite dividir la ciudad en zonas de distintas características o funciones, y se compone de una imagen a color con cada color representando un distrito:

Tipo de distrito	Color en el mapa	Valor RGB
Residencial	Rojo	(255,0,0)
Industrial	Violeta	(0,255,255)
Comercial	Amarillo	(255,255,0)
Zona Verde	Verde	(0,255,0)
Zona actuática	Azul	(0,0,255)

Con estos colores se pinta entonces una imagen separada en regiones (Fig. 3.3).

El mapa de distritos sólo se encarga de dividir la ciudad en zonas con un distrito

Figura 3.3: Ejemplo de mapa de zonas

asociado, pero no determina las características que tiene cada distrito ya que el desarrollador debe ser capaz de personalizar estos parámetros.

3.1.4 Distritos

La definición de distrito varía dependiendo del país en el que se esté, pero por lo general un distrito es una división política o administrativa de un territorio más grande (en este caso, una ciudad), y por lo general agrupan espacios de características similares o que están destinados al mismo tipo de actividades.

Lo anterior provoca que algunos edificios no aparezcan o tengan poca probabilidad de aparecer en algunso distritos: i.e. un centro comercial rompería la corrección del ambiente de estar generado en medio de un complejo de fábricas.

Edificios destinados a actividades similares suelen tener el mismo tipo de materiales y uso de espacio. i.e. en un distrito industrial, donde se suele mover maquinaria o vehículos amplios, se necesitan edificios amplios pero no muy altos, y deben estar construidos de materiales no inflamables para evitar accidentes.

Cada distrito afecta a los componentes generados en las siguientes capas:

- Caminos : No se generarán caminos vehiculares en áreas verdes y zonas acuáticas, ya que están destinados a otros fines.
- Manzanas : Tipo de manzana creada, ya que esta refleja el uso del espacio en la manzana y determina también el tipo de edificios que pueden generarse en él.
- Parcelas : Alturas mínimas y máximas de cada tipo de edificio, ya que refleja el uso del espacio.
- Edificios : Conjunto de materiales que tendrán los edificios.

El desarrollador necesita ser capaz de personalizar cómo se afectan estos componentes, dando origen a la siguiente parametrización:

Nombre de valor	Tipo de Valor	Utilidad
Color de identificación	Color RGB	Color que identifica al distrito.
Nombre	Cadena de caracteres ASCII	Nombre del distrito.
Materiales	Arreglo de materiales	Materiales que tendrán los edificios generados en el distrito.
Alturas	Arreglo de tuplas en $\mathbb{R}^2$	Arreglo de tuplas de altura (máx, mín) determinando las alturas máximas y mínimas para cada tipo de edificio generado en el distrito.

El color, junto con el nombre, da identificación única al distrito, ya que puede haber más de un distrito del mismo tipo, esto puede servir para colocar edificios únicos por distrito.

3.1.5 El mapa de ruido: Aleatoriedad en el ambiente

El mapa de ruido es una imagen en escala de grises donde el valor RGB promedio de cada pixel tiene una influencia en los elementos generados en la región de ese píxel (Fig. 3.4).

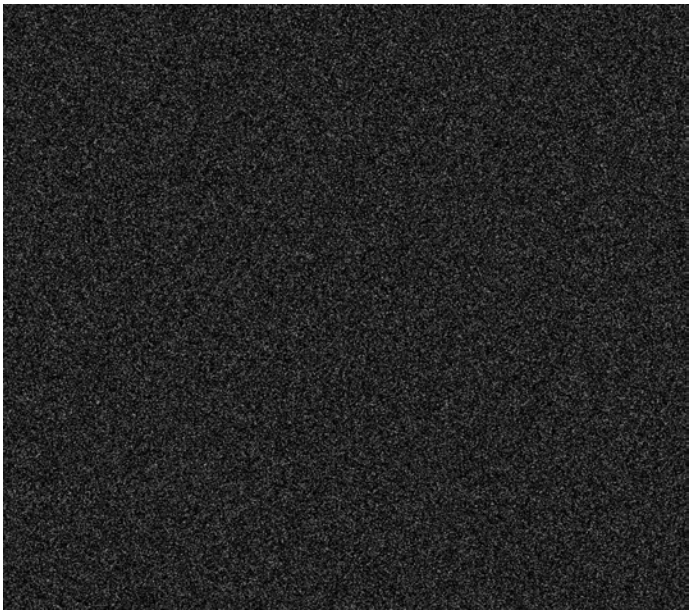


Figura 3.4: Ejemplo de mapa de ruido

Este mapa utilizará los intervalos de los parámetros que hay en algunos componentes (como la altura mínima y máxima de edificios) para determinar aleatoriamente el valor que les corresponde.

Este mapa también puede ser generado proceduralmente y es similar a un mapa de semillas, donde el valor de cada píxel es una 'semilla local' usada para generar el contenido de esa región.

3.1.6 Semilla de generación

Se dará además como entrada una semilla que afectará la generación aleatoria de múltiples elementos del ambiente de manera directa. El usuario podrá, más tarde, recuperar esta semilla del programa para replicar la ciudad.

La semilla (s_0) es un número entero de hasta 16 dígitos que se usará para generar números aleatorios durante todo el programa para darle unicidad a cada objeto generado; también da al algoritmo la capacidad de replicar los resultados, por lo que es imprescindible conservarla.

Aunque se podría calcular la semilla en base a los elementos de la entrada, hacer ésto no nos permitiría generar más de una ciudad usando los mismos parámetros. Si el usuario no queda satisfecho con la ciudad dada su entrada, puede cambiar la semilla para obtener un resultado diferente en lugar de variar los parámetros.

3.2 Diseño y parametrización del algoritmo.

La siguiente sección diseñará los distintos módulos a utilizar por el sistema para generar las capas de la ciudad, propondrá una técnica a usar por cada módulo y enlistará los parámetros modificables para controlar la salida de ellos.

3.2.1 Flujo del programa

El programa deberá ir creando la ciudad capa por capa:

1. Creación del terreno.
2. Creación de caminos.
3. Creación de manzanas.
4. Creación de las parcelas.
5. Creación de edificios en cada parcela.
6. Creación de vegetación.
7. Renderizado de ambiente.

El contenido creado en cada capa es manejado por un módulo que genera los datos usados por módulos siguientes (Fig. 3.5), genera las mallas de modelo del contenido. El módulo de caminos, por ejemplo, generará la red de caminos y su malla de modelo, y proporcionará la red como entrada al módulo de manzanas.

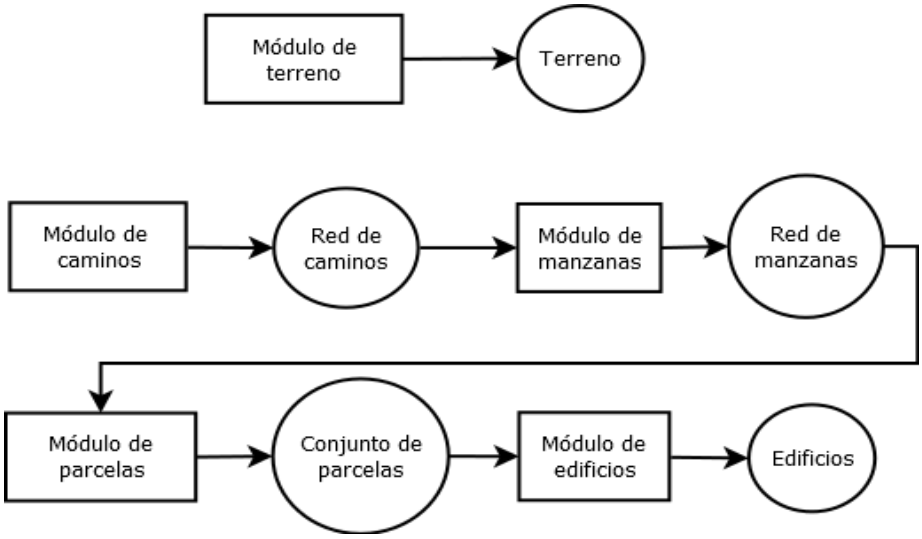


Figura 3.5: Flujo del Programa

3.2.2 Módulo de caminos

La red de caminos modelará un crecimiento en plan hipodámico (Fig. 3.6): El esquema de planeación urbano predominante en las ciudades contemporáneas, en donde los caminos crecen ramificándose en 2 o 3 nuevos caminos de 90, 180 y 270 grados en cada cruce, resultando en una mayoría de manzanas rectangulares.

La red de caminos empieza a crecer desde un punto inicial y elige cómo seguir mediante un conjunto de reglas, muy similar a el axioma y el conjunto de reglas de una gramática dependiente del contexto.

Sin embargo, usar una CDG no funcionaría ya que necesitamos 'percibir' todo el ambiente para determinar intersecciones entre los nuevos caminos generados y los ya existentes, y esto requeriría de tener un número muy elevado o incluso arbitrario de reglas.

Si a esto sumamos que cada nuevo cruce puede generarse de forma paralela, que el módulo de caminos requiere parametrizarse y percibir su ambiente, y este ambiente determina como crece (comunicación bilateral), un sistema 2L parametrizado es el candidato ideal para generar la red de caminos.

Se crecerá la red de caminos de forma similar a [6], generando nuevas ramas en

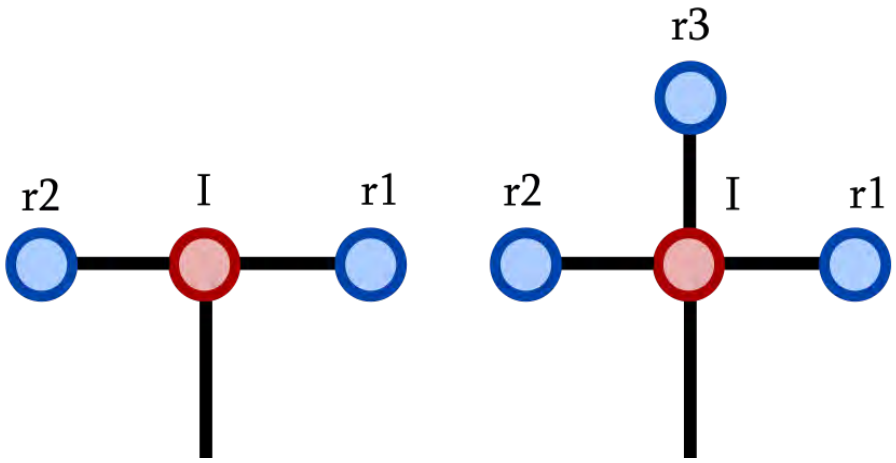


Figura 3.6: Ejemplo de ramificaciones para una red de caminos hipodámica.

un cruce mediante atributos globales: Tripletas (v_1,v_2,v_3), donde v_1 es el ángulo, v_2 es la longitud y v_3 es el ancho del nuevo camino a generar localmente a partir de un cruce I, ignorando algunas variables por fines de simplicidad

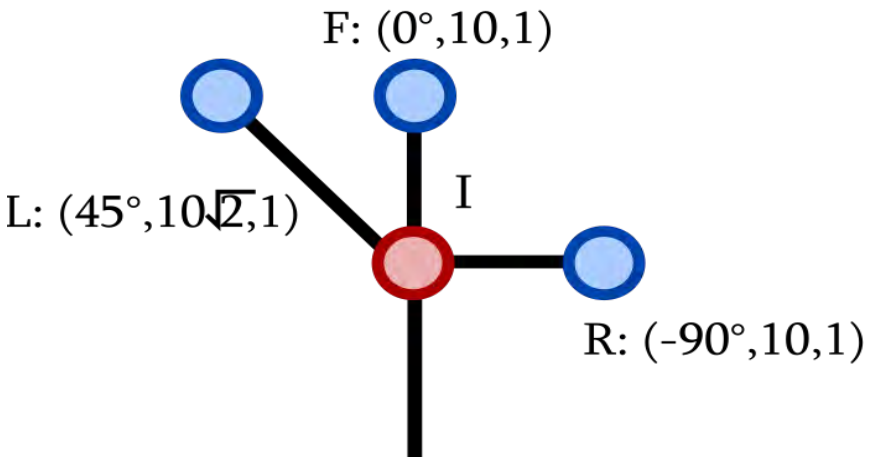


Figura 3.7: Ejemplo de nuevos cruces generados mediante el conjunto de atributos globales {L,F,R}

Para generar un nuevo cruce c_{i+1}, se agregan los valores de rotación, largo y ancho del atributo a c_i y se comprueba si c_i puede ser colocado en esa nueva posición.

Si el distrito donde el cruce se coloca es inválido, se modifica el atributo para intentar hacer válida su posición, probando las siguientes modificaciones:

1. Triplicar la longitud del camino si el distrito en que cae el cruce es una zona acuática (Para simular creación de puentes) Fig. 3.8(a). Fig. 3.8 (a).

2. Si cae en un distrito no transitable, rota el camino 45° a la derecha. Fig. 3.8 (b).

3. Rotar el camino 45° a la izquierda. Fig. 3.8 (c).

4. Reducir la longitud del camino a la mitad. Fig. 3.8 (d).

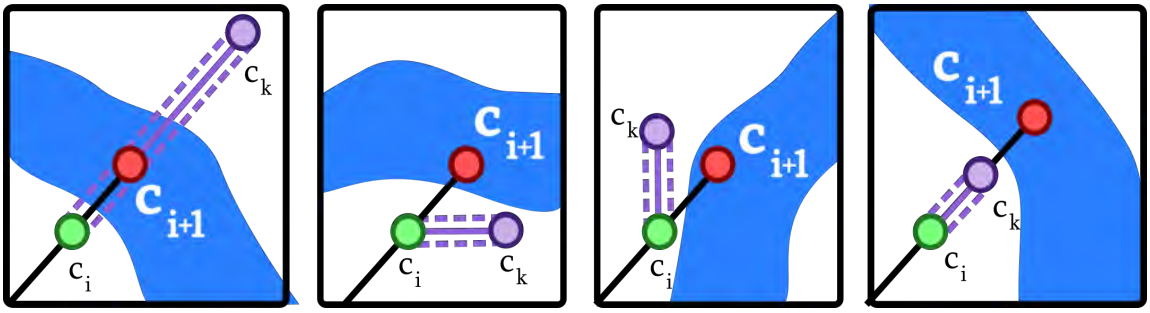


Figura 3.8: Distintos casos de adaptación de caminos al entorno: c_i es el cruce origen, c_{i+1} es el cruce inválido y c_k es la adaptación intentada.

De ser inválida en todos los esos casos, se omite la generación del nuevo cruce.

Parametrización

La parametrización debe dar al usuario el control sobre el número de caminos generados, sus características y de la influencia que tiene el mapa de densidad de población sobre la generación de la red, así como de las texturas a usar por la malla 3D:

Nombre de valor	Tipo de Valor	Utilidad
Caminos a crear	n$\in \mathbb{N}$	Número de cruces a crear por el módulo de caminos.
Punto de inicio	n$\in \mathbb{R}^3$	Punto del ambiente a partir del cual se empieza a crear la red de caminos.
Atributos de ramificación	Tupla (a_1,a_2,a_3) a_1,a_2,a_3 son atributos	Atributos de los nuevos caminos creados a partir de un cruce.
Umbral de eliminación	n$\in \mathbb{R}, n \in$[0,1]	Valor de densidad de población debajo del cual los caminos pueden ser eliminados.
Probabilidad de eliminación	n$\in \mathbb{R}, n \in$[0,1]	Probabilidad de que un camino por debajo del umbral de eliminación sea eliminado.
Umbral de contracción	n$\in \mathbb{R}, n \in$[0,1]	Valor de densidad de población debajo del cual los caminos pueden contraerse.
Probabilidad de contracción	n$\in \mathbb{R}, n \in$[0,1]	Probabilidad de que un camino por debajo del umbral de contracción se contraiga.
Material de cruce	Material	Material que se aplicará a los cruces generados por el módulo de caminos.
Material de segmento	Material	Material que se aplicará a los segmentos que unen cruces generados por el módulo de caminos.

Algunos problemas surgen de este diseño: ¿Qué hacer si un cruce intersecta un camino al generarse? ¿Qué hacer si están muy cerca un par de cruces? Deben tomarse medidas para evitar encimar caminos o crecer la red de manera innecesaria.

Sea c_i el cruce origen desde el que c_{i+1} (nuevo cruce) está siendo agregado. La red de caminos se consulta y actualiza en base a las siguientes condiciones:

1. c_{i+1} intersecta un camino $\overline{c_j c_k}$: Se mueve c_{i+1} al punto de intersección con $\overline{c_j c_k}$; se elimina $\overline{c_j c_k}$; se agregan $\overline{c_{i+1} c_j}$ y $\overline{c_{i+1} c_k}$; se agrega $\overline{c_i c_{i+1}}$. Fig. 3.9 (a).

2. c_{i+1} tiene cerca o intersecta un cruce c_j : Se borra c_{i+1} y se agrega $\overline{c_i c_j}$. Fig. 3.9 (b).

3. c_{i+1} tiene cerca un camino $\overline{c_j c_k}$: Se mueve c_{i+1} al punto más cercano con $\overline{c_j c_k}$, se elimina $\overline{c_j c_k}$; se agregan $\overline{c_{i+1} c_j}$ y $\overline{c_{i+1} c_k}$; se agrega $\overline{c_i c_{i+1}}$. Fig. 3.9 (c).

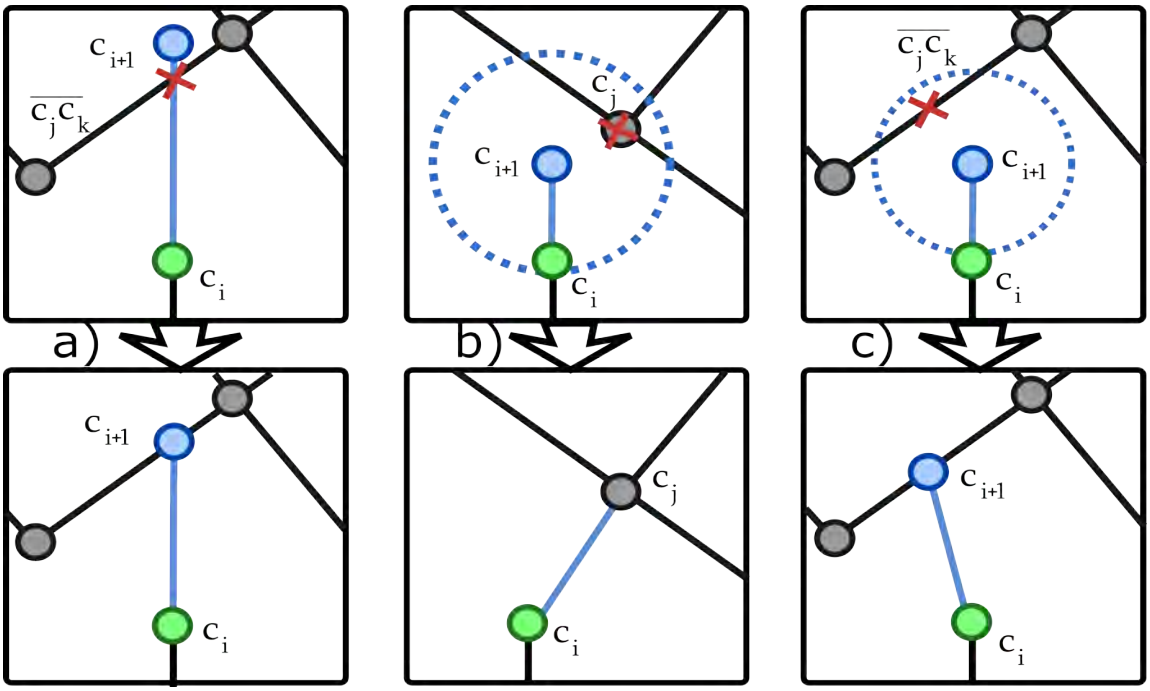


Figura 3.9: Distintos casos de cercanía a la red de caminos.

Parametrización Avanzada

Las reglas anteriores vuelven necesario agregar parámetros que determinen cómo es percibido el ambiente: ¿Cómo definimos que un camino está 'cerca' del otro?.

Al ser parámetros que requieren que el desarrollador conozca cómo funciona el sistema, dificultan el uso del mismo, por lo que conviene más tenerlos como parametrización avanzada disponible sólo para el desarrollador:

Nombre de valor	Tipo de Valor	Utilidad	Valor por defecto
Radio de cruces	n∈ $\mathbb{R}$	Radio en que los nuevos cruces detectan intersecciones con otros cruces.	Promedio(Ancho de camino) * 6
Radio de camino	n∈ $\mathbb{R}$	Radio en que los nuevos cruces detectan intersecciones con otros caminos.	Promedio(Ancho de camino) * 4
Distancia entre puentes	n∈ $\mathbb{R}$	Distancia mínima entre puentes.	Promedio(Ancho de camino) * 15
Grosor entre puentes	n∈ $\mathbb{R}$	Grosor de un camino al determinarse puente.	Promedio(Ancho de camino)

Así, si se detecta un cruce 'cerca' de otro cruce o camino, se unen. Con ésto se espera dar suficiente control sobre los aspectos fundamentales de la creación de caminos.

3.2.3 Módulo de manzanas

Primero se detectarán los conjuntos de cruces capaces de alojar manzanas. Sea C = { c_1,c_2,...,c_n} un conjunto de cruces, C es válido syss:

1. || C || >= 3
2. C forma un ciclo hamiltoniano sin aristas ni vértices internos. Contraejemplo en Fig. 3.10(e),(f).
3. C es convexo. Contraejemplo en Fig. 3.10(d).
4. $\forall$ c_i,c_j $\in$ C, si i$\neq$j $\rightarrow$ $c_i \neq c_j$

Una vez validados los cruces, se usan los puntos de sus mallas de modelo para formar un polígono que se usa para calcular los segmentos de banqueta y el patio interno de la manzana.

Deben darse parámetros para controlar la creación de manzanas, así como los materiales de banquetas y objetos encima de ellas, como postes y árboles:

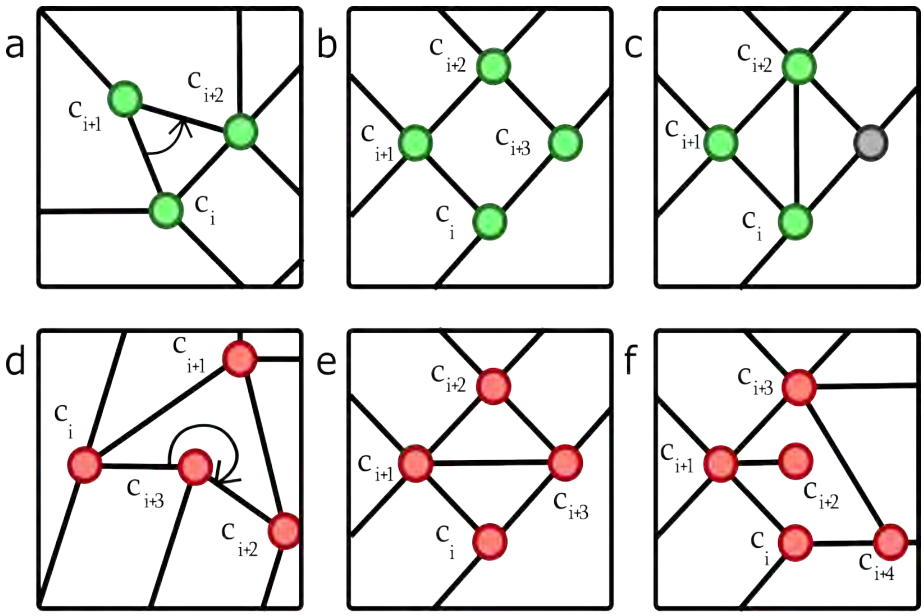


Figura 3.10: Arriba: Conjuntos de cruces válidos. Abajo: Conjunto de cruces inválidos

Nombre de valor	Tipo de Valor	Utilidad
Ancho de Banqueta	n∈ ℝ	Ancho máximo que pueden tener las banqueta.
Separación entre postes	n∈ ℝ	Separación entre postes eléctricos, luces y semáforos colocados en banquetas.
Separación entre árboles	n∈ ℝ	Separación entre árboles colocados en banquetas.
Ancho máximo de banqueta	n∈ ℚ	Ancho máximo que pueden tener las banquetas.
Altura de banqueta	n∈ ℚ	Altura de las banquetas por encima de la ciudad.
Manzanas por explorar	n∈ ℚ	Número de manzanas que generará el algoritmo.
Material de banqueta	Material	Material que tendrán las banquetas.
Material de patio interno	Material	Material que tendrán los patios internos.
Material de patio secundario	Material	Material que tendrán los patios secundarios.
Material de conexiones	Material	Material que tendrán los cables que conectan postes y semáforos.

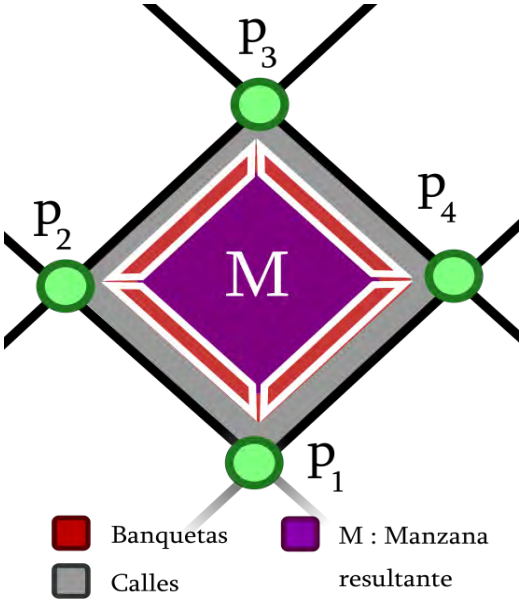


Figura 3.11: Componentes de la manzana: Verde: Cruces que forman la manzana; Rojo: Segmentos de banqueta; Morado: Patio interno.

El patio interno da una base para colocar los edificios, independientemente de si éstos agregan su propio patio.

Parametrización avanzada

Aunque la parametrización es suficiente para que utilizar el algoritmo con normalidad, hay algunos aspectos extra que deben ser considerados.

El primero es que si una manzana tiene poca área o es muy estrecha, resultará extraño colocarle edificios, ya que estos tenderán también a ser pequeños o estrechos, rompiendo con la corrección del ambiente. Debe entonces omitirse la creación de estas manzanas y dar un umbral que permita controlar cuándo pasa esto (Fig. 3.12).

También se debe determinar cómo dividir la manzana en parcelas. En [15] se describen varios tipos existentes.

El esquema de plan hipodámico está diseñado para que la mayoría de manzanas sean cuadriláteros, por lo que la mayoría de parcelamientos serán los Y, X y H (Fig.

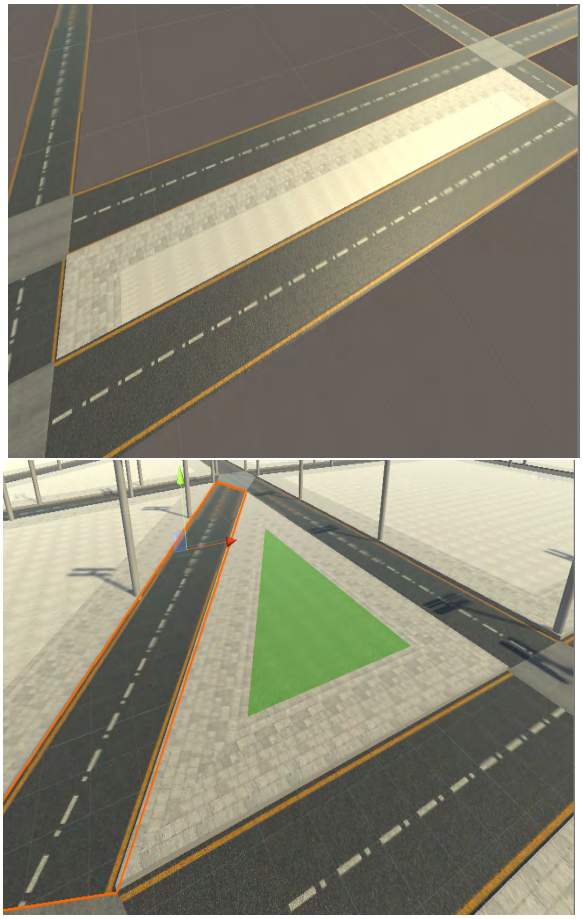


Figura 3.12: Manzanas inválidas debido a la falta de área o grosor.

3.13), y ya que no habrá manzanas descritas por curvas, el parcelamiento curvilíneo será descartado.

Además, existen edificios que ocupan toda la parcela. Se deben dar umbrales para determinar a partir de qué tipo de densidad de población se intenta utilizar qué parcelamiento, y las probabilidades de utilizarlos.

Sea **a** el ancho de banqueta. Considerando los factores anteriores, la siguiente parametrización avanzada es propuesta:

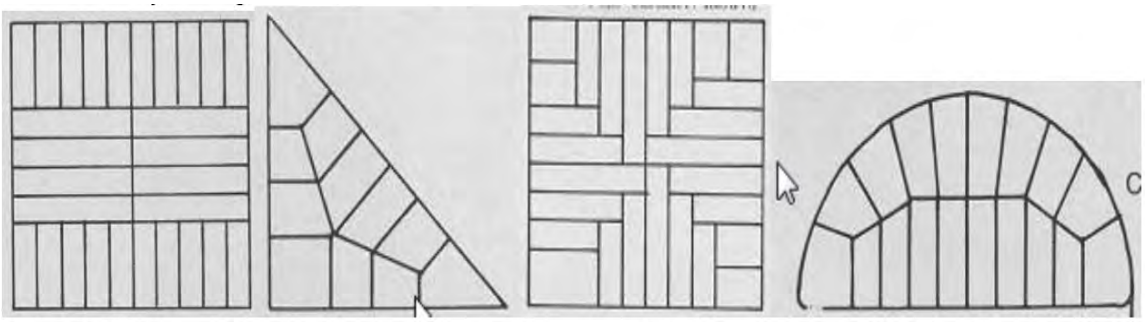


Figura 3.13: De izquierda a derecha: Parcelamientos H, Y, X y curvilíneo.

Nombre de valor	Tipo de Valor	Utilidad	Valor por defecto
Mínima área de manzana	Número real	Mínima área que debe tener una manzana para alojar parcelas.	a^2
Mínimo ancho de manzana	Número real	Mínimo ancho que debe tener una manzana para alojar parcelas.	$a/2$
Umbral de parc. total	n$\in \mathbb{R}, n \in [0,1]$	Valor de densidad de población encima del cual puede aparecer un parcelamiento total.	0.6
Umbral de parc. X	n$\in \mathbb{R}, n \in [0,1]$	Valor de densidad de población encima del cual puede aparecer un parcelamiento X.	0.2
Probabilidad de Parc. total	n$\in \mathbb{R}, n \in [0,1]$	Probabilidad de que una manzana por encima del umbral de parc. total cree un parcelamiento total.	0.2
Probabilidad de Parc. X	n$\in \mathbb{R}, n \in [0,1]$	Probabilidad de que una manzana por debajo del umbral de parc. X cree un parcelamiento X.	0.5

Los parcelamientos H pueden, además, generar un patio interno dentro de ellos que se puede usar con fines recreativos.

3.2.4 Módulo de parcelas

El conjunto de manzanas es proporcionado como entrada al módulo de parcelas, que divide el terreno de la manzana en parcelas sobre las que se edifican las estructuras.

Una vez más, se necesitan parámetros para personalizar las dimensiones de las parcelas. Se simulará también una interacción entre ellas: Podrán unirse, separarse o eliminarse para generar edificios más grandes o permitir el paso mediante callejones:

Nombre de valor	Tipo de Valor	Utilidad
Separación entre parcelas	n$\in \mathbb{R}$	Separación que habrá entre dos parcelas consecutivas.
Ancho	n$\in \mathbb{R}$	Ancho máximo que puede tener una parcela.
Altura	n$\in \mathbb{R}$	Altura máxima que puede tener una parcela.
Probabilidad de unión	n$\in \mathbb{R}$, n $\in$ [0,1]	Probabilidad de una parcela de unirse a una parcela vecina.

Parametrización avanzada

Algunos parámetros extra son necesarios para controlar algunos comportamientos especiales. El algoritmo tendrá la capacidad de crear 'megaedificios' como rascacielos o centros comerciales, por lo que se necesita un parámetro para definir qué tan grandes pueden llegar a ser.

Se debe determinar además qué probabilidad tiene un edificio de aparecer según el distrito en que esté la parcela, además de determinar la influencia del mapa de ruido en la separación entre parcelas. El usuario puede determinar su influencia también.

Nombre de valor	Tipo de Valor	Utilidad	Valor por defecto
Factor de ruido	n$\in \mathbb{R}$	Influencia de la función aleatoria en la separación entre parcelas. Aumentarlo separará más las parcelas.	0.2
Probabilidades de parcela	(p_1, p_2, p_3, p_4, p_5, p_6), $p_i \in \mathbb{R}$	Vector de probabilidades que contiene la probabilidad de cada tipo de parcela de ser elegido, divididos por tipo de distrito.	F*
Multiplicadores de altura	v $\in \mathbb{R}^6$	Vector que indica por cuanto se multiplica la altura en caso de que la parcela sea destinada a alojar un megaedificio.	H*

Tanto el valor de F* como de H* están estrechamente ligados al tipo de manzana en la que se está colocando la parcela, y tanto sus valores por defecto, como la respectiva justificación detrás de ellos, pueden verse en el capítulo de implementación.

3.2.5 Módulo de edificios

La colocación de edificios en la ciudad requiere de comprender un poco el proceso de diseño de los mismos. En [16] se describen principios que puede usar un arquitecto al diseñar un edificio para un cliente. De ellos, tomaremos los de mayor importancia en el apartado visual:

1. El edificio debe construirse para servir al propósito especificado por el cliente.
2. El diseño del edificio debe ser construible por técnicas conocidas con el equipo y labor disponible.
3. El edificio debe proveer el grado de protección requerido por el cliente.
4. El edificio debe ser visualmente atractivo por fuera y por dentro.
5. No existen elementos que supongan un riesgo a la salud o seguridad de sus ocupantes.

En este caso, el cliente somos nosotros, y cumplir cada punto arriba mencionado ayudará a que el ambiente sea más correcto e impredecible. Se usará el diseño de edificios tradicional, el cual divide el volúmen de la parcela en pequeños volúmenes, cada uno con una función específica destinada a satisfacer la necesidad del edificio.

Puesto que se busca únicamente emular el aspecto visual de la ciudad y no su distribución interna, los espacios de un edificio se diseñarán en base a dos necesidades:

1. Acceso al edificio : Determinado de acuerdo al tipo de edificio generado.

2. Función del edificio : Espacio en que se lleva a cabo la actividad o función del edificio.

Dos ejemplos de como se modela un edificio así serían los siguientes:

1. Centro comercial: El acceso requiere dos espacios: Un estacionamiento y un volúmen central; Para sus funciones, se necesitarían áreas amplias de altura media que estén conectadas al volúmen central.

2. Departamento: El acceso se puede modelar mediante un edificio alto central con una puerta en la que estaría colocado un elevador. La función de este edificio (alojar residentes) puede emularse colocando una o dos alas que salgan de este acceso y que tendrían muros y ventanas.

Una gramática independiente del contexto puede ser una buena aproximación a la construcción procedural de edificios. Iniciando con el volúmen de la parcela y, mediante producciones, colocando formas 3D y cambiando el volúmen para extender el edificio se puede lograr algo aceptable, similar a lo visto en [6].

La parametrización extensiva de éste módulo sería enorme, podrían incluso haber parámetros para controlar cuántos vértices y aristas puede tener cada malla de modelo, pero eso vuelve difícil de usar el algoritmo. Por ello, sólo se proporcionarán parámetros que puedan modificar el aspecto fundamental de los edificios: Materiales para el suelo, tejado y paredes.

Nombre de valor	Tipo de Valor	Utilidad
Ancho de pared	n$\in \mathbb{R}$	Ancho estándar de las paredes de los edificios.
Altura de piso	n$\in \mathbb{R}$	Altura a tener por cada piso que conforme el edificio.
Grosor de pilar	n$\in \mathbb{R}$, n$\in$[0,1]	Grosor estándar relativo al ancho de pared que tendrán los pilares.
Altura de escalón	n$\in \mathbb{R}$, n$\in$[0,1]	Altura relativa a la altura de piso que tendrán los escalones en las escaleras generadas.

La descripción de la gramática no forma parte de los parámetros pues forma parte integral del sistema, por lo que no existirá parametrización avanzada. Esta se verá en el capítulo de Implementación.

3.2.6 Generación del terreno

El módulo de terreno crea la malla 3D del terreno. Éste representa la capa de mayor profundidad que verá el usuario.

Debido a que el terreno puede tardar una cantidad de tiempo considerable en crearse, conviene segmentarlo en pequeñas regiones. Por ejemplo, si el mapa de terreno usado es de 1200x1200 pixeles, se puede dividir en pequeños mosaicos de 4x4, reduciendo la cantidad de polígonos en la malla. Aunque reduce el realismo del terreno, que este sea en su mayoría llano reduce el impacto de ésta segmentación.

Debe tener opciones para parametrizar los materiales del terreno y el agua, así como definir su profundidad y altura del nivel del mar. Por último, se da una separación de terreno, que alzará la malla del terreno para que los demás componentes de la ciudad no la intersecten.

Nombre de valor	Tipo de Valor	Utilidad
Texturas	Arreglo de texturas	Texturas a aplicar sobre el terreno
Separación de terreno	Número real	Altura que se reducirá al terreno, para no sobreponerse a la ciudad.
Material de Agua	Material	Contiene el material que se aplicará sobre la malla del agua
Altura del fondo	n$\in \mathbb{R}$	Altura del punto más bajo del terreno.
Nivel del mar	n$\in \mathbb{R}, n \in$[0,1]	Altura relativa al fondo sobre la que se coloca la malla del agua.
Árboles	Arreglo de objetos 3D	Mallas 3D de árboles que se colocarán sobre banquetas y áreas verdes.
Magnitud de modificación	n$\in \mathbb{Z}, n \mid h, n \mid w$	Valor de ancho y alto de la región simbólica para modificar el terreno.

3.3 Evaluación de la calidad del ambiente

A continuación se discutirán los elementos que deben considerar las funciones que califican corrección, impredecibilidad y replicabilidad. Sólo se introducirán los conceptos a tomar en cuenta, mientras que el capítulo de implementación será el que de forma concreta a lo aquí expuesto.

3.3.1 Evaluando corrección

La corrección del ambiente se evalúa según lo expuesto en el capítulo 2. Su valor se verá reflejado en un número real, el cual se reduce cada que se detecta un componente que:

1. Intersecta a otro componente (i.e. un edificio incrustado en otro).
2. Está suspendido en el aire.
3. Su posición / rotación desafía las leyes de la física.

4. Está en una capa que no le corresponde (i.e. una banqueta en un tejado o un poste en medio de una carretera).

De estos, se puede hacer que la implementación del algoritmo garantice 2. y 3., mientras que 1. y 4. deben ser comprobados manualmente o por una función.

Los componentes aún pueden afectar el realismo percibido de la ciudad si tienen características extremas o extrañas, por lo que también se restará valor a la corrección si:

1. Un edificio se halla en un distrito en que no tiene sentido encontrarlo (i.e. un rascacielos en un distrito industrial).

2. Hay demasiados edificios con alturas extremas (Muy altas o bajas), lo cual en la vida real no es factible económicamente.

3. Las dimensiones de los componentes crean objetos imposibles de encontrar en el mundo real (i.e. un rascacielos de $1m^2$).

4. La forma del edificio no tiene un equivalente en la vida real o desafía las leyes de la física (i.e. un edificio en forma de tornillo).

El valor de la corrección tendrá un valor del 0 al 100 y debe ser calculado en base al impacto que tiene el contenido generado por cada módulo en el ambiente, dando como resultado la siguiente fórmula:

Corrección del ambiente = C(Caminos)*0.1 + D(Manzanas)*0.1 + E(Parcelas)*0.2 + F(Edificios)*0.6.

Donde C : Red de caminos $\to \mathbb{R}$, D : Conjunto de manzanas $\to \mathbb{R}$, E : Conjunto de parcelas $\to \mathbb{R}$, F : Conjunto de edificios $\to \mathbb{R}$ son funciones que evalúan la corrección del contenido generado en cada capa de la ciudad.

3.3.2 Evaluando replicabilidad

El algoritmo será replicable syss a cada combinación de semilla, entrada de algoritmo y parametrización corresponde un único ambiente (La 'función' que genera la ciudad es inyectiva).

Sea e el conjunto de parámetros usado para la generación. Sea f una función de ruido escalonada cuyo estado inicial s_1 = f(s_0,e) es determinado por la semilla s_0. Cada vez que se solicite el i-ésimo número aleatorio s_i, se calcula f(s_{i-1},e)=s_i. Ésta función deberá generar únicamente números aleatorios enteros o flotantes.

Sea s_j el último número aleatorio generado y c_i el i-ésimo componente siendo generado. Un nuevo número aleatorio s_{j+1} modifica a c_i según los siguientes casos:

1. c_i es un camino: s_{j+1} determina si c_i es eliminado o su grosor es reducido.
2. c_i es una manzana: s_{j+1} determina el tipo de manzana de c_i.
3. c_i es una parcela: En la parcela, s_{j+1} determina el tipo de edificio a crear y s_{j+2} determina la altura del edificio contenido.
4. c_i es un edificio: s_{j+1} determina el conjunto de materiales a utilizar. [s_{j+2},s_{j+3},....,s_{j+k}] son utilizados por el generador de edificios para construirlo proceduralmente, ya que pueden necesitarse k números aleatorios al generar paredes, techos, ventanas, etc...

Cada uno de estos casos es parte de la función escalón f:

$$\text{f}(s_j,\text{e}) \begin{cases} g_1(s_j,e) & \text{Caso 1.} \\ g_2(s_j,e) & \text{Caso 2.} \\ \ldots & \ldots \\ g_n(s_j,e) & \text{Caso n.} \end{cases} \tag{3.1}$$

Para el caso 1, por ejemplo, s_{j+1} = f(s_j,e) = g_1(s_j,e) = Random(0,p_m) si estamos generando un camino.

Donde p_m es un parámetro de e. En el capítulo de desarrollo se definen qué operaciones (g_i) son usadas.

3.3.3 Evaluando predictibilidad

La predictibilidad del ambiente se reduce a más variados sean los componentes del ambiente en distintas capas:

Terreno:

El usuario tiene control directo sobre cómo se genera el terreno, por lo que al no depender del algoritmo, no se incluirá en la evaluación de impredecibildiad.

Caminos:

No se evaluará ya que el modelo de crecimiento de los caminos (Plan hipodámico) es altamente predecible pero se considera 'normal'.

Manzanas:

Si bien es casi imposible que dos manzanas sean iguales, sí pueden ser similares en cuanto al contenido generado dentro de ellas. Las características de éste son determinadas por el tipo de manzana, y mientras más manzanas del mismo tipo haya juntas, más predecible se considerará el ambiente pues se creerá que la siguiente manzana también será del mismo tipo.

Se podría considerar que mientras esté equilibrada la asignación de tipos (Por ejemplo, que los tipos sigan una distribución gaussiana), el ambiente será impredecible, pero esto no basta.

Si en una ciudad todos los parques (Manzanas tipo parque) están amontonados en un lugar lejano, el resto del ambiente será predecible pues no hay nada cerca con qué compararlo. Un cambio pequeño a intervalos regulares es mejor percibido que un gran cambio a intervalos amplios.

Cada edificio tiene entonces un valor que indica qué tantas manzanas de tipo distinto tiene 'cerca', donde cerca es determinado por un nivel de influencia de cada tipo.

Si el promedio de este valor es muy alto para las manzanas más comúnes, querrá decir que hay demasiada variedad en los tipos de manzana y el ambiente puede ser caótico. Si es demasiado bajo, el ambiente tiene poca variedad y el ambiente será predecible.

Parcelas:

Como se mencionó antes, las parcelas son conceptos meramente abstractos, y no

tienen una malla de modelo, por lo que su predictibilidad está determinada por el edificio erigido en ellas.

Edificios:

Este componente es el que más efecto tiene sobre la predictibilidad del ambiente, y evaluarlo debe comparar cuánto se parecen los edificios entre sí.

El índice de parentezco es un número en el intervalo [0,1], y representa qué tanto se parecen dos edificios entre sí dependiendo de los materiales, producción y dimensiones de los edificios.

Buscamos que nuestro ambiente sea impredecible pero no caótico, por lo que valores muy impredecibles (0) y muy predecibles (1) pueden afectar negativamente el ambiente y se buscará evitarlos al implementar el algoritmo.

3.4 Herramientas de desarrollo

El desarrollo del proyecto requiere del uso de herramientas para la visualización del ambiente y la creación de texturas. A continuación se enumeran las que se usarán.

3.4.1 Motor gráfico

Un motor gráfico es una compilación de funciones y API's (Interfaz de programación de aplicaciones) que aceleran y facilitan el desarrollo al no tener que programar desde cero todo lo que un proyecto necesita. Aunque usarlos es una ventaja, el programa resultante muchas veces queda ligado al motor, por lo que llevarlo a otros motores o agregar ciertas funcionalidades puede ser complicado o imposible.

Debido a que la generación procedural está diseñada para ahorrar costos y tiempos de desarrollo, resulta natural el uso de un motor gráfico, y se pensará en las siguientes cualidades para elegirlo:

- Multi plataforma: El motor debe ser capaz de llevar el programa resultante a múltiples sistemas operativos y permitir el desarrollo del proyecto en estos.

- Costo: El proyecto se realizará para dar a desarrolladores independientes herramientas para la creación de sus espacios virtuales, por lo que su costo debe ser lo más bajo posible.
- Bien documentado: No debe ser difícil encontrar información sobre las API's y funciones que el motor utiliza.

El motor de juegos Unity[1] permite la construcción de software multimedia para distintas plataformas, además de brindar suficientes herramientas para el desarrollo en su versión gratuita y no tener dificultades mayores con su documentación.

La similitud que tiene su API con los tipos encontrados en bibliotecas de graficación como OpenGL hace amigable el código incluso para un desarrollador sin experiencia. Al ser un motor de juegos, también es sencillo extender el funcionamiento del programa de ser necesario.

3.4.2 Contenido gráfico

Éste proyecto está enfocado en la generación de geometría 3D y valores de funciones, y no en la producción de imágenes o texturas, por lo que, aunque existan formas de generarlos proceduralmente, no se hará, pues va más allá de los alcances del proyecto.

Se descargarán texturas de sitios de imágenes o se crearán manualmente, luego se utilizará GIMP[2] para modificarlas y se introducirán al programa.

Un ejemplo de texturas a utilizar serían las siguientes:

[1] Motor de juegos que permite una fácil integración de contenido multimedia a proyectos de cómputo.

[2] Herramienta gratuita de manipulación de imágenes.

Figura 3.14: Ejemplo de texturas usadas en el algoritmo (1).

Figura 3.15: Ejemplo de texturas usadas en el algoritmo (2).

Capítulo 4

Implementación del algoritmo

En el presente capítulo se entrará en detalles sobre la implementación del algoritmo, que consiste en definir las ténicas que se usarán en cada módulo del programa, así como determinar el uso de la parametrización en la generación del contenido.

Se dará también la salida de cada módulo, la entrada que utilizan y el aporte que realizan a las funciones de evaluación del ambiente.

Tómese C como la ciudad siendo generada actualmente. Sea Random(C) = **Seq**, la secuencia de operaciones aleatorias usada para generar cada componente de C.

4.1 Creación de la red de caminos

Como se vió en el diseño del algoritmo, se utilizará un sistema L para simular el crecimiento de la red de caminos de la ciudad. A medida que vaya creciendo la red, se irá generando una gráfica, en la que los vértices serán los cruces y los segmentos de calle que los unen serán las aristas.

A continuación se define el sistema L a usar, se describe cómo percibe el ambiente y cómo influye esta percepción en las producciones tomadas por el sistema.

4.1.1 Gramática

La gramática a utilizar estará basada en un sistema 2L parametrizado, el cual extiende la funcionalidad de los sistemas L dependientes del contexto, permitiendo que

sus producciones, tanto del lado izquierdo como del derecho, tengan parámetros.

No es un sistema L debido a que no aplica en paralelo las producciones, sino que utiliza un autómata de pila que las aplica secuencialmente.

S = {{R(a,b),B(a,b,c),?I(a,b)},{+(a,b)},Σ,I,σ}. a ∈ $\mathbb{Z}$,b ∈ Σ.
Donde las reglas del sistema σ, basadas en [6], son las siguientes:

- I : R(caminoInicial) · ?I(camino,estado) : estado==SIN_ASIGNAR

- p2 : R(regla) · ?I(camino,estado) : estado==EXITOSO
  → +(camino.angulo) F(camino.longitud)
  B(pRegla[1],pCamino[1])
  B(pRegla[2],pCamino[2])
  R(pRegla[0]) ?I(pCamino[0],SIN_ASIGNAR)

- p3 : R(regla) · ?I(camino,estado) : estado==DETENER→+(camino.angulo) F(camino.longitud)

- p4 : R(regla) · ?I(camino,estado) : estado==FALLIDO → ϵ

- p5 : B(regla,camino) → [R(regla)?I(camino,SIN_ASIGNAR)]

- p6 : ?I(camino,estado) : estado==SIN_ASIGNAR → ?I(camino,estado)

- p7: ?I(camino,estado) : estado!=SIN_ASIGNAR→ ϵ

La producción p6 asigna valor al estado.

Los parámetros (Σ) están formados por pRegla y pCamino, que son asignados por los atributos globales, mientras que los límites locales modifican los valores del camino al aplicar la producción p6. La comprobación de límites locales se hace 2 veces para que el nuevo camino no intersecte otros al actualizarse.

El estado del camino será detener si los límites locales lo unen a un cruce o camino ya existente, lo cual evitará que la red crezca innecesariamente.

4.1.2 Malla de modelo de los caminos

Con la red de caminos se crea la malla de ésta, explorándola en BFS a partir de un nodo inicial elegido al azar. Debido a que la red de caminos es conexa (puesto que sólo se crean nuevos cruces y segmentos a partir de los existentes en la red), el elegir un nodo al azar no deja caminos sin explorar.

Sea V(n) = { n_1, n_2,..., n_j} los vecinos del nodo n ordenados por ángulo. Para todo par de vecinos de n: n_i, $n_{(i+1)\%j}$ se agregan los puntos de la malla de modelo del cruce 'ensanchando' las aristas que unen $\overline{nn_i}$ y $\overline{nn_{i+1}}$, y calculando intersecciones o esquinas en base al ángulo que forman:

1. $\measuredangle\, n_i n n_{i+1}$ forma un ángulo de menos de 180 : Se agrega la intersección entre las aristas. (Fig. 4.1) (a).

2. $\measuredangle\, n_i n n_{i+1}$ forma un ángulo de más de 180: Se agregan la esquina derecha más cercana de $\overline{nn_i}$ y la esquina izquierda más cercana de $\overline{nn_{i+1}}$. (Fig. 4.1) (b).

3. $\measuredangle\, n_i n n_{i+1}$ forma un ángulo de 180: Se agrega la esquina izquierda más cercana del rectángulo nn_{i+1}. (Fig. 4.1) (c).

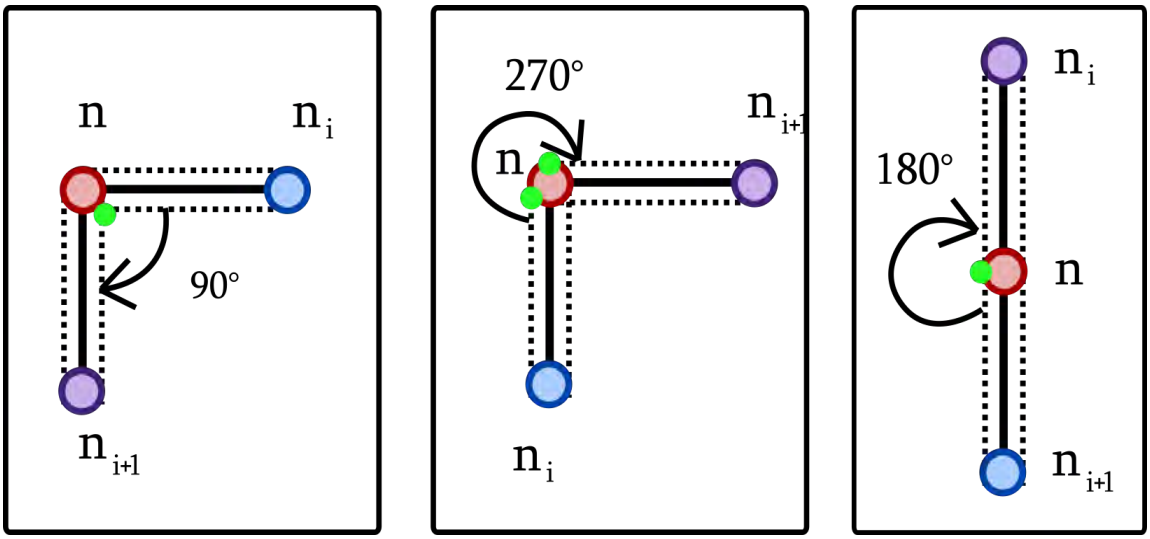


Figura 4.1: Distintos ángulos dando distintos puntos característicos (En verde).

Éstos puntos característicos son agregados a una lista que asocia ambos nodos, de modo que cuando se explore el nodo vecino, se completará el segmento de camino con los puntos del otro cruce. Como todo vecino se explora dos veces (una siendo el nodo principal n_i y la otra siendo n_{i+1}) por nodo, y cada nodo se visita al menos una vez, está garantizado que cada segmento de calle tendrá sus 4 puntos característicos.

El cálculo de los puntos uv de los cruces es una proyección:

p(x,y,z) = (x,z)

Pero el cálculo de los puntos uv de los segmentos de calle requiere alinear el segmento con la textura del camino (Fig. 4.2).

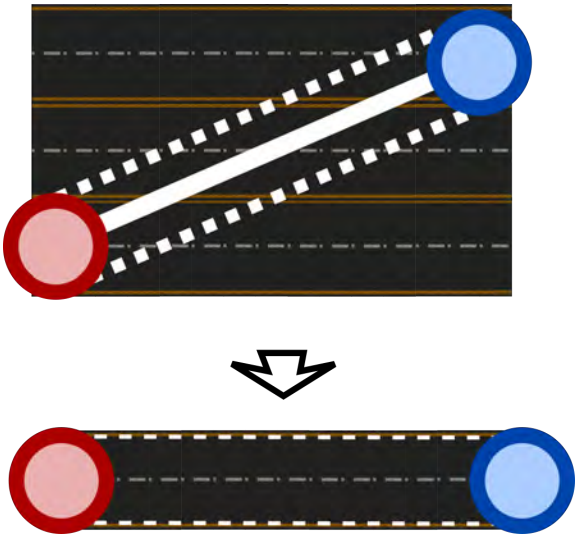


Figura 4.2: Alineamiento de un camino de inclinación diferente para que embone en la textura.

4.1.3 Calificando el módulo de caminos

Corrección:

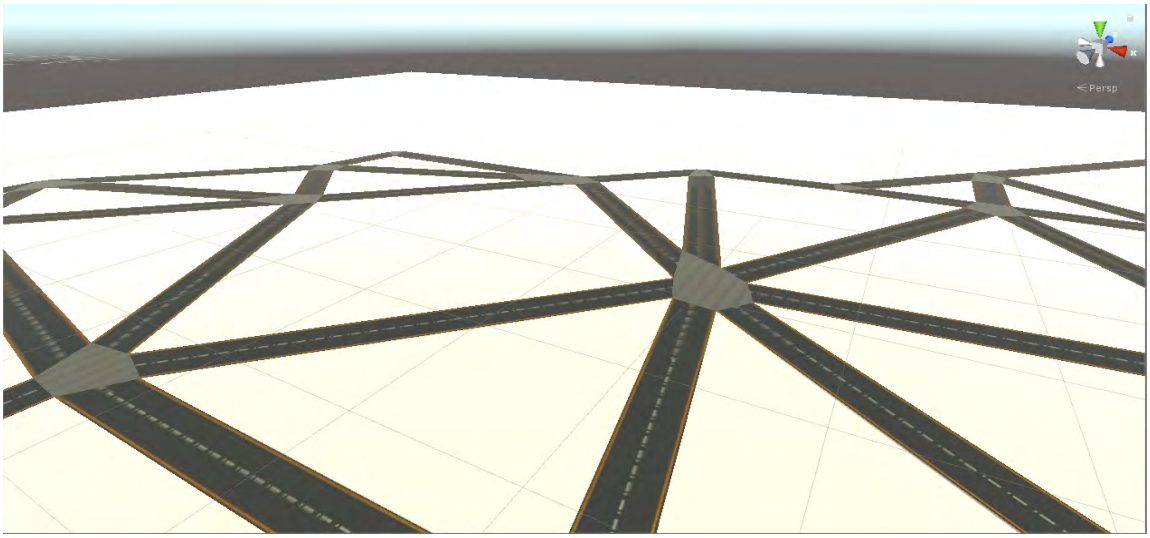


Figura 4.3: Ejemplo de una parte de la malla generada por el módulo de caminos.

Por construcción, el módulo de caminos no afecta la corrección del ambiente. He aquí el por qué:

- Un camino no puede intersectar a otro debido a la doble comprobación de límites locales que se realiza en cada cruce.
- Los caminos se colocan encima del terreno, y se le puede parametrizar la altura a este, por lo que para romper la corrección habría que hacerlo a propósito.
- Los caminos sólo se colocan en terreno llano y todos son rectos, además de que puede parametrizarse su ancho, por lo que no afectarán la corrección del ambiente a menos que el desarrollador lo haga a propósito.

El desarrollador es el que coloca los parámetros necesarios para que el ambiente no afecte la corrección, por lo que la asignación por defecto de la parametrización avanzada (La cual es definida por el desarrollador) debería ser suficiente para preservar esta corectitud.

Se pondrá a prueba esta teoría haciendo que la función de corrección devuelva siempre 100 para el módulo de caminos.

Replicabilidad:

Cada que se crea un camino, el algoritmo ejecuta (y guarda) las siguiente operaciones aleatorias en Seq:

- g_1 = Bernoulli(Umbral de eliminación) : Determina si un camino es eliminado.
- g_2 = Bernoulli(Umbral de contracción) : Determina si un camino reduce su grosor.

Predictibilidad:

El plan hipodámico, esquema utilizado para el crecimiento de los caminos, no afecta la impredecibilidad del ambiente pues aunque sea muy repetitivo el patrón con el que los caminos crecen, este es considerado normal.

Incluso si se considera repetitivo, este efecto se equilibra por la eliminación y contracción aleatorio de los caminos. Las probabilidades de que esto suceda son bajas por lo que se evita que se perciban como caóticos.

Ya que se pone a prueba la concepción que tiene el autor del entorno, se fijará el valor de impredecibilidad en 50.

4.1.4 Salida

La salida del módulo de caminos será la siguiente:

1. Gráfica conteniendo los cruces como nodos y los segmentos de calle como aristas.
2. Malla de modelo 3D de los cruces de la red creada a partir de la gráfica.
3. Malla de modelo 3D de las calles que unen los cruces creada a partir de la gráfica.

4.2 Cálculo de manzanas

La red de caminos es entonces introducida como entrada al módulo de manzanas, que determina los cruces que pueden formar una manzana, y genera la malla de modelo 3D de banquetas, postes y semáforos.

4.2.1 Cálculo de las manzanas

El cálculo de manzanas se hará recorriendo la red en BFS, y cada que se visite un nodo se calcularán todas las manzanas de las que el nodo forma parte. Sea v_i el i-ésimo nodo explorado. Sea $V(v_i) = \{ n_0, n_1, ..., n_j\}$ la vecindad del nodo v_i. Las manzanas que contienen a v_i son calculadas con el siguiente algoritmo:

1. Sea M = [] la lista de nuevas manzanas.
2. Sea a = v_i.
3. Sea b = n_j.
4. Sea c el vecino de b que minimiza el ángulo $\measuredangle$ abc, c $\neq$ a
5. $\forall$ b $\in V(v_i)$:
   (a) Sea A = [a, b] la lista de de cruces que componen la nueva manzana.
   (b) Mientras c $\neq$ a:
      i. Se inserta c en A.
      ii. Sea actualiza a = b, b = c.
      iii. Sea c el vecino de b que minimiza el ángulo $\measuredangle$ abc.
      iv. Si a = c, se aborta la creación de la manzana.
      v. Si $\measuredangle$ abc $>$ 180 ó | M | $>$ 15, la creación de la manzana se aborta.
   (c) Se agrega A a M.

En otras palabras, las manzanas se forman intentando crear un ciclo convexo desde el vértice inicial. Las condiciones iii) y iv) evitan que manzanas no convexas o con vértices internos sean creadas. La opción natural aquí sería triangular el espacio no convexo generando nuevos caminos, pero debido a los alcances del proyecto, simplemente será descartada.

Una vez determinados los cruces que conforman la manzana, se calcula un subpolígono con el ancho de la banqueta y se unen uno a uno los puntos de la manzana con los del subpolígono para formar las banquetas (Fig. 4.4).

Una vez terminado, se colocan aleatoriamente postes de luz, árboles, luces y semáforos a lo largo de las banquetas.

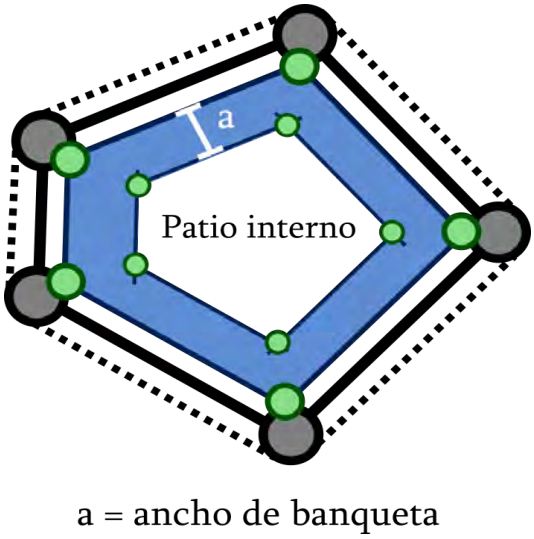


Figura 4.4: Ejemplo de creación del subpolígono.

4.2.2 Determinación de tipos

Las manzanas se componen de dos tipos: El primario y el secundario. Su efecto y las condiciones con que son elegidos se muestran a continuación:

Tipo primario

El tipo primario define el tipo de parcelamiento que será utilizado en la manzana. Los existentes y sus condiciones para ser elegidos son:

- INVÁLIDO: Área menor al área mínima ó grosor menor al grosor mínimo de manzana.
- TOTAL:
  - Densidad de población mayor a 0.6
  - Área de la manzana mayor a (Ancho promedio de parcela*1.5)^2.
  - Ancho mínimo de la manzana mayor a (Ancho promedio de parcela*0.8)
  - Elegido con 20% de probabilidad.
- COMPLETO:

  - Densidad de población mayor a 0.2
  - Elegido con 50% de probabilidad.

- INCOMPLETO: Elegido si los demás criterios no cazan.

Una alta densidad de población está entonces asociada a parcelamientos con menos espacios vacíos. i.e. el parcelamiento total no perderá espacio al usar toda la manzana para generar un edificio, entre otros.

Manzanas de áreas verdes

Para dar variedad al sistema, cada manzana tendrá un 2% de probabilidad de 'mutar' en una manzana de tipo área verde si cae en un distrito residencial, comercial o industrial.

4.2.3 Calificando el módulo de manzanas

Corrección:

Hay algunos aspectos de la corrección que el módulo de manzanas no puede romper por defecto:

- Una manzana no puede encimarse con otra ya que su construcción va buscando siempre el ángulo más pequeño, lo cual provoca que se detecten sólo ciclos sin aristas internas. Incluso si hubiera vértices internos, el paso 5.2.4 del algoritmo aborta la generación si los detecta.
- La escala de las banquetas es parametrizable, por lo que para que rompa la corrección, el desarrollador o usuario tendrá que poner un valor irreal.
- Los objetos propios de las banquetas sólo se generan a lo largo de estas, lo que vuelve imposible encontrarlos en otra capa.
- Las restricciones impuestas a las manzanas (ciclos hamiltonianos sin aristas ni vértices internos), vuelven imposible un fallo de forma en la corrección.

Sin embargo, en casos extremos se puede generar un edificio en un distrito que no le corresponde. Esto no debería reducir siempre la corrección, pues deben variarse el

tipo de edificios generados para no aburrir al usuario, de esto se ocupará el módulo de parcelas, ya que en éste se decide el tipo de edificios que se generan.

Replicabilidad:

Se hace la siguiente operación cada que se determina el tipo de una manzana:

- g_3 = Random(0,1) : Se genera un número aleatorio de 0 a 1, y se usa para determinar el tipo de parcelamiento usado en la manzana.
- g_4 = Bernoulli(0.02) : Determina si la manzana 'muta' en una manzana verde.

Predictibilidad:

Para generar variedad en el ambiente, los distritos pueden generar tipos de manzana distintos, i.e. un distrito residencial puede generar manzanas de tipo comercial o verdes.

Sin embargo, de esta aproximación surgen problemas: De hacerse muy probable generar estas manzanas, no servirá de nada la división de distritos pues habrá edificios de todos tipos en todos lados, y el ambiente generado será caótico.

Si, por otro lado, se vuelven inexistentes estas variaciones, el ambiente generado será monótono y predecible.

Para ello se medirá la distancia promedio que tienen las manzanas a la manzana de tipo distinto más cercana. La predictibilidad de una manzana tomará valores entre 0 y 100, tomando 0 si la distancia es igual al ancho de un camino reducido, y tomando 100 si es mayor o igual a 6 veces el ancho de la manzana.

Esta evaluación tiene en cuenta la distribución de las manzanas en distritos: Las distancias a manzanas diferentes en el centro de un distrito serán mayores que en el borde, y en el medio debería equilibrarse con la generación aleatoria de tipos de manzana diferentes.

4.2.4 Salida

La salida del módulo de manzanas consiste en lo siguiente:

1. Malla 3D de las manzanas.

2. Malla 3D de las banquetas.

3. Malla 3D de postes eléctricos, semáforos, luces y el cableado entre éstos.

4. Lista de manzanas de tipo VERDE.

5. Lista con ubicaciones para árboles.

Las mallas 3D forman parte del contenido gráfico que se muestra al usuario, mientras que la lista de manzanas y ubicaciones de los árboles son proporcionadas al módulo de terreno para que éste coloque árboles.

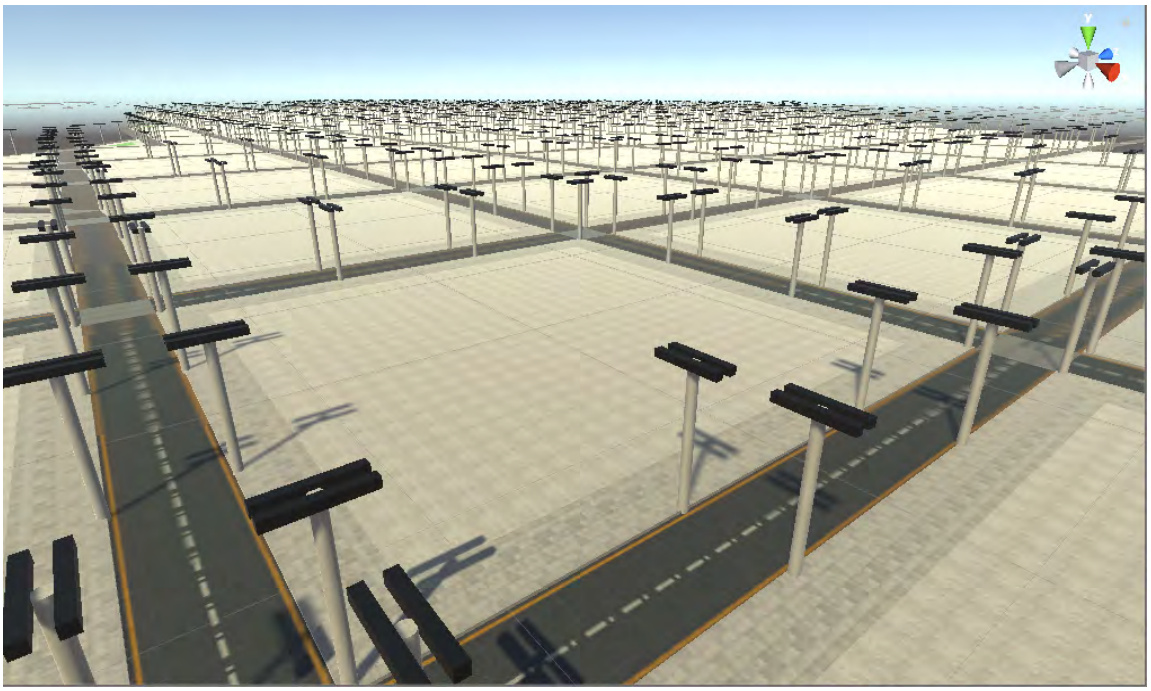


Figura 4.5: Ejemplo de malla 3D de la ciudad con caminos y manzanas colocados.

4.3 Parcelamiento de manzanas

El proceso de división de los patios internos en parcelas y la asignación del tipo de edificio que contendrán se da a continuación.

4.3.1 Determinando el tipo de parcelamiento

El tipo de la manzana determina el parcelamiento a utilizar según las siguientes reglas:

- TOTAL → Se genera una sola parcela que ocupa toda la manzana.
- COMPLETO → Parcelamiento X.
- INCOMPLETO → Parcelamiento H modificado.
- INVÁLIDO → No se divide el terreno de la manzana en parcelas.

De estos, el parcelamiento H modificado (Fig. 4.6) es el único no visto hasta ahora. Este reduce el tamaño de las parcelas al generar un patio interno que puede sevir como área verde o recreativa, característico de regiones con menor densidad de población, en las que la optimización del espacio no es tan importante.

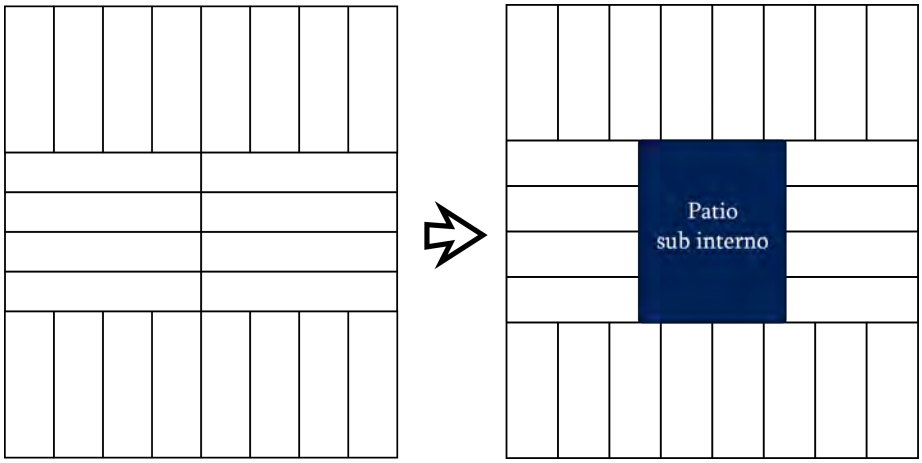


Figura 4.6: Parcelamiento H y parcelamiento H modificado.

4.3.2 Cálculo de dimensiones de la parcela

El polígono que conforma la base de cada parcela se calcula a partir dos conjuntos de puntos: El conjunto de puntos superiores y el conjunto de puntos inferiores, los inferiores son unidos por pares con los superiores uno a uno para formar la base de la parcela. El cómo se determinan éstos conjuntos es decidido por los tipos de parcelamiento.

Puntos inferiores:

Los puntos inferiores son determinados dividiendo cada segmento del patio interno de la manzana, usando el parámetro de ancho de parcela y de separación entre parcelas.

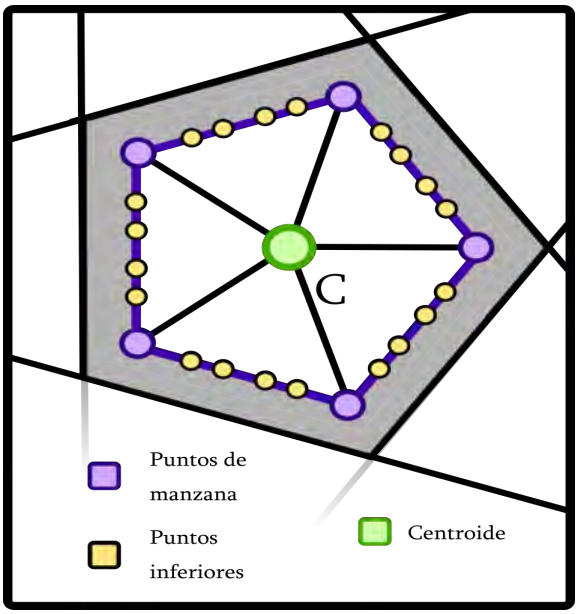


Figura 4.7: Ejemplo de salida para el módulo de manzanas.

Cada par de puntos inferiores se ensancha o estrecha aleatoriamente para tener más variedad en las dimensiones resultantes de las parcelas.

Puntos superiores:

Se generará un punto superior por cada punto inferior, utilizando la posición de este y el tipo de parcelamiento usado en la manzana.

Sea c el centroide de la manzana. Sean l_j, l_{j-1} el j-ésimo y j-1-ésimo lados del polígono que conforma la manzana. Sean p_0, p_1, p_n, p_i, p_{i+1} el punto inicial, segundo, final, i-ésimo e i+1-ésimo de los puntos inferiores generados en l_j. Sea p_{m-1} el penúltimo punto inferior del segmento l_{j-1}.

Sea $\overrightarrow{p_k p_l}$ la dirección entre dos puntos. Sea **b** = **ProfundidadParcela**. Sea $d_1 = \overrightarrow{p_0 p_c}$ y $d_2 = \overrightarrow{p_n p_c}$. Sea $\times$ el producto cruz en $\mathbb{R}^3$. Se determinan los puntos superiores P_i, P_{i+1}, P_{i+2} conforme a los siguientes casos:

1. Esquina (i=0)

Se empieza tomando a $P_i = p_{m-1}$, debido a que una esquina debe tomar puntos del segmento anterior para no dejar espacios vacíos.

Sean L_j y L_{j-1} rectas perpendiculares a l_j y l_{j-1} | L_j pasa por p_{i+1} y L_{j-1} pasa por p_{m-1}. Se prueban los siguientes casos:

1. P_{i+1} es igualado a la intersección entre L_j y L_{j-1}. Este caso funciona para la mayoría de parcelas, puesto que en la mayoría de los casos $| \overrightarrow{p_0 p_{m-1}} | = |\overrightarrow{p_0 p_1}|$. De no ser así, es muy probable que la intersección caiga fuera de la manzana y se prueba el caso siguiente.

2. Si P_{i+1} cae fuera de la manzana, P_{i+1} se iguala a la intersección entre L_j y d_1. Si la intersección no está en d_1, P_{i+1} se iguala a la intersección entre L_{j-1} y d_1.

3. Si P_{i+1} sigue cayendo fuera de la manzana o $| \overrightarrow{p_0 P_{i+1}} | > | \overrightarrow{p_0 c} |$, se intenta lo siguiente: $P_{i+1} = p_1 + \mathbf{b}(\overrightarrow{p_0 p_n} \times (0,1,0))$. $P_{i+2} = p_{m-1} + \mathbf{b}(\overrightarrow{p_{m-1} p_0} \times (0,1,0))$, es decir, se agregan dos puntos a distancia fija de L_j y L_{j-1} y se unen.

4. Si, aún así, alguno de los puntos quedan fuera de la manzana, se aborta la generación de P_{i+1} y P_{i+2}, dejando una base de manzana triangular.

Otro caso que se intentó fue utilizar la intersección de perpendiculares con la recta hacia el centroide de p_{m-1} y p_1 (como en el caso 2), y luego unirlas mediante un nuevo polígono, sin embargo, eso genera intersecciones con otras parcelas, por lo que no fue incluido.

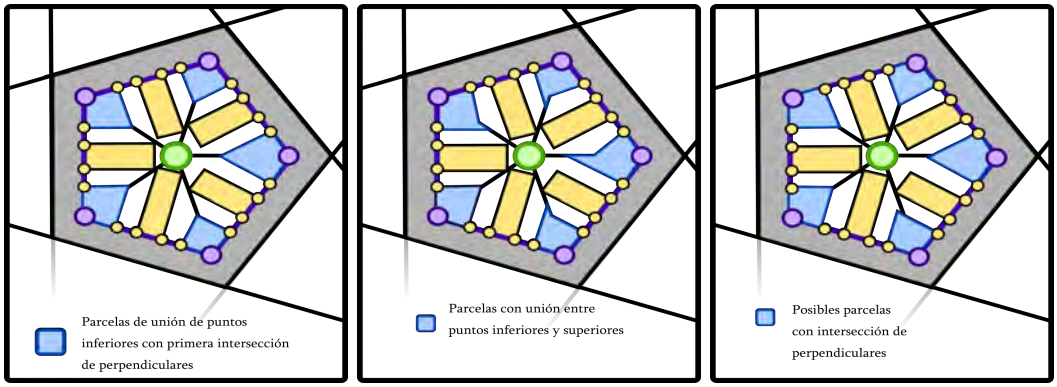


Figura 4.8: En azul: Distintos resultados de casos para las parcelas en las esquinas..

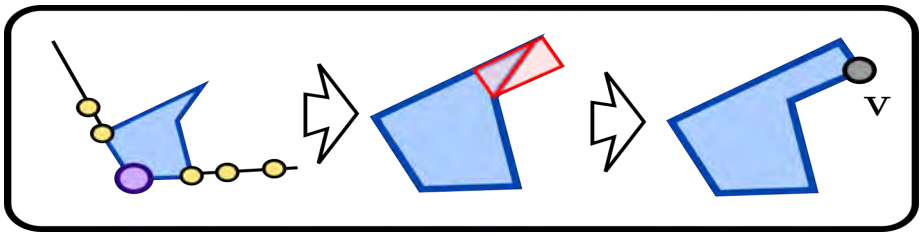


Figura 4.9: Ejemplo de caso no incluido en la lista de casos para parcelas en esquinas.

2. i>2 y se tiene un Parcelamiento H modificado:

$P_i = p_i + \mathbf{b}(\overrightarrow{p_0 p_n} \times (0,1,0))$ y $P_{i+1} = P_i + \overrightarrow{p_i p_{i+1}}$.

3. Parcelamiento X:

Sean L_i y L_{i+1} rectas perpendiculares a l_j | L_i pasa por p_i y L_{i+1} pasa por p_{i+1}. Se prueban los siguientes casos:

1. P_i se iguala a la intersección entre L_i y d_1. Si la intersección no está en el segmento descrito por d_1, P_i se iguala a la intersección entre L_i y d_2. Luego se iguala $P_{i+1} = P_i + \overrightarrow{p_i p_{i+1}}$.

2. Si la intersección sigue sin estar en d_1, se repite el proceso análogo pero con P_{i+1} y L_{i+1}. Se iguala $P_i = P_{i+1} + \overrightarrow{p_{i+1} p_i}$.

Éste parcelamiento da lugar a la creación natural de callejones.

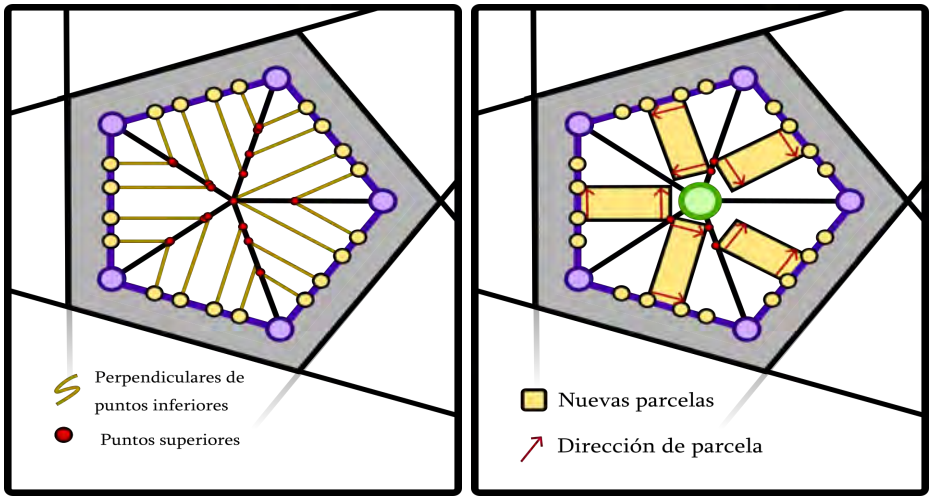


Figura 4.10: Ejemplo de creación de parcelas en medio usando la dirección al centroide (d_1 y d_2)

Si el ángulo de d_1 es 45, se extenderá la parcela con cierta probabilidad, dando lugar a parcelas más grandes y a un uso de espacio más eficiente.

Los puntos se guardan como polígono de la parcela en el siguiente órden: [p_i, p_{i+1},P_{i+1}, P_i] o en el siguiente órden [p_i, p_{i+1},P_{i+2},P_{i+1}, P_i] en el caso de algunas esquinas.

Notas: En la práctica, se tiene que comprobar si d_1 está a la derecha de L_j, si es así, se realiza el producto cruz con el vector (0,-1,0), ya que su resultado apuntaría en dirección contraria a donde se necesita. El resultado de este producto se normaliza.

4.3.3 Generación de callejones y patios internos

Los callejones se generan usando las regiones vacías en medio de cada par de parcelas consecutivas, utilizando los puntos superiores de estas.

Estos puntos inferiores se usan también para generar un patio interno en el que se colocan objetos o se generan estructuras si hay suficiente espacio.

4.3.4 Determinación del tipo de edificio

Ahora se determina el tipo de edificio que se edificará en la parcela, revisando los ángulos internos del polígono:

- Si alguno de los ángulos internos que forman la parcela es distinto de 90, el tipo de la parcela será ESQUINA.
- Si todos son de 90, se elige de F*.

F* es la tabla en la que se almacena la probabilidad que tiene cada parcela de aparecer según el distrito en el que su centroide se encuentra, y se compone de los siguientes valores:

Distrito / Probabilidad de edificio	RESIDENCIAL	OFICINA	TIENDA	MANUFACTURA	INDUSTRIAL
VERDE	0	0	0	0	0
RESIDENCIAL	0.7	0.2	0.1	0	0
COMERCIAL	0.65	0.1	0.25	0	0
MANUFACTURA	0.1	0.1	0	0.25	0.55
INDUSTRIAL	0.1	0.1	0	0.55	0.5

Transformación de tipo en parcelas totales

En la mayoría de ciudades podemos encontrar parcelas que ocupan toda la manzana, en ellas se construyen edificios que pueden aprovechar este espacio para sus actividades, i.e.: Rascacielos, bodegas y centros comerciales.

Estos edificios se generarán si el tipo de la parcela es TOTAL. Los tipos a los que se transforman se muestran a continuación:

- RESIDENCIAL → COMPLEJO_DEPARTAMENTAL : Edificios que parezcan alojar habitantes de la ciudad.
- OFICINA → RASCACIELOS : Edificios que luzcan como lugares de trabajo, administración o de gobierno.

- TIENDA → PLAZA: Aloja edificios que parezcan vender objetos o víveres, como mercados, tiendas, etc...
- INDUSTRIAL → DEPÓSITO: Aloja edificios que asemejen lugares en que se procesen materias primas o transporten objetos.
- ESQUINA → ESQUINA: Edificios especiales colocados en las esquinas de las manzanas.

De transformarse, la altura del edificio se multiplica por un valor dependiendo del tipo, dicho multiplicador está contenido en la tabla H*:

Tipo de parcela	RESIDENCIAL	OFICINA	TIENDA	MANUFACTURA	INDUSTRIAL
Modificador de altura	2	9	3	3	3

Así, un centro comercial será más alto que una casa, pero un rascacielos lo será mucho más. Si el tipo primario de la manzana no es total, el multiplicador de altura tendrá valor de 1.

4.3.5 Altura del edificio dentro de la parcela

Calcular la altura del edificio toma en cuenta los siguientes factores:

- Densidad de población en la región ocupada por la parcela [**d**]
- Altura máxima del edificio [**M**]
- Altura mínima del edificio [**m**]
- Multiplicador de altura [**h**]

La altura se calcula entonces con la siguiente fórmula:

A : (m + (M-m) • d) • h

Esta altura, junto con el polígono que forma la base, crea un poliedro que determina el volúmen ocupado por el edificio.

4.3.6 Salida

La salida gráfica del módulo de parcelas consiste en:

1. Malla de modelo de los callejones entre cada par de parcelas.
2. Malla de modelo del patio interno de la manzana (En caso de existir)

Cada parcela define una región con forma de prisma en la que el módulo de edificios genera estructuras.

4.3.7 Calificando las parcelas

Corrección:

Al estar utilizando esquemas de parcelamiento conocido, la corrección de las parcelas construidas por el algoritmo se considerará 1 (Completamente correctas).

Replicabilidad:

Se realizan las siguientes operaciones al crear las parcelas:

- g_5 = Random(0,Separación entre parcelas) : Genera un núm. que determina el ancho de cada callejón.
- g_6 = Random(0,1) : Genera un núm. en el intrevalo [0,1] para determinar el tipo de edificio generado.
- g_7 = Bernoulli(Probabilidad de unión) : Núm. aleat. que determina si dos parcelas se 'unen'.

Predictibilidad:

La predictibilidad de una parcela será qué tan similar es en tamaño a las parcelas cercanas. Sin embargo, el módulo no genera contenido gráfico directamente asociado a las parcelas, por lo que este valor es meramente simbólico y sólo se usa al evaluar la predictibilidad de los edificios.

4.4 Construcción de edificios

El módulo de edificios toma la región definida por cada parcela (Por lo general, un prisma rectangular), y genera pequeñas sub-regiones en las que se coloca y texturiza geometría para simular edificios.

Sea p_i la i-ésima parcela de tipo distinto a esquina, Sea C el conjunto de puntos que definen la parcela, Sea d_i la altura de p_i. La región (Fig. 4.11) que el módulo de edificios utiliza consiste en:

- Posición: p_i : Ubicación actual del generador del edificio
- Alcance: Vector v : (distancia(c_0,c_1),d_i,distancia(c_0,c_f)). Donde c_0, c_1, c_f son el primero, segundo y último puntos de la parcela, respectivamente
- Rotación: Tres Vectores: dir(c_0,c_1), (0,d,0), dir(c_0,c_f) representando los valores para los vectores locales X,Y,Z.

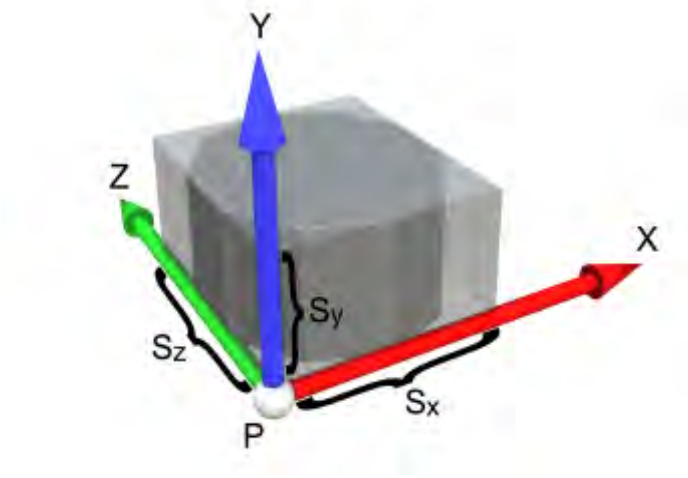


Figura 4.11: Valores del edificio. P es la posición. (s_x,s_y,s_z) es el alcance y X,Y,Z son la orientación actual del eje X,Y,Z dada por la rotación.

La posición, alcance y rotación conforman el estado del generador y son modificadas por una gramática para ir generando estructuras.

4.4.1 Producciones básicas

Las producciones de la gramática tendrán la siguiente forma:

Id (param) : Id_1 : c_1 ... Id_n : c_n

Donde Id es el identificador, param son los argumentos con que se ejecuta la producción, Id_i : c_i es una subproducción que se ejecuta con la condición c_i.

Control:

Sea x $\in \mathbb{R}^3$. Producciones de control son las que modifican la región en que se genera la geometría. (Fig. 4.12).

- M(x): Mueve la posición del generador en dirección x.
- T(x): Convierte la posición del generador en x (Coordenadas globales).
- S(x): Cambia el alcance del generador a x.
- R(x): Rota en dirección x. Recalcula los vectores locales.

Instancia:

Producciones que instancian geometría en la región (Fig. 4.13). Por ejemplo:

- Ventana(). Coloca una ventana en dirección del alcance actual.
- Pared(). Coloca una pared en dirección del alcance actual.
- Tejado(k). k $\in \mathbb{R}$. Coloca un tejado de altura k en el alcance actual.

Agrupación:

Producciones que agrupan operaciones para modificar/segmentar la región en la que se está construyendo, simplificando la creación de ciertas estructuras (Fig. 4.14).

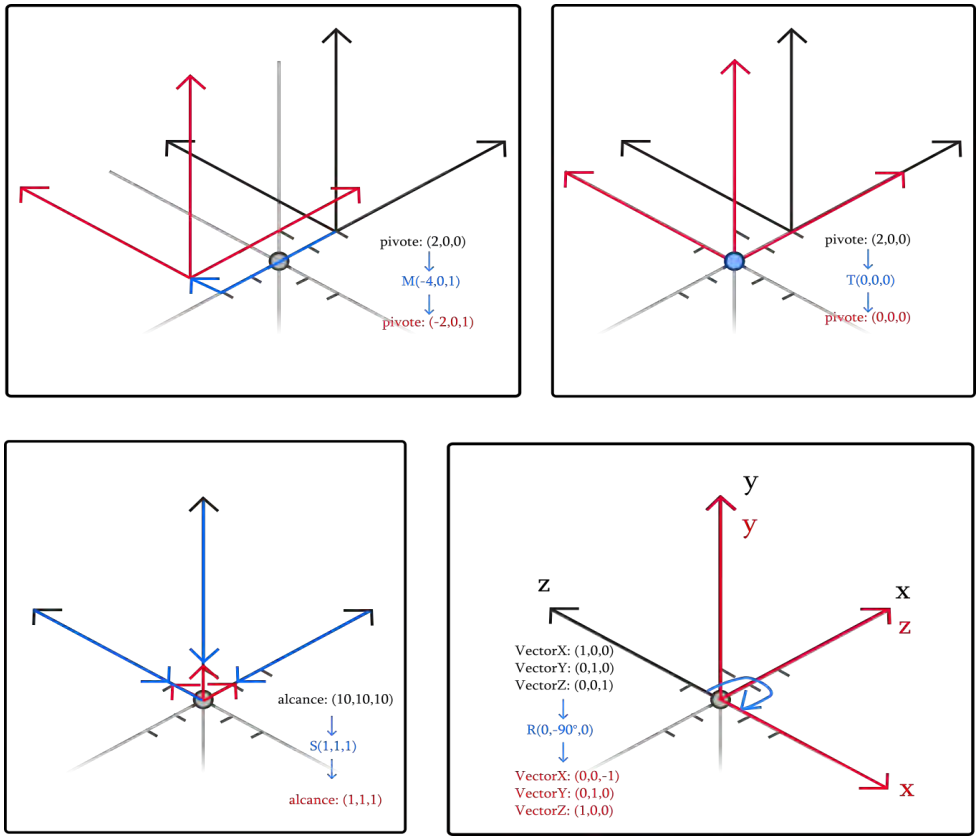


Figura 4.12: Efecto de las operaciones mencionadas. En negro: Estado del generador antes de la operación. En rojo: Estado del generador después de la operación.

No todos los objetos escalan de la misma manera. Si generamos un edificio en un espacio más grande, eso no implica que las ventanas deban ser más grandes, ya que romperían el realismo del ambiente. En cambio, se colocarían más ventanas. Marcando algunos valores como absolutos o relativos se puede forzar un tamaño fijo a las producciones.

- Subdivision(a,n,prod) : Subdivide la región actual a lo largo del eje a en segmentos de longitud n, y coloca en cada uno una producción del conjunto prod. Alternativamente, se puede sustituir n con un conjunto que determina la longitud de cada segmento con valores absolutos y relativos.

- Repetir(a,n,p) : Divide la región a lo largo del eje a en n segmentos y coloca

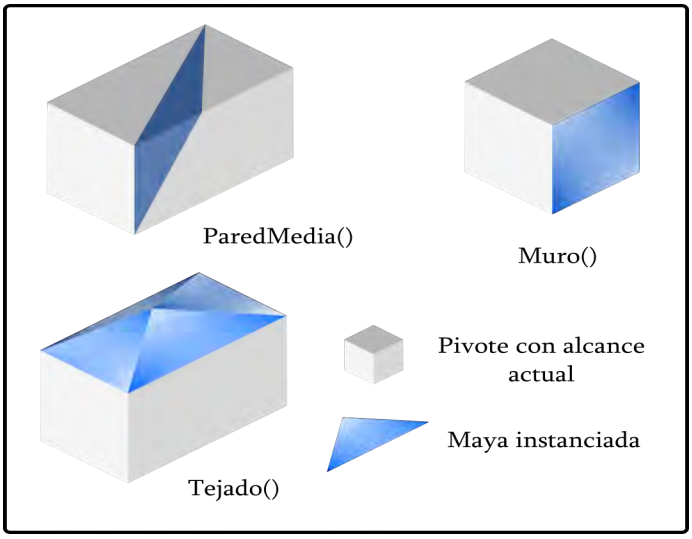


Figura 4.13: Instanciación de geometría. En negro: El estado del generador. En azul: Puntos de la geometría generada.

la producción p en cada uno.

- Contraer(a,n) : Contrae el alcance del generador en el eje a en n unidades.
- Caras(p) : Calcula las caras del estado actual, coloca la producción p en cada cara

4.4.2 Gramática para la generación

Sea **a**, **r** y **p** el alcance, rotación y posición actual del generador. Sean h, w el alto y ancho estándar especificado por el módulo de edificios. Los valores relativos tendrán una r al final del número.

La gramática de generación sería muy extensa de colocar en el documento, por lo que, en su lugar, se muestran como ejemplo las producciones necesarias para generar un edificio residencial:

Edificio(t) $\rightarrow$ Departamentos() : t==RESIDENCIAL

Departamentos() $\rightarrow$ Subdividir(Y, $\frac{a.y}{2}$, {EdificioVentanas(), EdificioVentanasMitadX()}

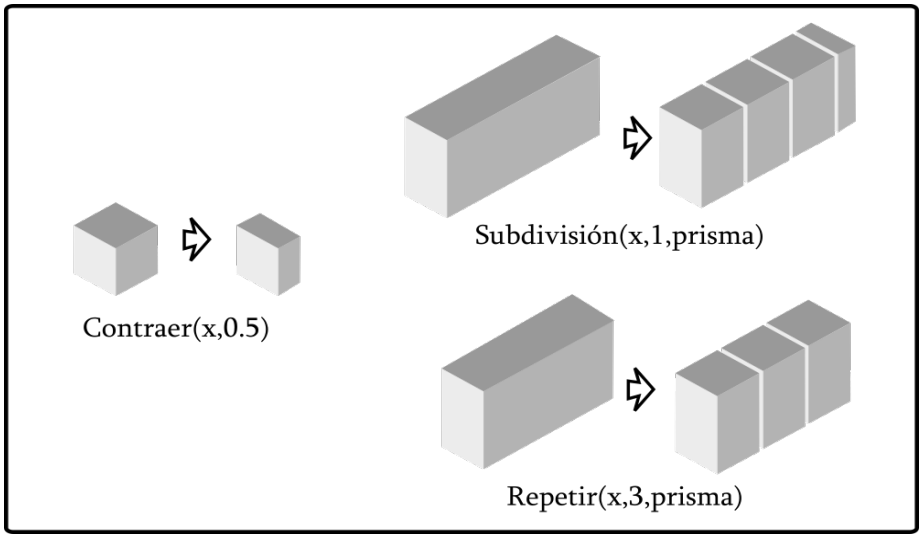


Figura 4.14: Efecto de producciones de agrupación en la región del generador

EdificioVentanasMitadX() → M(Vec($\frac{a.x}{2}$, 0, 0)) S(Vec($\frac{a.x}{2}$, a.y, a.z)) EdificioVentanas()

EdificioVentanas() → Tejado(a.y%h) S(Vec(a.x, a.y - a.y%h, a.z)) Caras(ParedVentanas())

ParedVentanas() → Repetir(Y,$\frac{a.y}{h}$ VentanasEP(h))

VentanasEP() → Subdividir(X, {$\frac{a.x\%w}{2}$, $1r$, $\frac{a.x\%w}{2}$}, {$Pared(), Ventanas(), Pared()$})

Pared() → Caras(ParedSólida())

Ventanas() → Subdividir(X, W, { CrearVentana() })

Tejado(altura) → M(Vec(0, a.y - a.y%h, 0)) S(Vec(a.x, altura, a.z)) PisoPlano() Murallas()

Murallas() → Caras(CrearParedSólida())

Toda operación que modifique el estado del generador (M,S,etc..) tiene que revertirse de modo que al terminar de procesarse todos sus símbolos, el generador pueda volver a su estado anterior. Por fines de legibilidad, dichas operaciones no fueron

colocadas en las producciones arriba mostradas.

4.4.3 Diseño de los edificios

El diseño de los edificios debe estar orientado al uso que se le va a dar [16]. En la mayoría de los casos, se tendría que ver el interior pues este es el que más información nos da sobre qué tipo de edificio es. Dado que el proyecto sólo generará fachadas de edificios (su aspecto exterior), para poder mostrar de qué tipo es un edificio, se considerará lo siguiente:

1. Un acceso al edificio: Estacionamientos, puertas y escaleras que indiquen un medio de acceso al interior.

2. Área útil: Estructuras orientadas a contener un espacio de trabajo o vivienda. Esto determina diversas características del edificio, como los materiales usados o el tipo de producciones que tomar en la gramática.

A continuación, se mostrarán algunos de los edificios generador por Urbanist.

Residencial:

Edificios destinados a servir de vivienda a una población. Deben tener ventanas que permitan la ventilación y entrada de luz para que sus residentes puedan habitarlos (Fig. 4.15).

Las puertas o portones se usan para representar el acceso al edificio, generando un edificio central con el que se accede a las demás 'alas'.

Por otro lado, la mayor parte de la malla de modelo se compondrá de ventanas y muros que ocupan todas las caras (salvo algunas excepciones).

La altura del edificio forma parte importante de su aspecto. Un edificio en un lugar de mayor densidad poblacional será más alto para poder alojar una mayor cantidad de gente, mientras que las zonas alejadas de los centros urbanos tendrán edificios de altura menor.

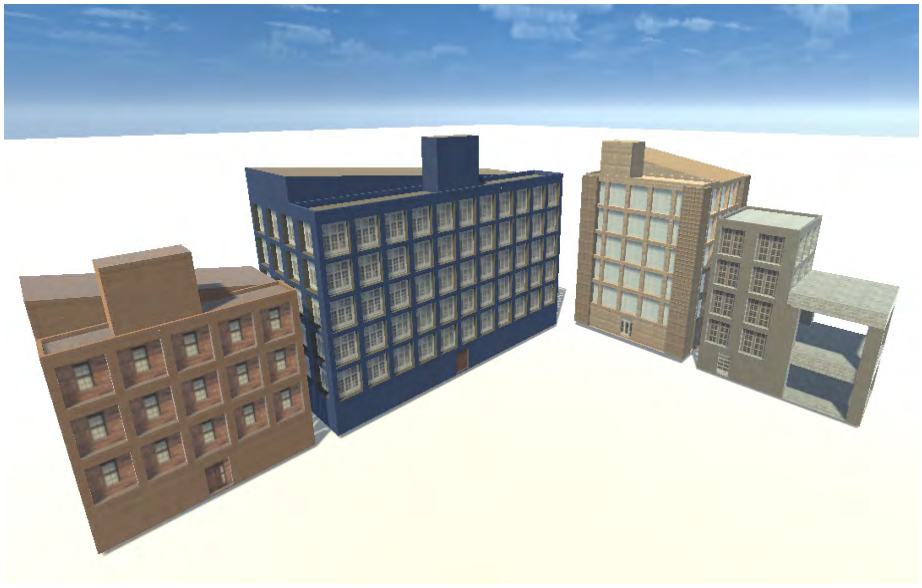


Figura 4.15: Diferentes edificios de tipo residencial.

Oficina:

Modelados de manera similar a los edificios residenciales de alta densidad poblacional: Se tiene un acceso y varios pisos conformados en su mayoría de amplias ventanas (Fig. 4.16).

Los rascacielos son modelados como tipos particulares de oficinas, que tienen mucha más altura y que poseen pilares en sus esquinas.

Compuesto:

Los edificios compuestos dividen la parcela en un eje, colocando distintos 'subedificios' en cada división, cada una con distintos materiales, haciendo que luzcan ligeramente más complejos (Fig. 4.18).

Los demás tipos de edificios se pueden observar al ejecutar el programa y recorrer la ciudad.

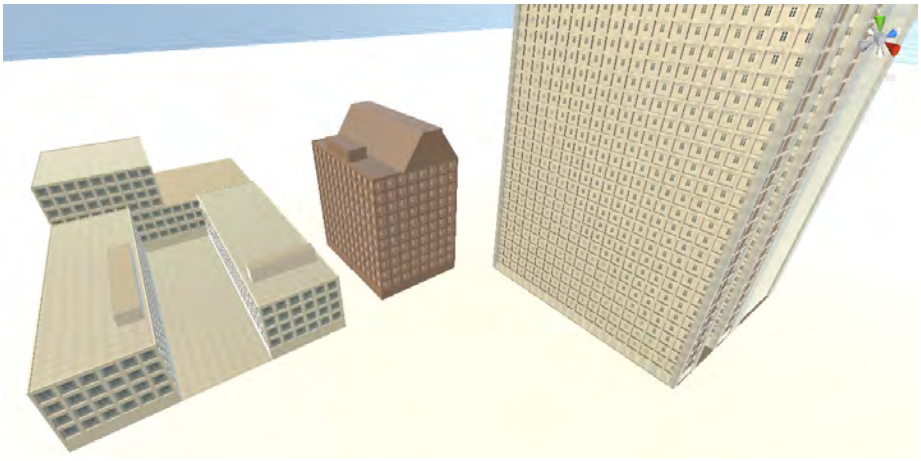


Figura 4.16: Diferentes resultados al generar oficinas.

4.4.4 Salida

La salida del modulo de edificios consiste en lo siguiente:

1. Malla 3D texturizada de cada edificio.
2. Malla 3D de los patios internos de cada parcela.

4.4.5 Calificando el módulo de edificios.

Corrección:

Hay diversas cualidades semánticas que resulta impráctico evaluar (por ejemplo: ¿El edificio tiene un diseño arquitectónico eficiente? El edificio: ¿Podría sostenerse con su forma actual?) para un equipo cuyo único propósito es entretener a un espectador. Crear una función capaz de evaluar la corrección de un edificio tan sólo por su aspecto escapa el alcance del proyecto.

Por ello, consideraremos correcto un edificio si la geometría generada permanece dentro de la región definida por la parcela. Si se hace con cuidado, la generación puede cumplir esta propiedad por construcción. La corrección teórica del módulo de edificios asume entonces un valor de 1 (Completamente correcto), puesto que se pretende evitar todo error por construcción.

Figura 4.17: Ejemplo de generación de rascacielos.

Predictibilidad:

Los edificios resultarán más 'impredecibles' mientras menos similares sean entre sí. La similitud de dos objetos sólo resulta más evidente a más cerca estén, por lo que la predictibilidad es calculada en base a qué tan parecidos son los edificios con los que tienen a su alrededor.

Cada edificio tiene un índice de similitud, calculado en base a qué tanto se parece a los edificios más cercanos en cierto radio. Es afectado según las producciones tomadas por la gramática, las dimensiones de su parcela, y los materiales usados en su creación.

Sea E_i el i-ésimo edificio generado Sea $V(E_i, r) = C$ el conjunto de edificios a distancia r o menor de E_i. Sea E_j un edificio, $E_i \in C$. Tómense en cuenta las siguientes funciones:

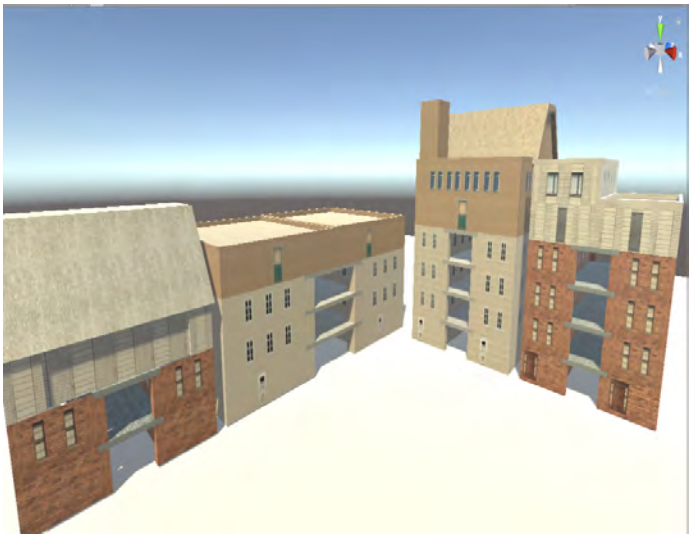


Figura 4.18: Diferentes resultados para la producción de edificios de tipo compuesto.

$$\text{Producción}(E_i, E_j) \begin{cases} 1 & \text{Si la producción inicial de la gramática al generar } E_i \\ . & \text{fue igual al generar } E_j. \\ 0 & \text{En otro caso.} \end{cases} \tag{4.1}$$

$$\text{Materiales}(E_i, E_j) \begin{cases} 1 & \text{Si los materiales de } E_i \text{ son iguales a los de } E_j. \\ 0 & \text{En otro caso.} \end{cases} \tag{4.2}$$

Sea d = ||alcance(E_j) - alcance(E_i) || la magnitud de la dirección de E_i a E_j.

$$\text{DifDimensiones}(E_i, E_j) \begin{cases} 1 - \frac{d}{4 \cdot alturaPiso} & \text{Si d} < 4 \cdot \text{alturaPiso.} \\ 0 & \text{En otro caso.} \end{cases} \tag{4.3}$$

En otras palabras, las parcelas se consideran distintas si la diferencia entre la magnitud de sus dimensiones es mayor a cuatro veces la altura del piso.

El Índice de Similitud (IS) entre E_i y E_j que determina qué tanto se parece el edificio i al edificio j, y su valor es determinado por:

$$\begin{aligned} \text{IS}(\text{E}_i, \text{E}_j) = 0.25 * \text{Producción}(\text{E}_i, \text{E}_j) + 0.5 * \text{Materiales}(\text{E}_i), \text{E}_j) \\ + (0.25) * \text{DifDimensiones}(\text{E}_i), \text{E}_j) \end{aligned} \tag{4.4}$$

La predictibilidad total del ambiente es entonces el promedio de predictibilidad de todos los edificios.

Replicabilidad:

En ocasiones, los edificios generan números aleatorios para decidir qué producciones tomar:

- g_8 = Random(0,1) : Se usa para determinar qué producciones tomar en la generación de algunos edificios.

4.5 Generador de terreno

Ahora se darán detalles sobre la generación del terreno. Ésto abarca la segmentación del terreno en bloques representativos y la colocación de vegetación y agua. Sea w,h el ancho y alto del mapa de terreno que se proporcionó como entrada.

4.5.1 Generación de superficie

La superficie del terreno se segmenta en pequeñas regiones utilizando la magnitud de modificación (Fig. 4.19). Sea m este valor.

Se segmenta el terreno en $t_1 = \frac{h}{m}$ filas a lo alto. Como $m \mid h$, $t_1 \in \mathbb{Z}$. Esas t filas son divididas cada una en $t_2 = \frac{h}{m}$ columnas a lo ancho. Como $m \mid h$, $t_2 \in \mathbb{Z}$. Las intersecciones y extremos son transformados en puntos, dejando $t_1{+}1 \cdot t_2{+}1$ puntos.

Cada punto $p_i = (x_i,y_i)$ de la *lattice* es transformado a un espacio 3D mediante la siguiente función:

$p_i = (x_i,g,y_i)$. Donde g = a·f; a es la altura del mapa de terreno en p_i y f la altura del fondo.

Los puntos forman caras con los puntos más cercanos y forman así la malla de terreno. Ésto da como resultado terrenos con el siguiente aspecto.

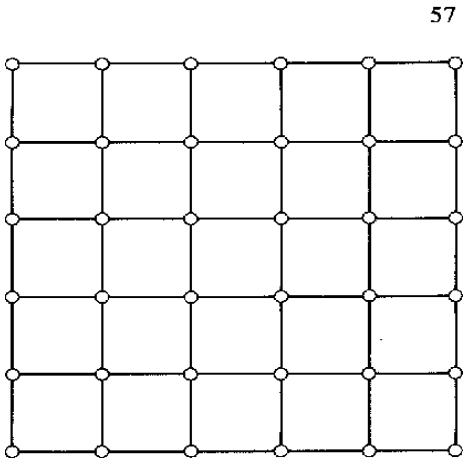


Figura 4.19: Ejemplo de polígonos formando una malla de terreno.

4.5.2 Malla de agua

La malla de agua es una malla de w de ancho por h de alto. Ésta malla se eleva a la altura de nivel del mar especificada antes. Luego se le aplica el material de agua que el usuario especificó (Fig. 4.21).

4.5.3 Generación de vegetación

Una vez colocada la superficie del terreno y la malla de agua, se procede a colocar árboles.

Colocación en superficie

Para colocar árboles en las zonas verdes se usa el mismo algoritmo que para la creación de la malla de la superficie de terreno, sólo que en lugar de ajustar una altura, se hace una consulta al mapa de distritos, si el punto cae en una zona verde, se coloca un árbol, si no, se ignora.

Colocación en manzanas verdes y patios internos

Sea M = $\{p_1, p_2, ..., p_m\}$ Sea p_i=$\{v_1, v_2, ..., v_n\}$ los puntos del i-ésimo polígono, ya sea de una manzana tipo verde o un patio interno.

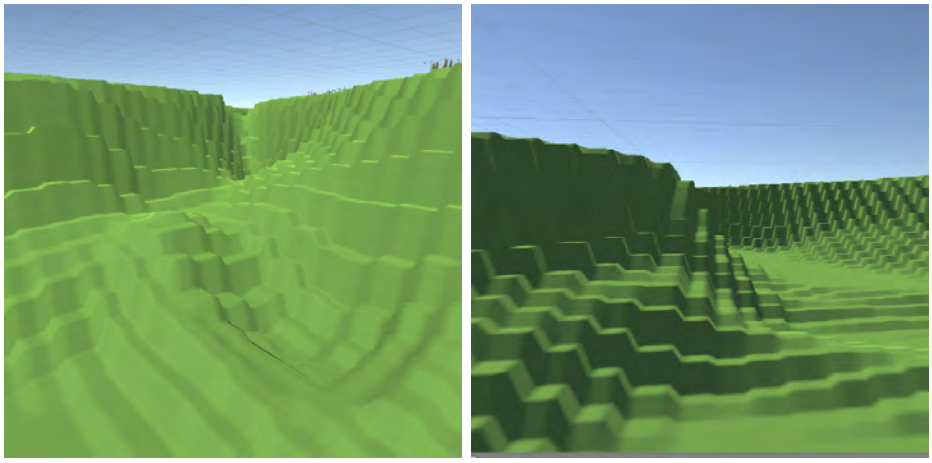


Figura 4.20: Terreno de 1200x1200 segmentado en regiones de 4x4.

Sea c_i el centroide de p_i. Para cada tripleta continua de puntos p_ip_{i+1}, se forma el triángulo $p_ip_{i+1}c$. Se calcula b = Random(mín, máx), el número de árboles que se colocarán. Estas operaciones se agregan a Seq.

Para colocar el punto sobre el que se coloca un árbol a_j, se usa el siguiente algoritmo:

Sea f=1. Sean d_1,d_2,d_3 las direcciones normalizadas p_ip_{i+1}, $p_{i+1}c$ y cp_i, respectivamente.

- Se recalcula f = p(0,f) un valor de interpolación aleatorio. Se obtiene fd_1
- Se recalcula f = p(0,f) un valor de interpolación aleatorio. Se obtiene fd_2
- Se recalcula f = p(0,f) un valor de interpolación aleatorio. Se obtiene fd_3
- Se obtiene el punto para el árbol a_j: $fd_1 + fd_2 + fd_3$

El algoritmo se puede generalizar a polígonos convexos con k lados. Éste proceso se repite b veces en cada triángulo hasta completar el polígono, para los m polígonos.

Colocación en banquetas

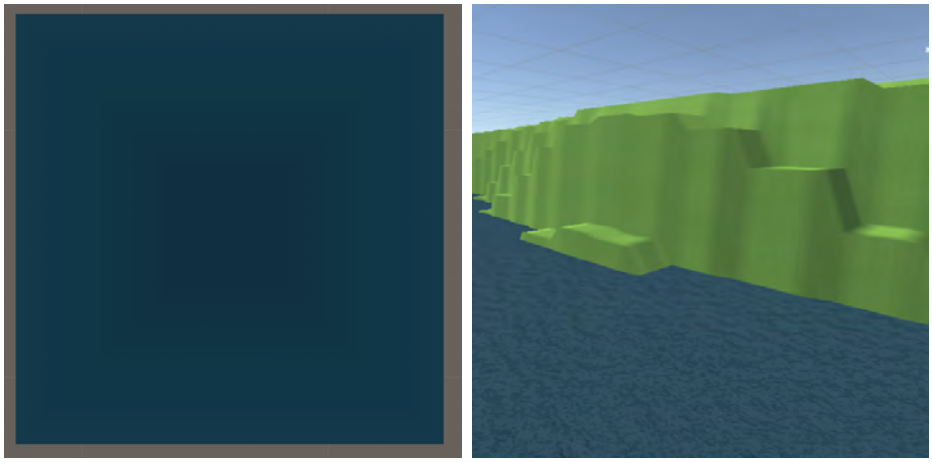


Figura 4.21: Izquierda a derecha: Malla de agua de 1200x1200. Agua colocada sobre superficie de terreno.

Los puntos en los que se colocan árboles sobre banquetas son proporcionados por el módulo de manzanas, y son generados aquí, instanciando un árbol sobre cada punto en la lista.

4.5.4 Salida

La salida del módulo de terreno es la malla 3D del terreno, la malla 3D del agua y la malla 3D de todos los árboles colocados en patios internos, parques y regiones de terreno vacías.

Capítulo 5

Prueba del algoritmo

El punto central del trabajo es probar qué tan efectiva está siendo la interpretación del diseñador al construir las soluciones, se deben generar entonces conjuntos que representen la teoría con que fue modelado el problema, y compararlos con la calidad que los usuarios perciben.

La prueba del algoritmo seguirá el siguiente flujo:

- Generación de soluciones características.
- Selección de población de interés.
- Diseño del recorrido y encuesta.
- Recopilación de resultados.

5.1 Generación de soluciones características

Para generar el conjunto de soluciones características se crearán conjuntos de prueba: Agrupaciones de parámetros y variables fijos en los que el diseñador busca expresar cierta característica.

Dichos conjuntos y las características que buscan expresarse en las ciudades se muestran a continuación:

5.1.1 Conjunto estándar:

Entradas y parámetros que tendrá el algoritmo por defecto y buscan simular una ciudad real. Debido a la naturaleza caótica de las ciudades, se generarán elementos con baja predictibilidad ($< 30\%$) y alta corrección ($> 80\%$).

Entradas para este conjunto:

Figura 5.1: Mapa de zonas, terreno y densidad para el conjunto estándar.

Ejemplo de algunos valores en los parámetros:

Parámetro	Valor
Probabilidad de disolver camino	4%
Probabilidad de manzana verde	4%
Probabilidad de parcelamiento total	20%
Probabilidad de parcelamiento completo	50%
Probabilidad de unión de parcelas	20%
Intervalo de altura aleatoria	[0,1]

Extra: Ninguno.

5.1.2 Conjunto de repetición

Aquí las ciudades generadas buscan probar si lo que el diseñador percibe como 'repetitivo' es lo adecuado. Se reducen al mínimo las probabilidades en los parámetros aleatorios y se acortan los intervalos de valores en algunos módulos.

Por consiguiente, la predictibilidad del entorno será alta ($> 80\%$), al igual que su corrección.

Figura 5.2: Mapa de zonas, terreno y densidad para el conjunto repetitivo.

Los mapas de terreno y densidad no se ven porque son imágenes en blanco. Estas entradas provocan que sólo exista un distrito en la ciudad, y que todos los edificios tengan la misma altura.

Los parámetros aleatorios se reducen al mínimo, para que los elementos generados tengan poca variedad:

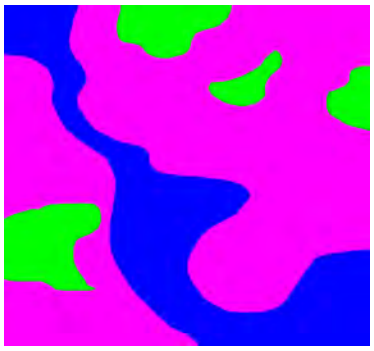
Parámetro	Valor
Probabilidad de disolver camino	0%
Probabilidad de manzana verde	0%
Probabilidad de parcelamiento total	0%
Probabilidad de parcelamiento completo	0%
Probabilidad de unión de parcelas	0%
Intervalo de altura aleatoria	[0,0]

Extra:

- Sólo existe un material con que generar los edificios.
- Sólo se puede generar un tipo de edificio (Residencial)

5.1.3 Conjunto de caos

Este conjunto representará la definción de desorden dada por el diseñador, colocando valores extremos en entradas y parámetros. Se busca una predictibilidad aún más baja que la del conjunto estándar ($< 15\%$), sin que deje de ser correcta la generación.



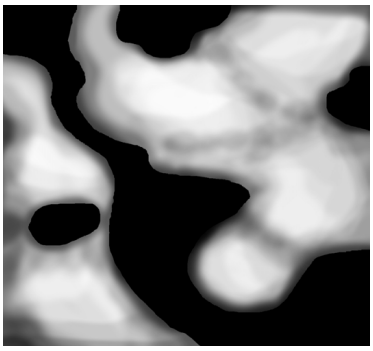


Figura 5.3: Mapa de zonas, terreno y densidad para el conjunto repetitivo.

Los mapas de densidad y terreno no se ven porque son una imagen en blanco. Estas entradas provocan que sólo exista un distrito en la ciudad, y que todos los edificios tengan la misma altura dentro de él.

Ejemplo de algunos valores en los parámetros:

Parámetro	Valor
Probabilidad de disolver camino	8%
Probabilidad de manzana verde	2.5%
Probabilidad de parcelamiento total	25%
Probabilidad de parcelamiento completo	60%
Probabilidad de unión de parcelas	25%
Intervalo de altura aleatoria	[-6,6]

Extra:

- Todos los materiales pueden ser usados al generar cualquier edificio.
- Todos los tipos de edificio tienen la misma probabilidad de aparecer.

5.1.4 Selección de soluciones

Los conjuntos de prueba se utilizan para generar las soluciones que mejor representen las características teóricas planteadas por el diseñador.

De entre 50 ciudades generadas en cada conjunto con diferentes semillas, se selecciona la de mejor calificación obtenida en las funciones de evaluación, según la característica que se busque optimizar en cada conjunto:

1. Conjunto estándar : Ciudad con corrección más alta y predictibilidad más cercana a 30%
2. Conjunto de repetición : Ciudades con predictibilidad más alta.
3. Conjunto de caos: Ciudades con predictibilidad más baja.

Así, una ciudad con baja predictibilidad será elegida si fue generada en el conjunto de repetición, pero no si fue generada en el conjunto estándar.

Los elementos característicos finales, junto con una pequeña imagen de la ciudad, se muestran a continuación.

Ciudad característica del conjunto estándar (Fig. 5.4):

- Semilla: 2520
- Predecibilidad: 28%
- Corrección: 98%

Ciudad característica del conjunto predecible (Fig. 5.5):

- Semilla: 77860
- Predecibilidad: 92%
- Corrección: 100%

Ciudad característica del conjunto de caos (Fig. 5.6):

Figura 5.4: Imágen de la ciudad característica para el conjunto estándar.

Figura 5.5: Imágen de la ciudad característica para el conjunto predecible.

- Semilla: 47511
- Predecibilidad: 11%
- Corrección: 95%

5.1.5 Selección de población de interés

Se tomará como población para realizar el estudio individuos con conocimiento de entornos urbanos, o que tengan afinidad a medios de entretenimiento digitales:

- Habitantes de ciudades.

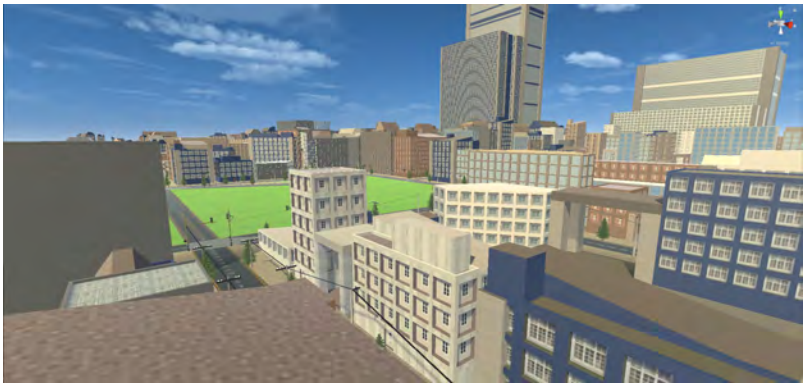


Figura 5.6: Imágen de la ciudad característica para el conjunto de caos.

- Consumidores recurrentes de contenido audiovisual en formato digital (Videojuegos, animaciones, películas, etc...).
- Modeladores o animadores 3D.
- Estudiantes / Profesionales de Arquitectura o Ingeniería civil.
- Diseñadores y/o programadores de videojuegos.

5.2 Recopilación de resultados.

Para obtener los resultados se necesitan los resultados de las funciones de evaluación (los cuales ya se tienen) y la retroalimentación directa del usuario. Estas últimas se obtuvieron mediante encuestas a usuarios luego de que tomaran un recorrido virtual pregrabado en vídeo atravesando los elementos que forman las soluciones características de cada grupo.

El recorrido tiene la desventaja de tomar sólo una parte de los edificios, parcelas y caminos, por lo que puede no contener los mejores representantes: Si por casualidad tomara una parte de la ciudad densamente parecida, el usuario percibiría la ciudad como predecible, incluso cuando el resto del ambiente no lo es. Por ello, se anexan imágenes de las ciudades que muestran una parte más general del entorno.

Se hicieron tres recorridos en total: Para cada grupo, se seleccionó el elemento característico y se recorrieron las calles y edificios durante 1 minuto y medio en el siguiente orden:

1. Conjunto caótico
2. Conjunto predecible
3. Conjunto estándar

Luego de completarlos, se pidió al participante responder la siguiente encuesta.

5.2.1 Encuesta

El siguiente estudio tiene como fin analizar la calidad de las ciudades generadas por el programa Urbanist. Respóndase sólo al completar los 3 recorridos.

Por favor, responde con honestidad a las siguientes preguntas trazando un circulo sobre el número que corresponde a tu respuesta.

1. ¿Qué tan predecible consideras que es la ciudad 1?
   1. Muy caótica 2. Algo caótica 3. Equilibrada 4. Algo predecible 5. Muy predecible
2. ¿Qué tan predecible consideras que es la ciudad 2?
   1. Muy caótica 2. Algo caótica 3. Equilibrada 4. Algo predecible 5. Muy predecible
3. ¿Qué tan predecible consideras que es la ciudad 3?
   1. Muy caótica 2. Algo caótica 3. Equilibrada 4. Algo predecible 5. Muy predecible
4. ¿Cuántos errores notaste en la ciudad 1?
   1. Ningún error 2. Pocos errores 3. Varios errores 4. Muchos errores
5. ¿Cuántos errores notaste en la ciudad 2?
   1. Ningún error 2. Pocos errores 3. Varios errores 4. Muchos errores
6. ¿Cuántos errores notaste en la ciudad 3?
   1. Ningún error 2. Pocos errores 3. Varios errores 4. Muchos errores

7. ¿Qué tan interesante es la ciudad 1?
   1. Muy interesante 2. Algo interesente 3. Normal 4. Poco interesante 5. Aburrida

8. ¿Qué tan interesante es la ciudad 2?
   1. Muy interesante 2. Algo interesente 3. Normal 4. Poco interesante 5. Aburrida

9. ¿Qué tan interesante es la ciudad 3?
   1. Muy interesante 2. Algo interesente 3. Normal 4. Poco interesante 5. Aburrida

   Cada ciudad tiene un número asociado al órden en que la viste por favor, circula el número que creas que corresponde a la ciudad que mejor responde a la pregunta planteada.

10. ¿Cuál de las ciudades luce más natural?
    1 2 3

11. ¿Cuál de las ciudades luce menos natural?
    1 2 3

12. ¿Cuál de las ciudades tiene el mejor aspecto?
    1 2 3

13. ¿Cuál de las ciudades tiene el peor aspecto?
    1 2 3

14. Selecciona cuales (si es que hay alguna) ciudades podrían utilizarse en un videojuego, animación o simulación que requiera un ambiente urbano 3D.
    1 2 3

15. Observaciones extra _____

Fin de encuesta.

Capítulo 6

Resultados y conclusiones

6.1 Recopilación y análisis de resultados

Por cuestiones de la contingencia en el momento de realización (Enero 2022), las encuestas se aplicaron mediante un formulario de Google, que se puede encontrar aquí.

La prueba del algoiritmo se condujo en un grupo de 30 personas, con candidatos que vivieran o trabajaran en una ciudad, o que tuvieran afinidad a contenido que requiera ambientes virtuales: Videojuegos, animaciones o simulaciones. Estos están más familiarizados con estos ambientes y pueden medir mejor la replicabilidad y corrección de las ciudades generadas, así como ver qué tan viable sería su uso en proyectos que hagan uso de este contenido.

Para fines prácticos, se tomará el valor numérico de las preguntas que se responden en forma de escala lineal para el análisis de los datos (Por ejemplo, si se responde '5. Muy predecible' en la pregunta 1, ese valor sumará 1 a promedios y varianzas de predictibilidad, mientras que será 0.2 si se responde '2. Algo caótica').

Los resultados de las encuestas pueden verse en el Anexo 1 de este documento. A continuación se hace el vaciado y análisis de los datos.

6.1.1 Predictibilidad

Ciudad / Valor	Más votado	Promedio	Varianza	Desviación Estándar
Ciudad 1	3. Equilibrada (33.3%)	0.49	0.07	0.26
Ciudad 2	5. Muy predecible(53.3%)	0.81	0.05	0.22
Ciudad 3	3. Equilibrada (56.7%)	0.5	0.046	0.21

Como se ha mencionado, las ciudades tienen cierta naturaleza caótica, por lo que no es de extrañar que la ciudad 3, que es el conjunto representativo de las soluciones estándar, fuese percibida como equilibrada en esta característica.

No sólo eso, sino que el 56% de los usuarios opinaron que era equilibrada, y con una varianza menor, este valor es el dominante en los resultados para la ciudad 1.

La ciudad 2 obtuvo el resultado esperado, con una predictibilidad alta y poca variedad en los resultados. Una vez más, más de la mitad de los usuarios votaron por el resultado dominante (Muy predecible).

Por otro lado, la ciudad 1 no tuvo un descenso tan marcado en la predictibilidad comparado con la ciudad 3, pues tienen un promedio casi idéntico. Sin embargo, la ciudad 1 tiene mayor varianza en los datos, y un 40% de las opiniones registradas la sitúan del lado caótico, por lo que se puede afirmar que es ligeramente más caótica que la ciduad 3, similar a los resultados en las funciones de calificación.

No fue posible definir un estándar para la predictibilidad: Varios comentarios en las encuestas volvierón imposible el identificar un patrón que sirva para determinar si una ciudad es predecible: Algunos consideran predecible el plan hipodámico, y otros que es un error que existan cruces triangulares.

Se necesitan más tipos de edificios y materiales, más cambios en las vialidades, como segundos pisos, puentes, avenidas y pequeños callejones para que haya mayor control en la variedad del contenido generado.

6.1.2 Corrección

Ciudad / Valor	Más votado	Promedio	Varianza	Desviación Estándar
Ciudad 1	1. Ningún Error (50%)	0.79	0.06	0.25
Ciudad 2	1. Ningún Error (56.7%)	0.8	0.05	0.24
Ciudad 3	1. Ningún Error(53.3%)	0.8	0.05	0.24

Dos errores principales fueron detectados en las ciudades:

- El 'parpadeo' de edificios: En ocasiones, estos titilaban durante un momento.
- Intersección con árboles: A veces, estos intersectan semáforos y postes eléctricos.

Ambos errores tienen una baja incidencia, he ahí el por qué la corrección sigue teniendo un valor elevado.

Muy pocos usuarios señalaron edificios sobrepuestos como error, incluso siendo esta la principal métrica usada en las funciones de evaluación para la corrección. Esto se debe en gran parte a que cuando los edificios se intersectan entre sí, lo hacen en regiones no visibles al espectador.

Aunque los errores no compromenten la calidad del ambiente para la mayoría de los espectadores, deben ser corregidos si se pretende competir en la industria.

6.1.3 Interés

Ciudad / Valor	Más votado	Promedio	Varianza	Desviación Estándar
Ciudad 1	1. Muy interesante (50%)	0.76	0.07	0.26
Ciudad 2	3. Normal (26.7%)	0.42	0.1	0.31
Ciudad 3	3. Normal (33.3%)	0.66	0.1	0.32

La ciudad percibida como mayor interés corresponde también a la evaluada como más caótica por las funciones de evaluación, teniendo una varianza y una desviación estándar menor en las encuestas, con una opinión dominante de 'Muy interesante', con exactamente la mitad de los participantes.

La ciudad 2, por otro lado, tuvo un bajo interés, contrario a la predictibilidad percibida. Esto resulta adecuado, ya que mientras más caótica o equilibrada esté la ciudad, mejor será percibida por el espectador.

Con un promedio de 0.66, la ciudad 3 genera un grado aceptable de interés suficiente para poder ser usada en proyectos que requieran ambientes virtuales. La alta varianza sugiere que un agregar un poco más de caos genera ciudades más interesantes.

Uno de los comentarios de la encuesta ofrece una perspectiva interesante respecto a este resultado.

> Todo es demasiado cuadrado. Aunque las ciudades sean así, en un videojuego o animación preferiría no ver lo mismo que veo en la vida real.

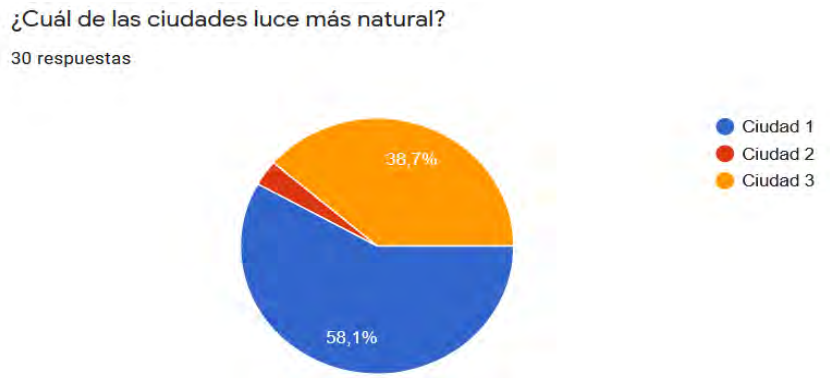
Esto apoya el hecho de que las opiniones de los espectadores respecto a qué es adecuado de ver en una ciudad son muy variadas, para las que no existe una única respuesta.

6.1.4 Aspecto, naturalidad y viabilidad.

Se puede establecer una relación entre el nivel de interés de la ciudad y la calidad de la misma: Las ciudades 1 y 3, que más interés generaron, son las de mejor aspecto y que más naturales se ven, además de ser también las mejores candidatas para su uso en proyectos.

Esta afirmación es también respaldada con la opinión sobre la ciudad 2, que fue la de peor aspecto, la menos natural, y también la menos viable para su uso en proyectos.

Aunque no de manera tan extrema, la predictibilidad también forma parte importante en la calidad: Las ciudades 1 y 3, con una predictibilidad equilibrada, ocupan



más del 96% en la naturalidad y más del 90% de las ciudades con mejor aspecto.

Que sean las de mejor aspecto no implica que su aspecto sea bueno, pero más de $\frac{2}{3}$ de los participantes apoyaron el uso de la ciudad 1 en proyectos, y más de la mitad, el uso de la ciudad 3.

Comparando las funciones de calificación y los análisis del usuario, ahora sabemos

Figura 6.1: Distribución de viabilidad de las ciudades. En el momento de captura, una persona más ya había respondido la encuesta.

que la interpretación del diseñador de interés o corrección está bien en la mayoría de aspectos, y el desarrollo del algoritmo puede continuar por el mismo camino.

Sin embargo, también se notaron errores que no estaban contemplados en las funciones de evaluación, ayudando así al diseñador a tener un mejor entendimiento del contenido que crea.

6.1.5 Replicabilidad

En las ciudades generadas, se pudo comprobar que tanto la secuencia de números aleatorios generados (Anterior edición de la tesis), como las operaciones realizadas, eran idénticas en dos generaciones seguidas.

6.2 Conclusiones

En general, el algoritmo ofrece suficiente control para crear ciudades con un nivel aceptable de calidad, ya que el interés y aspecto de las ciudades generadas fueron, en promedio, percibidos de manera positiva.

Esto no sería suficiente por sí mismo de no ser porque es posible configurar el algoritmo para generar una ciudad repetitiva, permitiendo la creación de un amplio abanico de contenido, con fallos propios de cualquier proyecto.

No se puede afirmar que exista o no una correlación entre la predictibilidad y el interés en un problema con tantas variables y factores como la creación de una ciudad, ya que cada persona tiene su propio criterio para lo que considera 'interesante' y resultaría imposible satisfacerlo para todos. Tan sólo se puede crear un algoritmo que sea capaz de adaptar su contenido al contexto necesario o a la opinión del diseñador.

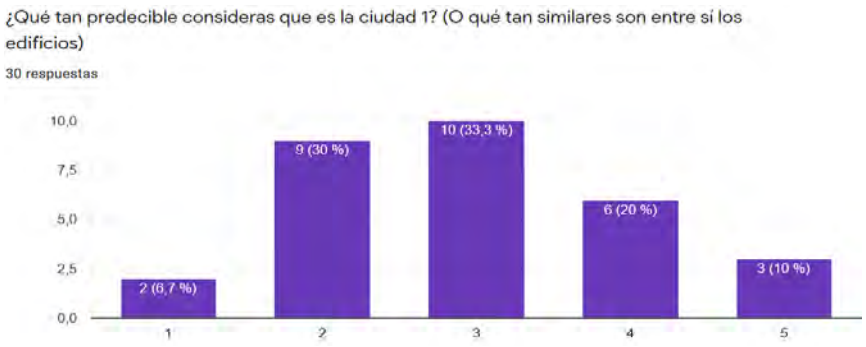
En este aspecto, la separación del algoritmo en parámetros simples y avanzados hace más controlable el algoritmo para quien desee hacer uso de todo el potenial que ofrece el programa, evitando una búsqueda extensiva de los parámetros óptimos.

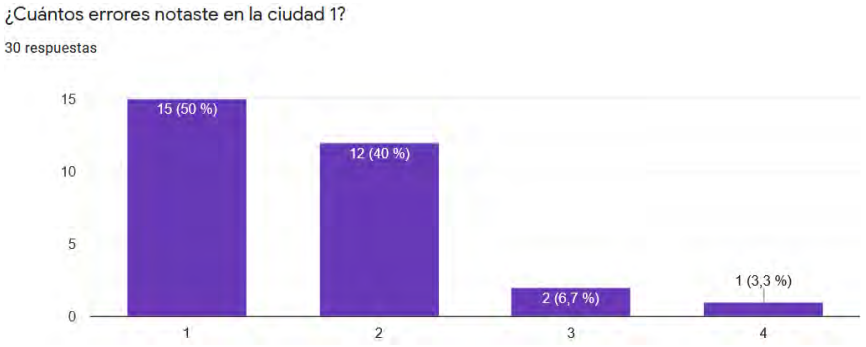
Los resultados apoyan la hipótesis de que las gramáticas son una buena opción en la generación procedural de ciudades virtuales, ofreciendo un nivel positivo de calidad y un alto control en el contenido generado, con un alto potencial de mejora.

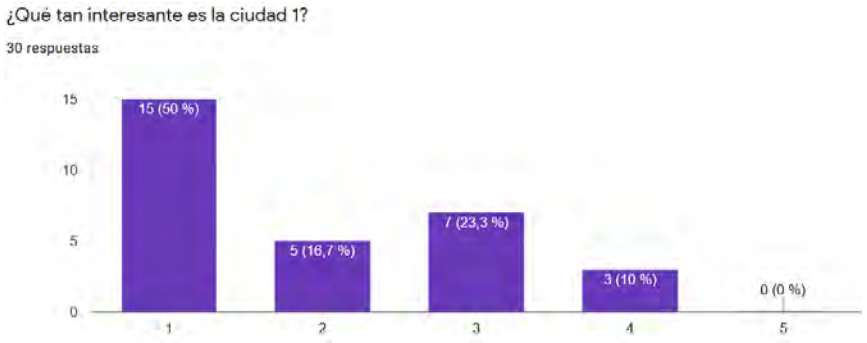
6.2.1 Trabajo futuro

1. Corrección de cálculos de normales (Edificios que parpadean)
2. Adición de más fachadas de edificios y producciones para el generador de edificios.
3. Suavizado y correcto mapeado de cruces de caminos.

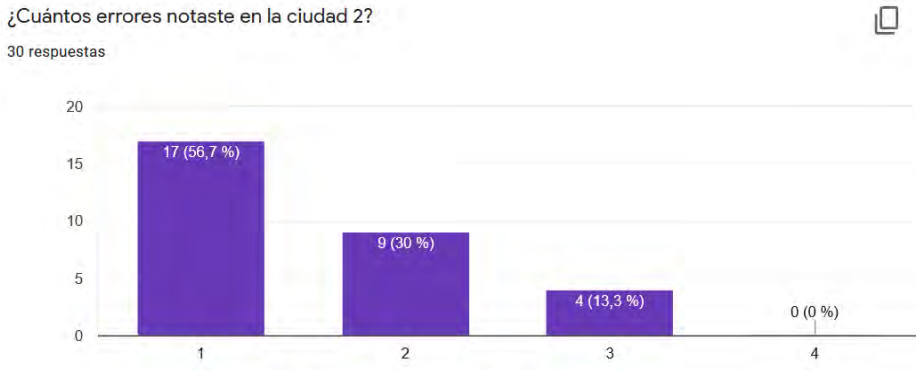
Anexo 1 : Resultados de encuestas





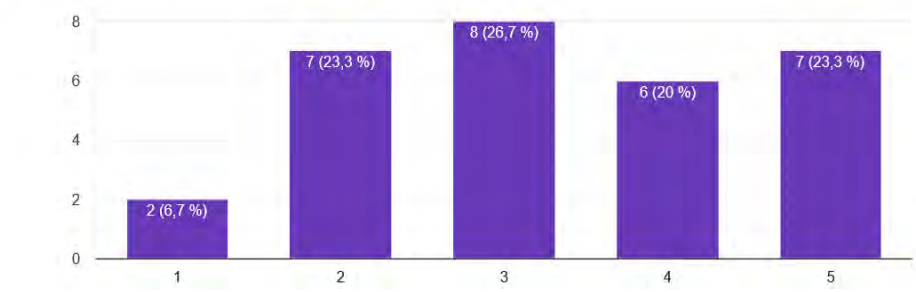

¿Qué tan interesante es la ciudad 1?
30 respuestas
15
10
5
0
15 (50 %)
5 (16,7 %)
7 (23,3 %)
3 (10 %)
0 (0 %)
1
2
3
4
5


¿Qué tan predecible consideras que es la ciudad 2?
30 respuestas
20
15
10
5
0
0 (0 %)
1 (3,3 %)
6 (20 %)
7 (23,3 %)
16 (53,3 %)
1
2
3
4
5


¿Cuántos errores notaste en la ciudad 2?
30 respuestas
20
15
10
5
0
17 (56,7 %)
9 (30 %)
4 (13,3 %)
0 (0 %)
1
2
3
4

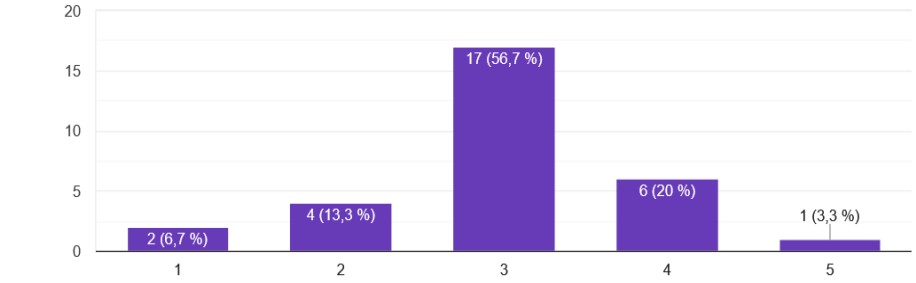
¿Qué tan interesante es la ciudad 2?

30 respuestas



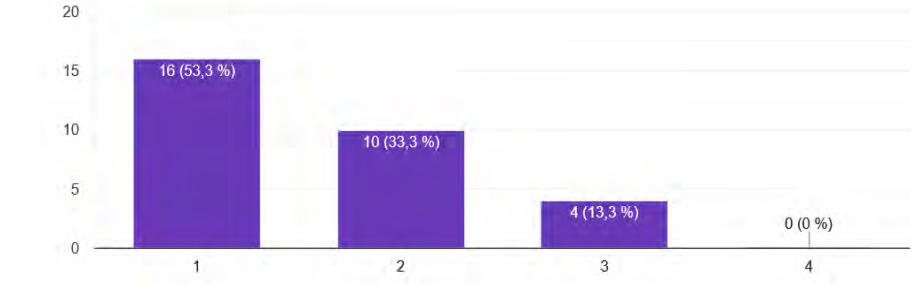
¿Qué tan predecible consideras que es la ciudad 3?

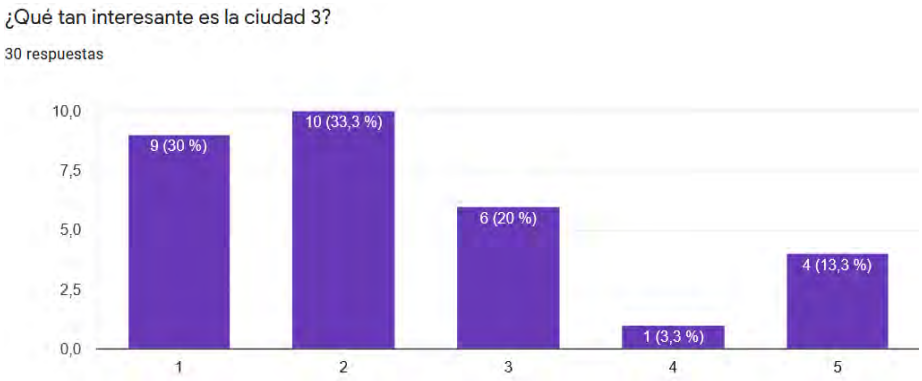
30 respuestas



¿Cuántos errores notaste en la ciudad 3?

30 respuestas




¿Qué tan interesante es la ciudad 3?
30 respuestas
10,0
7,5
5,0
2,5
0,0
9 (30 %)
10 (33,3 %)
6 (20 %)
1 (3,3 %)
4 (13,3 %)
1
2
3
4
5

Anexo 2: Imágenes extra

Imágenes de ciudad generada con el conjunto de parámetros estándar:

Figura 6.2: Vistas aérea e isométrica de la ciudad estándar.

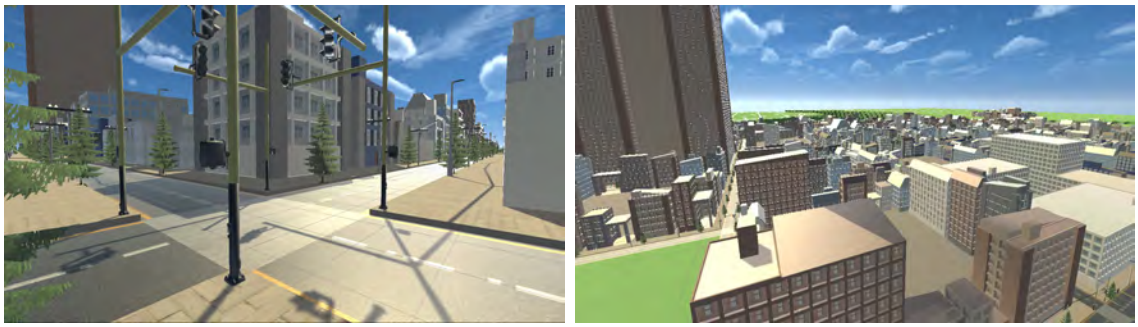


Figura 6.3: Tomas extra de la ciudad estándar.

Imágenes de ciudad generada con el conjunto de parámetros predecible:

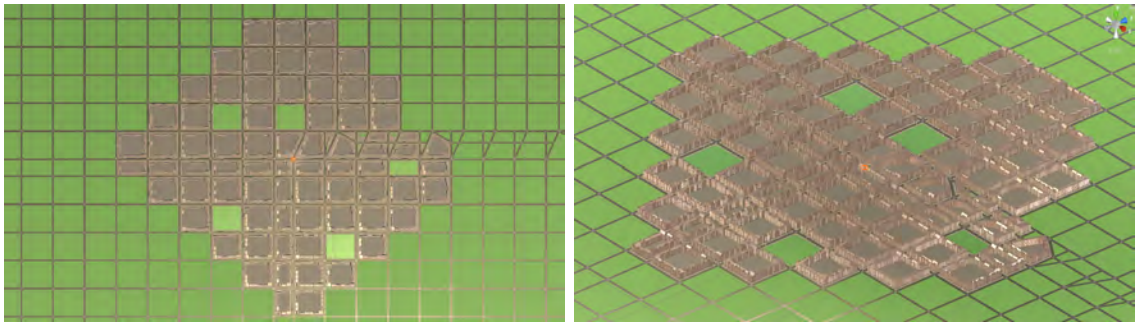


Figura 6.4: Vistas aérea e isométrica de la ciudad predecible.

Figura 6.5: Tomas extra de la ciudad predecible.

Imágenes de ciudad generada con el conjunto de parámetros caóticos:

Figura 6.6: Vistas aérea e isométrica de la ciudad caótica.

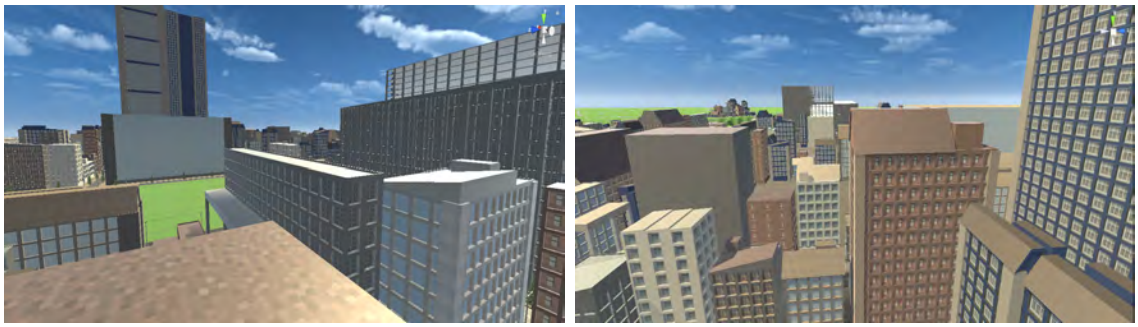


Figura 6.7: Tomas extra de la ciudad caótica.

Anexo 3: Código - Ejemplos

Todo el código desarrollado fue hecho por el autor de la tesis. A excepción de:

- Intersección de polígonos (Fue usado Clipper).
- Determinar si un vector está a la izquierda o derecha (Copiado de un foro).

Ejemplo: Semillas de prueba del algoritmo.

```
private readonly int[] seedTestCases = {
          3,44,72,125,5,6,5361498,35840,1658,32891,
          45016,54821,68927,77860,84690,91235,16634,23004,38765,47511,
          56922,687514,78801,89633,90014,12405,22038,39714,44180,59130,
          251,3028,459302,52,668,7950,81,1,8040,32503,
          9891,84230,900611,105842,2520,4170,5668,9814,92,109520
      };
```

Ejemplo: Adición de puntos característicos.

```
   public void CreateRoads(GameObject start_Node, List<GameObject>
       neighbors) {
       Vector3 node_Position = start_Node.transform.position;
       int j = neighbors.Count - 1;
       // Add the characteristic points determind by the nieghbors
       List<Vector3> cross_Polygon = new List<Vector3>();
       for (int i = 0; i < neighbors.Count; i++) {
           int char_Type = 0;
           float road_Thickness1 = road_Network.GetEdge(start_Node,
               neighbors[j]).thickness, road_Thickness2 =
               road_Network.GetEdge(start_Node, neighbors[i]).thickness;
           /**/
```

```
        List<Vector3> char_Points = getCharPoints(node_Position,
            neighbors[j].transform.position,
            neighbors[i].transform.position, road_Thickness1,
            road_Thickness2, out char_Type);
        List<Vector3> connection_List1 = GetConnectingList(start_Node,
            neighbors[j]);
        List<Vector3> connection_List2 = GetConnectingList(start_Node,
            neighbors[i]);
        switch (char_Type) {
            case SIMPLE: // If simple, there is only one neighbor on
                startNode, so only points
                connection_List1.Add(char_Points[0]);
                connection_List1.Add(char_Points[1]);
                road_Network.ConnectWithValue(start_Node, neighbors[j],
                    connection_List1);
                break;
            case ROUND: // If round or intersected, add first char point
                to first list and second char point to second list
            case CROSSED: // If intersection, the single feature point
                is added to both both lists
                connection_List1.Insert(0, char_Points[0]);
                connection_List2.Insert(connection_List2.Count %
                    2,char_Points[1]);
                road_Network.ConnectWithValue(start_Node, neighbors[j],
                    connection_List1);
                road_Network.ConnectWithValue(start_Node, neighbors[i],
                    connection_List2);
                break;
        }
        // If segment has 4 points, it forms a road, so it's added to
            segment points list
        if (connection_List2.Count == 4) {
            segmentPoints.Add(connection_List2);
        }
        if (connection_List1.Count == 4) {
            segmentPoints.Add(connection_List1);
        }
        // All points that formed the cross are added
        foreach (Vector3 v in char_Points) {
            cross_Polygon.Add(v);
        }
```

```
            j = i;
        }
        crossPoints.Add(start_Node, cross_Polygon);
    }
```

Ejemplo: Determinando el tipo de un lote.

```
public int GetType(Allotement a){
    List<Vector3> allotement_Points = a.external_Points;
    Vector3 centroid = GenMath.centroid(allotement_Points);
    int type = Allotement.INCOMPLETE;
    float density = Auxiliar.GetPixelValueAt(centroid, density_Map);
    // First, a green allotment is tried to be randomly generated
    if(GenMath.GetRandom(0f,1f)<greenAllotementChance) {
        type = Allotement.GREEN;
    } // Then, allotment dimensions are checked, if not able to place
        buildings on them, they are converted to green allotements
    else if (a.area < minimum_Allot_Area || a.thickness <
        minimum_Allot_Thickness) {
        type = Allotement.GREEN;
    } // Total types can only have 4 sides, since most total buildings are
        squared
    else if (density > TOTAL_THRESHOLD && allotement_Points.Count==4){
        if (GenMath.Bernoulli(totalChance))
            type = Allotement.TOTAL;
    }
    else if (density > COMPLETE_THRESHOLD){
        if (GenMath.Bernoulli(completeChance))
            type = Allotement.COMPLETE;
    }
    return type;
}
```

Ejemplo: Determinando la altura de una parcela.

```
public float GetBuildingHeight(Vector3 location, int type) {
    District d = district_Manager.GetDistrict(location);
    float pixel_Value = Auxiliar.GetPixelValueAt(location, density_Map),
        max_Height = d.GetMaxHeight(type), min_Height =
        d.GetMinHeight(type);
```

```
    float difference = (max_Height - min_Height) * pixel_Value;
    // Height is determined according to type and density, and a randomly
        generated floor is added or sustracted
    float height = min_Height + difference -
        GenMath.GetRandom(minRandomHeight,maxRandomHeight);
    if (height < min_Height) {
        height = min_Height;
    }
    return height;
}
```

Ejemplo: Creación de edificio residencial.

```
public Production[] Residential(object[] parameters) {
        Production mainBuilding = null;
        if (GenMath.Bernoulli(0.5f))
        {
            mainBuilding = new Production(BuildingWithEntrance,new object[]
                {new Production(Appartment)});
        }
        if (GenMath.Bernoulli(0.5f)) {
            mainBuilding = new Production(BuildingWithEntrance,new object[]
                {new Production(WingedAppartment)});
        }

        Production roof_Production = new Production(CompoundRoof);
        mainBuilding = new Production(RandomPillaredBuilding,new object[] {
            new Production(FramedWindowedBuilding,
            new object[] { roof_Production, STREET_LOOKING_DOOR }) });
        return BaseEmplacement(mainBuilding);
    }
```

Ejemplo: Creación de rascacielos.

```
public Production[] Skyscraper(object[] parameters) {
        int levels = 1;
        char[] available_Axis = { 'X', 'Z' };
        char axis =
            available_Axis[GenMath.ChooseOne(available_Axis.Length)];
        Production roof_Production = new Production(PlanarRoof, new
```

```
            object[] { parent_Grammar.wallHeight / 8f });
        Production child = new Production(PillaredBuilding, new object[] {
            new Production(WindowedBuilding, new object[] { roof_Production,
            0 }), true });
        float amount = 0;
        Production firstRegion = new Production(Contract);
        Production secondRegion = new Production(Contract);
        if (GenMath.Bernoulli(0.2f))
        {
            levels = 2;
            axis = 'Y';
            child = new Production(Contract, new object[] { new char[] {
                'X', 'Z' }, 3f, child });
            amount = parent_Grammar.scope.y;
        }
        Production contraction = new Production(Contract, new object[] {
            new char[] { 'X', 'Z' }, 3f, child });
        Production[] result = { new Production(PlanarFloor),
            new Production(S, new object[] { parent_Grammar.scope +
                parent_Grammar.YVector * amount}),
            new Production(RecursiveBuilding, new object[] { levels, axis,
                new Production(DualAppartment), child })};
        return result;
    }
```

Ejemplo: Creación de números aleatorios.

```
public System.Random rng;
public List<String> operations;
public void Initialize(){
   operations = new List<String>();
}
public static float GetRandom(float minValue, float maxValue) {
    float randomNumber = (float) rand.NextDouble();
    float result = randomNumber * (maxValue - minValue);
    randomGeneratedNumbers?.Add(randomNumber);
    operations.Add("RF(" + minValue + "," + maxValue + ")");
    return minValue + result;
}
public static int GetRandom(int minValue, int maxValue)
```

```
{
    int randomNumber = rand.Next(minValue,maxValue);
    randomGeneratedNumbers.Add(randomNumber);
    operations.Add("RI(" + minValue + "," + maxValue + ")");
    return randomNumber;
}
public static bool Bernoulli(float p) {
    float randomNumber = (float) rand.NextDouble();
    randomGeneratedNumbers?.Add(randomNumber);
    operations.Add("B(" + p + ")");
    return randomNumber<=p;
}
```

Ejemplo: Creación de muros.

```
public void AddWallMesh()
    {
        Vector3[] wall_Points = GetLocalBaseScopePoints();
        Vector3 height_Vector = new Vector3(0, scope.y, 0);
        MeshGenerator.AddWallData(new Vector3[] { wall_Points[0],
            wall_Points[1], wall_Points[1] + height_Vector, wall_Points[0] +
            height_Vector },
             wall_Width, wallHeight, building.wallMesh, false);
    }
```

```
public static void AddWallData(Vector3[] points, float wallWidth, float
    wallHeight, UMesh mesh, bool inverted,
            Vector2 uvDisplacementVector = new Vector2())
        {
            float cosAngulo1 = Mathf.Cos(GenMath.AngleBetween(points[0],
                points[1], points[3], Vector3.up)*Mathf.Deg2Rad),
                cosAngulo2 = Mathf.Cos(GenMath.AngleBetween(points[0],
                    points[1], points[2], Vector3.up)*Mathf.Deg2Rad);
            float xeq1 = 0, xeq2 =
                Vector3.Distance(points[0],points[1])/wallWidth,
                xeq3 = (cosAngulo2 * Vector3.Distance(points[0], points[2]))
                    / wallWidth,
                xeq4 = (cosAngulo1 * Vector3.Distance(points[0], points[3]))
                    / wallWidth;
            float yeq1 = 0, yeq2 = 0,
```

```
            yeq3 = (Vector3.Distance(points[0],
                points[2])*Mathf.Sqrt(1-Mathf.Pow(cosAngulo2,2))) /
                wallHeight,
            yeq4 = (Vector3.Distance(points[0],
                points[3])*Mathf.Sqrt(1-Mathf.Pow(cosAngulo1,2))) /
                wallHeight;
        Vector2 uv1 = uvDisplacementVector + new Vector2(xeq1, yeq1),
            uv2 = uvDisplacementVector + new Vector2(xeq2, yeq2),
            uv3 = uvDisplacementVector + new Vector2(xeq3, yeq3),
            uv4 = uvDisplacementVector + new Vector2(xeq4, yeq4);
        List<Vector2> uvs = new List<Vector2> { uv1, uv2, uv3, uv4, uv1,
            uv3 };
        // Quad Addition
        for (int j = 0; j < 6; j++) {
            mesh.triangles.Add(mesh.points.Count + j);
            mesh.normals.Add(Vector3.up);
            mesh.uvs.Add(uvs[j]);
        }
        // Inferior triangle is added first
        List<Vector3> triangle_List;
        if (!inverted) {
            triangle_List = new List<Vector3> { points[0], points[1],
                points[2], points[3], points[0], points[2] };
        }else {
            triangle_List = new List<Vector3> { points[2], points[1],
                points[0], points[2], points[0], points[3] };
        }
        mesh.points.AddRange(triangle_List);
        mesh.TrimMax();
    }
```

Anexo 4: Repositorio

Solicitar acceso al repositorio enviando mensaje al siguiente correo:

katarsis@ciencias.unam.mx

Referencias

[1] Michael F. Barnsley, «Fractals Everywhere», Morgan Kaufman, 1999.

[2] Jon McCormack, «GENERATIVE MODELLING WITH TIMED L-SYSTEMS», pp. 9-14

[3] Przemyslaw Prusinkiewicz, Aristid Lindenmayer « The algorithmic Beauty Of Plants », pp. 3-5, 2004.

[4] Radomir Mech and Przemyslaw Prusinkiewicz. Visual Models of Plants Interacting with Their Environment. Proceedings of SIGGRAPH 96 (New Orleans, Louisiana, August 4-9, 1996). In Computer Graphics Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 397-410.

[5] George Kelly, Hugh McCabe, «A Survey of Procedural Techniques for City Generation», 2006.

[6] Yoav I H Parish,Pascal Müller « Procedural Modeling of Cities», pp. 1-8

[7] Niclas Olsson, Elias Frank «Procedural city generation using Perlin noise», Bachelor of Science in Computer Science

[8] Yakin Najahi, Urbis Terram - Designing and Implementing a Procedural City Generation Tool for Unity3D Game Engine

[9] Procedural City Generator, MSc Master's Project, Praveen Kumar Ilangovan i7834000

[10] S. Thon, J.M. Dischler, D. Ghazanfarpour, Ocean waves synthesis using a spectrum based turbulence function, Laboratory MSI, ENSIL -University of Limoges

[11] Melissa E. O'Neill, «PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation», pp. 9 - 11.

[12] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, Cameron Browne « Search-based Procedural Content Generation», pp. 1-7, 2011.

[13] Lewis Dijkstra and Hugo Poelman, «CITIES IN EUROPE THE NEW OECD-EC DEFINITION», Regional Focus, RF 01/2012.

[14] European Cities Characterization as Basis towards the Replication of a Smart and Sustainable Urban Regeneration Model, Article in Energy Procedia · March 2017

[15] Mg. Arq. Alejandra M. SGROI «MORFOLOGÍA URBANA - PAISAJE URBANO -», pp. 10 - 11, 2016

[16] Frederick S. Merritt (Deceased) Editor Jonathan T. Ricketts, Building Design and Construction Handbook

[17] Yannakakis G. N. y Togelius J., «Experience-Driven Procedural Content Generation» IEEE TRANSACTIONS ON AFFECTIVE COMPUTING, vol. 2, no 3, 2011.

[18] Jacob Olsen, «Realtime Procedural Terrain Generation», pp. 5-6 2004.

[19] Miller G. S. P.,«The Definition and Rendering of Terrain Maps» Proc. ACM SIGGRAPH,vol. 20, 1986.

Bibliografía

[20] Gillian Smith, «An Analog History of Procedural Content Generation», Applications and Expert Systems – Games, pp.1-5,2015.

[21] Javier Lluch, Emilio Camahort, Robert Vivó, «Procedural Multiresolution for plant and tree rendering», 2003.

[22] Cansın Yıldız, «An Implementation of a Simple, Efficient Method for Realistic Animation of Clouds»