UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Posgrado en Ciencia e Ingeniería de la Computación

Facultad de Estudios Superiores Cuautitlán

DESARROLLO DE UN PROGRAMA DE ENUMERACION EXACTA APLICADO AL MODELO BICOLOR DLA

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS

P R E S E N T A

SERGIO RAMIREZ GARCIA

DIRECTORA DE TESIS: DRA. SUEMI RODRIGUEZ ROMO

MEXICO, D. F. 2001

AGRADECIMIENTOS

- *Para la realización de esta tesis fue necesario la colaboración y ayuda de mis compañeros, amigos y profesores que integran el grupo estudiantil, academico y administrativo de la maestria en Ciencias e Ingeniería de la Computación, a todos ellos les agradezco su apoyo y contribución en este trabajo.*

- *Unas gracias muy especiales a la Dra. Suemi Rodríguez Romo quien desde el principio no sólo demostro su apoyo, sino que con su tiempo, perseverancia, energia e información a la investigación, hemos logrado finalizar este trabajo.*

- *Otro agradecimiento de igual importancia es para el Dr. Vladimir Tchijov por su excelente trabajo en caminatas aleatorias y por su empeño incansable en promover la investigación en la FES-C.*

- *A mi madre y hermanos que siempre han creido en mí, lo cual es un gran estimulo y motivación en la vida.*

- *Y como no, todo se los debo a mi esposa e hijos, que sin ellos no hubiera tenido la inspiración y ganas por finalizar esta meta. Muchas gracias por aguantarme*

INDICE

1. INTRODUCCION

El entendimiento del universo que nos rodea en términos de las interacciones de las partículas fundamentales y de sus propiedades es el sueño de la Física. Modelar el comportamiento de moléculas que contienen pocos átomos mediante desarrollos de software es relativamente simple, sin embargo se observa que los procesos naturales contienen sistemas muy complejos y la mayoría de las veces no es posible desarrollar programas que se adecúen perfectamente a éstos.

La importancia de este trabajo de tesis se basa en su contribución a los trabajos de investigación que se realizan entorno al entendimiento de sistemas complejos en crecimiento aleatorio mediante el desarrollo de un programa que entregue distribuciones de probabilidad calculadas por enumeración exacta en las etapas iniciales de crecimiento de estos sistemas.

El progreso sustancial en el entendimiento del comportamiento de sistemas complejos proviene de conceptos de física estadística, tales como escalamiento y en el desarrollo de la geometría fractal de Benoit Mandelbrot[1]. El avance de estas áreas parte del conocimiento detallado de sus componentes microscópicos, y de las propiedades comunes que todos los materiales poseen, sin considerar su estructura atómica y molecular.

1.1. Objetivo

El objetivo de este trabajo es el desarrollo del software de un modelo computacional de enumeración exacta en las etapas iniciales para sistemas complejos de crecimiento desordenado que proporcione las distribuciones de probabilidad en el crecimiento inicial de agregados bicolores DLA. Dicho modelo se implementará como una herramienta que cuantitativamente arroje datos estadísticos y probabilísticos del origen de crecimiento de dicho proceso, tomando como base el modelo bicolor DLA (Agregación Limitada por Difusión). Las fluctuaciones iniciales en el crecimiento de agregados son muy importantes para explicar la morfología macroscópica de los mismos.

Buscamos establecer parámetros que nos permitan explicar los diferentes regímenes morfológicos obtenidos experimentalmente usando enumeración exacta.

La complejidad computacional de este modelo tendrá que ver con los recursos computacionales requeridos para resolver estos problemas orientados al desempeño de los algoritmos en función del tiempo y del espacio

1.2. Antecedentes

La amplia diversidad de formas y fenómenos naturales que nos rodean tienen un impacto profundo sobre la calidad de nuestras vidas. Por esta sola razón, no es sorprendente que los orígenes de estas formas y cosas han sido tema de serios estudios desde la antigüedad hasta nuestros días. La caracterización cuantitativa de las formas naturales es un paso importante hacia el entendimiento de sus orígenes. Desgraciadamente han habido pocas aproximaciones computacionales hacia la descripción cuantitativa de patrones complejos y desordenados que son característica de la mayoría de fenómenos naturales. Hoy en día, con el gran avance en la computación, se abre la posibilidad de desarrollar software para procesos complejos de crecimiento aleatorio, cuya ejecución realice una enorme cantidad de cálculos en tiempos muy pequeños y con el uso tan solo de computadoras personales.

Las técnicas físicas y computacionales en la formación de procesos y patrones naturales alejados del equilibrio han tenido un lento desarrollo. Las técnicas sistemáticas y bien entendidas de mecanismos estadísticos en equilibrio no pueden ser aplicadas en la mayoría de procesos de formación de patrones.

En las últimas dos décadas esta perspectiva ha progresado sustancialmente. El trabajo interdisciplinario del pionero B. Mandelbrot ha demostrado que aquellos conceptos matemáticos[2], alguna vez creídos sin relevancia posible en el mundo real, pueden ahora proporcionarnos nuevas formas de describir y pensar acerca de un extenso y asombroso rango de estructuras y fenómenos naturales, al igual que los conceptos de escalamiento que fueron originalmente aplicados a un limitado rango de problemas tales como fenómenos críticos en el movimiento de partículas suspendidas en el aire.

En muchos casos, la aproximación a la geometría fractal desarrollada por B. Mandelbrot se usa para proporcionar una mejor interpretación intuitiva sobre el comportamiento de escalamiento.

1.2.1. La variedad de patrones en la naturaleza

La naturaleza exhibe una amplia variedad de patrones desde modelos casi completamente Euclidianos, hasta modelos complejos completamente caóticos. Esta amplia riqueza puede observarse mediante diversos patrones familiares sobre la tierra. Aunque la formación de patrones complejos como un resultado de mecanismos simples es relevante, la formación de patrones regulares con escalas de longitud bien definidas, como un resultado de procesos más complejos, es también común. Ejemplos comunes incluyen los surcos que se forman en la arena de las playas al chocar de las olas, o bien las formas de las nubes en el cielo por el flujo del viento. Otros ejemplos

menos familiares pueden ser la formación de cerros de grava sobre el desierto. Los ejemplos de patrones naturales completamente desordenados no pueden ser descritos en términos de la geometría Euclidiana o modelos que incluyen escalas de longitud bien definida como las montañas, ríos y costas. Una amplia variedad de estructuras entre estos tipos de patrones también se puede observar. Estas estructuras intermedias no pueden ser descritas en términos de cualquier modelo estadístico o modelo Euclidiano y son a menudo, los más difíciles para caracterizar en términos cuantitativos.

Las hermosas figuras formadas por el crecimiento de cristales de hielo en la formación de copos de nieve han inspirado una extensa gama de investigaciones sobre la formación de patrones. La formación de patrones simétricos pero con puntos complejos dentro de un mecanismo de crecimiento determinístico lo hace un tema exquisitamente sensible para un estudio profundo.

1.2.2. Patrones Euclidianos

Hay muchas clases de patrones que son más o menos regulares y pueden ser descritos en términos totalmente Euclidianos simples. Frecuentemente, los patrones más simples son formados cercanos al equilibrio y se van haciendo más y más complejos conforme se van alejando del equilibrio. Sin embargo es peligroso afirmar que dos patrones son lo mismo, dado que sus orígenes pueden ser explicados de la misma forma.

1.2.3. Patrones complejos y desordenados

En la naturaleza, los patrones complejos y/o desordenados son los más comunes. Ejemplos familiares incluyen cuando rompen las olas sobre la playa, la estructura interna del cuerpo humano y los paisajes naturales. Hace pocas décadas, las tareas para describir tales patrones en términos cuantitativos era casi imposible. Hoy en día, el desarrollo de computadoras de alta velocidad y la alta resolución gráfica juegan un papel muy importante en el desarrollo de este tipo de modelos. Los matemáticos han estimado por mucho tiempo que ecuaciones simples, pueden proporcionar soluciones complicadas. No obstante, solo en la última década ésto ha sido apreciado por una amplia variedad de científicos. Esta simple idea nos llama a la reflexión de lo que significa cuando un patrón es descrito como complejo. Es natural preguntarse "¿Es un patrón que puede ser interpretado mediante una sola ecuación o mediante un algoritmo complejo?".

1.2.4. Agregados

El proceso de agregación (auto-ensamble de pequeñas partículas para formar grandes estructuras) juega un papel muy importante en el entendimiento del crecimiento y la apreciación de la geometría fractal. Las grandes estructuras de agregación de pequeñas partículas encontradas en sistemas tales como la contaminación y coloides han sido descritas por muchos años en términos tales como "tenues", "ramificadas", "filamentosas" y "esponjadas".

La figura 1.2.4.1 muestra una micrográfica de un agregado de partícula de hierro que fué estudiada por Forrest y Witten[3]. Porque este agregado es tan tenue, su proyección no cubre un plano. Esto significa que esencialmente la estructura puede ser vista en una proyección y sus propiedades de escalamiento fractal pueden ser completamente caracterizada, usando una imagen de la proyección vista en un micrógrafo electrónico. De esta manera Forrest y Witten pudieron demostrar que los agregados formados bajo estas condiciones pueden ser descritos en términos de la geometría fractal. Dichos agregados son propiamente fractales con una dimensión fractal de 1.7 a 1.9[4], esto constituyó una de las primeras aplicaciones de la geometría fractal en un experimento en física de materia condensada y ciencia de materiales, además de problemas en física de polímeros, los cuales fueron discutidos en términos diferentes pero con lenguajes cercanos. Este trabajo fué importante porque estimuló más tarde el desarrollo de la agregación limitada por difusión DLA, modelo de Witten y Sander[5], y fué la primera demostración de la naturaleza fractal de un agregado real.

Figura 1.2.4.1 Agregado de una partícula de hierro visto mediante un micrógrafo electrónico

Estructuras como la mostrada en la figura anterior se explican mejor en términos de modelos de agregación grupo-grupo, que en modelos de agregación partícula-grupo, tal como en el modelo DLA. La agregación grupo-grupo es uno de los más extensamente estudiados en los procesos que se dirigen a la formación de patrones.

1.2.5. Generador de números aleatorios

La formación de agregados de crecimiento aleatorio no es sino la acumulación de pequeñas partículas en movimiento que se integran en alguna posición de un espacio delimitado y forman grandes estructuras. Las técnicas de modelación computacional de estos fenómenos hacen especialmente adecuadas la utilización de generadores de números aleatorios, en la simulación de las caminatas aleatorias.

Para el modelo computacional de este trabajo de tesis, fué necesario implementar el uso del generador de números aleatorios *ran3*, con el fin de simular las caminatas, posiciones y colores que toman las partículas en cada experimento.

La definición de aleatoriedad en el contexto de secuencias generadas a través de una computadora, nos dice que un programa determinístico que produce una secuencia aleatoria debe ser diferente y sin correlaciones con el programa que use su salida. En otras palabras, sí dos generadores de números aleatorios producen estadísticamente los mismos resultados cuando se aplican a algún programa en particular, podremos decir entonces que al menos uno de ellos no es un buen generador.

Las características generales que debe cumplir un buen generador de números aleatorios son:

- Período amplio de generación. El período para repetir la misma secuencia debe ser muy grande
- Libre de correlaciones. Las secuencias generadas deben ser independientes, sin ninguna relación de recurrencia, o al menos aparentarlo, porque como ya se dijo, estos programas de generación de números aleatorios son estrictamente deterministicos.

Los sistemas que incluyen generación de números aleatorios, son casi siempre generadores de congruencia lineal (LCG), los cuales generan una secuencia de enteros I_1, I_2, . cada uno entre 0 y $m-1$, expresado como sigue;

$$I_{j+1} = (aI_j + c)^* \bmod m$$

Donde m es llamado *modulo*, y a y c son enteros positivos llamados *multiplicador* e *incremento* respectivamente. La recurrencia de la ecuación anterior se repetirá con un período que obviamente no será mayor que m. Veamos el siguiente ejemplo en la tabla 1.2.5.1

DATOS	$a = 3$	$a = 7$	$a = 7$	$a = 7$	$a = 7$	$a = 7$
	$c = 5$	$c = 5$	$c = 5$	$c = 5$	$c = 5$	$c = 5$
	$m = 11$	$m = 18$	$m = 18$	$m = 18$	$m = 18$	$m = 18$
I_{j+1}	E_1	E_2	E_3	E_4	E_5	E_6
0	5	12	19	23	29	36
1	9	17	12	4	10	5
2	10	16	17	15	3	4
3	2	9	16	2	8	15
4	0	14	9	1	7	2
5	5	13	14	12	0	1
6		6	13	17	5	12
7		11	6	16	4	17
8		10	11	9	15	16
9		3	10	14	2	9
10		8	3	13	1	14
11		7	8	6	12	13
12		0	7	11	17	6
13		5	0	10	16	11
14		4	5	3	9	10
15		15	4	8	14	3
16		2	15	7	13	8
17		1	2	0	6	7
18		12	1	5	11	0
19		17	12	4	10	5

Tabla 1.2.5.1 *Ejemplo simple de un generador de números aleatorios*

Debido al módulo aritmético usado en el LCG, la secuencia de números producidos será cíclica. La longitud del ciclo es conocida como periodo, por ejemplo en la columna E_1 de la tabla anterior, el periodo es igual a 5, observe que el número 5 se repite después de 5 iteraciones.

El periodo máximo para un LCG con módulo m, es m por sí mismo. Un LCG con un periodo máximo posible, se conoce como un LCG de ciclo completo. Un LCG de ciclo completo produce, cada ciclo, una permutación de los números entre 0 y $m-1$.

Se puede obtener un generador de ciclo completo para cualquier m, escogiendo los valores apropiados para a, c y x_0. Existen algunas reglas que se aplican cuando c es igual 0, para garantizar un ciclo completo.

En una secuencia de números aleatorios, cada número en la secuencia es totalmente independiente de los números que llegaron antes que él, de tal manera que no se puede predecir el número siguiente.

La técnica más popular para generar secuencias de números aleatorios, es el uso del LCG, por su facilidad de implementación, rapidez, y si se genera apropiadamente, es capaz de producir secuencias bastante aleatorias.

Si se usa el LCG para generar secuencias, los números obtenidos están lejos de ser independientes por su relación de recurrencia del LCG. Sin embargo para quienes no conocen esta relación fundamental, los números aparentan ser aleatorios.

En la columna E_2 de la tabla 1.2.5.1 del ejemplo anterior, se puede observar que con valores de

$m = 18$

$a = 7$

$x_0 = 12$

$c = 5$

La secuencia obtenida tiene un período de solo 18 números. La forma de obtener un periodo largo, es usando un valor grande para el módulo y seguir las reglas de la generación de un ciclo completo.

La utilización de un módulo bastante grande no garantiza una secuencia aleatoria, por ejemplo para $m = 2^{31} - 1 = 2,147,483,647$

$a = 1$

$c = 2$

$x_0 = 1$

se obtiene una secuencia de 1, 3, 5, 7, 9, 11, ...

Esta secuencia no es aleatoria, de hecho para cualquier valor de c, la secuencia producida tendrá a c como la diferencia entre dos números consecutivos.

Una solución para este tipo de problema es elegir el valor más alto posible para m, cuyo valor no sea primo o producto primo. Probablemente esto nos lleve a una secuencia más aceptable. Otra solución sería utilizar un generador multiplicativo puro, con $c = 0$ Entonces la relación de recurrencia se reduce a;

$$x_n = ax_{n-1} \bmod m$$

Utilizar un generador multiplicativo puro, involucra un cálculo menos (la suma). Debido a esto, los generadores multiplicativos puros son comúnmente usados en generadores de números aleatorios.

Existe el problema de que nadie puede garantizar la aleatoriedad. Lo que se debe hacer es intentar con varios valores para *a* y *m*, y probar las secuencias resultantes. Para un generador multiplicativo puro, es necesario un grupo diferente de reglas para asegurar un ciclo completo, pero lógicamente con matemáticas más complejas.

Los generadores multiplicativos puros han sido estudiados ampliamente para su uso como generadores de números aleatorios. A diferencia del LCG, se quiere un módulo primo. Para el módulo 2,147,483,647 hay más de 500 millones de valores para *a* que garantizan un ciclo completo, sin embargo muchos de ellos no son buenos generadores de números aleatorios.

Un grupo de parámetros que ha tenido un amplio uso es m = 2,147,483,647 y a = 16,807 . Un generador con este particular grupo de parámetros se le denomina Generador Estándar Mínimo. Un generador estándar mínimo fué ampliamente probado para su aleatoriedad y se considera bueno para producir secuencias aleatorias.

Una crítica del generador de estándar mínimo, así como de los generadores basados en LCG, es que fallan en una prueba denominada *Prueba Espectral.*

En general la *prueba espectral* busca las correlaciones entre los números producidos en un espacio de k dimensiones. Muchos generadores de números aleatorios tenderán a formar grupos de hiperplanos. Consideremos el siguiente ejemplo para el caso de k = 2. Aquí los números consecutivos se prueban para ver si existen correlaciones. Una forma gráfica para desarrollar esta prueba, es tomar los números aleatorios de la secuencia en pares, formando de esa manera coordenadas bidimensionales:

1, 4, 8, 2, 5, 9, 12, 43, 24, 7, 0, 1, ...

(1, 4), (8, 2), ...

Los puntos se trazan entonces en estas coordenadas. Este proceso se repite varias veces hasta obtener una gráfica llena de puntos.

Si los números son realmente aleatorios, no habrá ningún patrón notable en los puntos. Si hay correlaciones, se observan en la gráfica los puntos alineados, como se muestra en la fig. 1.2.5 1

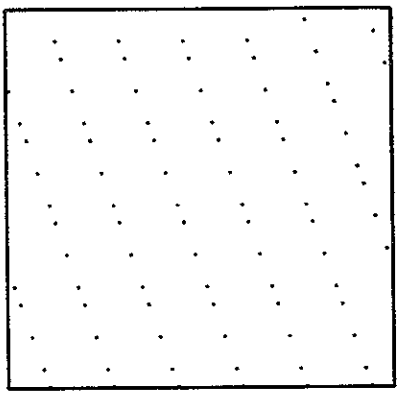


Fig.1.2.5.1 Gráfica espectral por el generador de estándar mínimo

Para tres dimensiones, es necesario considerar los tres primeros números de la secuencia como un punto de coordenadas (x, y, z), los siguientes tres números de la secuencia como otro punto, y así sucesivamente, graficándolos en un espacio tridimensional, verificando entonces que en cada plano del espacio no se formen patrones repetitivos en ninguno de los planos.

Probablemente la prueba espectral de números aleatorios no compruebe la aleatoriedad de una secuencia de números, pero existe una prueba más eficiente, la Entropía de Kolmogorov[6].

Básicamente la entropía de Kolmogorov estudia la relación entre cada número de la secuencia y el resto de números que forman parte de ésta. En este sentido, la entropía puede tener los siguientes valores:

$K = 0$

K = Un número positivo

$K = \infty$

Para el primer caso, se dice que la secuencia representa un sistema periódico.

Para el segundo caso, se dice que el sistema es caótico.

Para el tercer caso, se dice que la secuencia es aleatoria.

Desde luego, la gran cantidad de cálculos que deben hacerse para calcular la entropía de Kolmogorov en una secuencia de más de 60,000 números, requiere una gran cantidad de recursos de computadora.

Con el propósito de simular las caminatas, posiciones y colores que toman las partículas en cada experimento, fué necesario implementar el uso del generador de números aleatorios *ran3*.

El procedimiento para implementar la generación de números aleatorios en el programa, consistió en tres pasos fundamentales;

- Inicialización, proporcionando una semilla inicial aleatoria (a partir del reloj interno de la computadora).
- Generación de un número aleatorio, en este caso se utiliza el algoritmo de ran3, el cual nos devuelve un número de punto flotante sobre el rango de 0 a 1 sin correlaciones
- Destrucción del generador, con lo cual se permite liberar memoria cuando ya no se vaya a usar más.

Este método ran3 genera números aleatorios con distribución uniforme en el rango de [0.0 a 1.0][I].

Los tiempos de ejecución de diversos algoritmos de generación de números aleatorios, son inevitablemente dependientes de cada computadora. A pesar de eso, la tabla 1.2.5.2 es un indicativo de tiempos relativos entre diversos generadores de números aleatorios[II].

Generador	Tiempo de Ejecución Relativo
ran0	1.0
ran1	1.3
ran2	2.0
ran3	0.6
ranqd1	0.10
ranqd2	0 25
ran4	4 0

Tabla 1 2 5.2 Tiempos relativos entre generadores de números aleatorios

[I] Ver anexo 1, en donde se explica el algoritmo del generador de numeros aleatorios ran3
[II] Los valores mas pequeños en la tabla indican generadores mas rapidos

El generador ran1 se utiliza en propósitos generales, es portátil (puede ser ejecutado con pocas o ninguna modificación en un gran rango de sistemas de cómputo) y esta basado en el generador estándar mínimo de Park&Miller[7].

El ran2 se recomienda cuando se requieren generar mas de 100,000,000 de números aleatorios.

El ran3 ofrece una rutina portátil y muy rápida, recomendada en donde se requiera rapidez y amplios períodos.

El ran4 es extremadamente bueno, pero es lento.

La rapidez de generación de números aleatorios R de amplio periodo, uniformemente distribuido sobre un rango de $0 < R < 1$, sin correlaciones que pudieran influir en la salida de la simulación, es parte crucial de cualquier generador de números aleatorios.

1.2.6. Fractales y escalamiento

Los conceptos relacionados a la geometría fractal y escalamiento han probado ser recursos extremadamente valiosos en la descripción y entendimiento de un amplio rango de estructuras desordenadas y sus orígenes. Hay muchos tipos de fractales que pueden ser descritos cualitativa y cuantitativamente, en muchas formas.

Cualquiera que sea el método de aproximación al concepto de fractal que utilicemos, hay un concepto central, que es el de dimensión. Más precisamente, consideraremos varios conceptos de dimensión; y el primero de ellos, el de dimensión topológica.

En los "Elementos" de Euclides, ya se define, implícitamente y de forma inductiva, el concepto de dimensión. Se dice que una figura es unidimensional, si su frontera está compuesta de puntos; bidimensional, si su frontera está compuesta de curvas y tridimensional, si su frontera está compuesta de superficies.

La construcción de la dimensión topológica se puede basar en la idea de generalizar el concepto de que la dimensión de una bola es tres mientras que la dimensión de la esfera que la limita es dos: dimensión de un conjunto X a partir de la dimensión de su frontera dX.

Por otra parte, un objeto fractal es, ante todo, un subconjunto de R^n. En este contexto, preferimos una definición equivalente de dimensión topológica basada en la dimensión de recubrimiento, concepto que juega un papel importante en la definición de dimensión fractal.

Consideremos un subconjunto S de R^n

Un recubrimiento abierto de S es cualquier colección de conjuntos abiertos a cuya reunión contiene al conjunto S, como se muestra en la figura 1.2.6.1

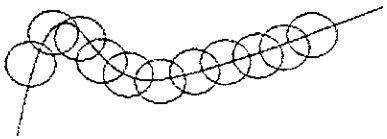


Figura 1.2.6.1 Dimensión topológica 1

Un refinamiento abierto α' del recubrimiento abierto α es otro recubrimiento tal que cada abierto $A' \varepsilon \alpha'$, está incluido en algún abierto $A \varepsilon \alpha$.

En algún sentido, un refinamiento abierto α' de S, proporciona un recubrimiento "más detallado" de S que α.

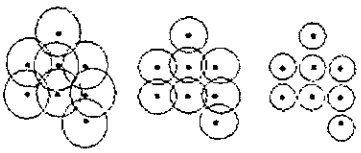


Figura 1 2 6 2 Dimensión Topológica 0

Decimos que α es un recubrimiento abierto de orden k del conjunto S, si, cualquiera que sea $x \varepsilon S$, x pertenece a un máximo de k abiertos del recubrimiento α.

El conjunto S tiene dimensión de recubrimiento (dimensión topológica) n, si cualquier recubrimiento abierto a de S admite un refinamiento abierto de orden $n+1$, pero no de orden n.

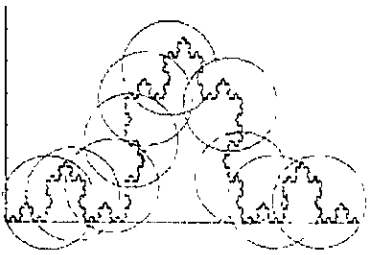


Figura 1 2 6 3 Dimensión topológica 1

Uno de los procedimientos para caracterizar e incluso para clasificar los objetos fractales consiste en atribuir a cada uno de ellos una cantidad numérica, la dimensión fractal.

Para realizar esta atribución, la cantidad numérica correspondiente, debe satisfacer algunas propiedades que contribuyen a dotar de credibilidad al citado parámetro.

Los requerimientos apuntados por Yamaguti y otros en "Mathematics of Fractals"[8], referidos a subconjuntos de R^n, y considerados como una relación de "mínimos" a satisfacer por cualquier concepto de dimensión, son los siguientes:

a) Para conjuntos constituidos por un solo elemento {a}, la dimensión deberá ser 0. Para el intervalo unidad [0,1], el valor deberá ser 1.

b) Carácter monótono: Si $X \subset Y$, la dimensión de X deberá ser menor o igual que la dimensión de Y.

c) Estabilidad numerable: Sea X_j una sucesión de conjuntos cerrados de R^n. Entonces,

$$\dim\left(\bigcup_{j=1}^{\infty} X_j\right) = \sup_{j\geq 1} \dim\left(X_j\right)$$

d) Invariancia: Para alguna clase de aplicaciones ψ de R^n en R^n (homeomorfismos para la dimensión topológica)

$$\dim(\psi(X)) = \dim(X)$$

Una de las interpretaciones de la dimensión, posiblemente la más natural, está relacionada con la capacidad de los objetos para ocupar el espacio euclidiano en el que se encuentran sumergidos. Dicho de otra forma, la dimensión ayudará en la determinación del contenido o medida de un conjunto, en particular de los conjuntos fractales.

Así, cuantificar fractales es definir, por algún procedimiento, la proporción del espacio físico en el que se inscriben y que es llenado por ellos.

Establecemos distintos conceptos de dimensión que, por otra parte, hacen uso de algunos objetos euclidianos (conjuntos abiertos, cerrados, segmentos, bolas, etc.) y los pasos a límite correspondientes.

Encontraremos una diferencia fundamental con los objetos euclidianos:

Teniendo en cuenta que se cumple la propiedad de que la dimensión fractal de F es mayor que su dimensión topológica[9], establecida al tratar el concepto de objeto fractal

Los valores obtenidos serán racionales o irracionales (no enteros).

Habitualmente, para cuantificar conjuntos buscamos "bloques unidad" con cuales compararlos. La correspondiente suma proporciona la longitud, área o volumen del conjunto considerado. Esta técnica funciona con objetos euclidianos, incluyendo eventualmente procesos de paso a límite.

Para objetos fractales no disponemos de bloques unidad semejantes.

Por ejemplo, si magnificamos sucesivamente un objeto euclidiano "unidimensional", encontramos segmentos rectilíneos. Sin embargo, si magnificamos sucesivamente un objeto fractal, encontramos objetos con niveles de complicación comparables a los del conjunto de partida.

1.2.7. Fractales auto-semejantes

Uno de los principales fractales estudiados desde el punto de vista matemático fué el conjunto Cantor, ilustrado en la Figura 1.2.7.1. El *conjunto Cantor*[10] puede ser construido, primero removiendo la tercera parte central de un segmento de línea que cubre el intervalo de *0* a *1*. En la siguiente etapa del proceso de construcción, son removidas las terceras partes centrales de los dos segmentos de líneas restantes. Durante cada etapa siguiente, la tercera parte central es removida para cada uno de los segmentos de línea restante, de tal forma que después de *n* etapas, el número de segmentos de líneas han crecido a 2^n y su longitud total se ha reducido a $(2/3)^n$. En el limite $n \rightarrow \infty$, un conjunto fractal será generado. Este "objeto", con medida cero (longitud total de los segmentos de líneas restantes) y un número infinito de piezas, se consideró una paradoja, sin relevancia alguna en el mundo real.

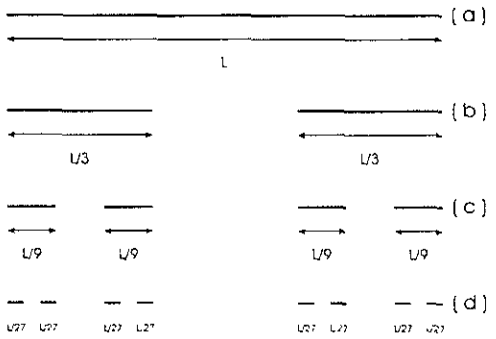


Fig 1 2 7 1 Tres etapas en la construcción de un conjunto Cantor (a) muestra el segmento de linea original, (b), (c) y (d) muestran los prefractales después de las primeras tres etapas en la construccion del proceso

El *conjunto Cantor* posee algunas propiedades comunes a muchos otros fractales Contiene huecos en todas sus escalas de longitud y llena una fraccion despreciable del espacio que ocupa Si el

conjunto Cantor es dilatado por un factor de 3^m, puede ser cubierto por un mínimo de 2^m réplicas de si mismo. Estructuras con esta propiedad dicen ser de *escala invariante*, similar propia o que tienen simetría en su dilatación. En general, sí M^m es el número mínimo de réplicas del patrón original que son requeridas para cubrirse a sí mismo después de la dilatación (cambio de escala de longitud) mediante un factor de N^m, sus propiedades de escalamiento geométrico pueden ser caracterizado por un fractal con dimensionalidad D dado por

$$D = \log(M)/\log(N)$$

Para objetos Euclidianos, tales como líneas continuas, planos rectangulares o cubos, la anterior ecuación proporciona una dimensionalidad Euclidiana de 1, 2 y 3 respectivamente. Para el *conjunto Cantor* la dimensionalidad fractal es *log(2)/log(3)* o *0.6309* . Este resultado *(0 < D < 1)* es intuitivamente razonable. El *conjunto Cantor* es mas que un punto dimensional *0* (contiene un número infinito de ellos) pero menos que una línea dimensional *1* (la longitud total ocupada es cero). En este caso, la dimensionalidad fractal puede ser considerada como una medida de cuanto efectivamente el *conjunto Cantor* llena el espacio dimensional *1* en el cual reside.

El *triangulo de Sierpinski*, mostrado en la figura 1.2.7.2 es otro ejemplo simple de un fractal auto similar. El patrón puede ser construido encajando juntos tres triángulos equiláteros idénticos, con lados de longitud ε, para formar un nuevo triangulo con lados de longitud 2ε y con un hueco triangular en la parte media. En la siguiente etapa del proceso de construcción jerárquico, tres de los triángulos con lados de longitud 2ε, con huecos triangulares en sus partes medias, son colocados juntos para formar un triángulo con lados de longitud 4ε, con un hueco triangular de tamaño 2ε y tres huecos triangulares de tamaño ε. Después de n generaciones, un patrón triangular con lados de longitud $2^n\varepsilon$ conteniendo 3^n triángulos sólidos, cada uno con lados de longitud ε, son formados. Cada vez que la escala de longitud es incrementada mediante un factor de *2*, el área o medición se incrementa por un factor de *3*, y la dimensionalidad fractal esta dada por $D = log\ 3/\ log\ 2$. Muy a menudo, solamente los bordes de los triángulos de tamaño ε son usados para formar un marco o enrejado en el *triangulo de Sierpinski* que también tiene una dimensionalidad fractal de $D = log\ 3\ /\ log\ 2$.

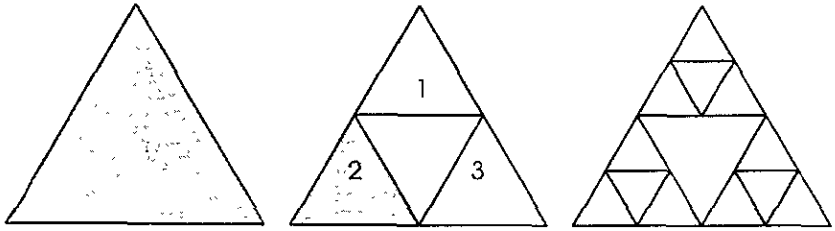


Fig.1.2 7.2 Estas figuras muestran los primeros dos estados en la construcción del triangulo de Sierpinski (D = log 3/ log 2)

Este proceso de continuar encajando triángulos equiláteros idénticos puede continuar hasta llegar a formar estructuras más complejas, como la mostrada en la figura 1.2.7.3, que corresponde a este mismo *triangulo de Sierpinski* en su 8ª generación

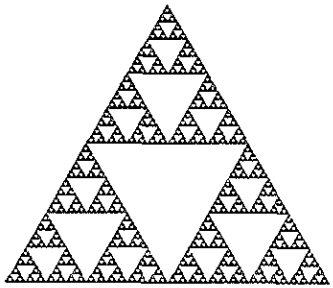


Fig 1.2 7.3 Triangulo de Sierpinski en su 8ª generación

Un procedimiento alternativo será iniciar con un triángulo con lados de longitud $L = 1$ y remover triángulos con longitud de $L/2$ desde su parte media, para generar un patrón de tres triángulos con lados de longitud $L/2$. Este procedimiento será repetido para todos los triángulos con lados de longitud $L/2$ y para todos los sub triángulos durante cada etapa del proceso de generación. Después de n etapas, los lados de los sub-triángulos tendrán longitudes de $l = L/2^n$.

El proceso de construcción del prefractal en el *triangulo de Sierpinski* desde pequeños prefractales parece un proceso de agregación sistemático. Consecuentemente, esta aproximación es útil para ilustrar los conceptos de geometría fractal en el contexto de problemas tales como agregación.

1.2.8. Modelos de Crecimiento

El desarrollo de modelos extremadamente simples ha proporcionado mucho de la motivación para el estudio sistemático de crecimiento desordenado usando aproximaciones teóricas y experimentales. Es el caso del *modelo Eden*[11], estos modelos ahora están totalmente bien entendidos

desde el punto de vista teórico. En otros casos, tal como en el modelo DLA, están totalmente alejadas de un entendimiento satisfactorio, y estos modelos plantean una discusión teórica importante.

Mucha atención se enfoca sobre el modelo *DLA*. Este modelo puede ser usado como base para entender un amplio rango de fenómenos de formación de patrones. Este modelo se ha estudiado extensamente durante los últimos 15 años y juega un papel muy importante en el estudio de formación de fractales. A parte de su amplia aplicación, importancia teórica y atractivo estético, el modelo DLA proporciona un excelente ejemplo de progreso llevado a cabo en las discusiones que permanecen en el *estudio de procesos de crecimiento alejado del equilibrio*. La *sensibilidad* a cambios en la geometría fractal de este modelo partiendo de sus fluctuaciones iniciales, la enumeración exacta de las formas y cantidad de partículas que integran la estructura fractal durante su crecimiento inicial, proporciona un reto de estudio importante.

Bibliografía

- Paul Meakin. *Fractals, scaling and growth far from equilibrium*, Cambridge University Press 1998
- Benoit B. Mandelbrot. *The fractal geometry of nature*, W. H. Freeman 1983
- McCauley, Joseph L. *Chaos, Dynamics, and Fractals*, Cambridge Nonlinear Science Series 1994
- R. C. Hilborn. *Chaos and Nonlinear Dynamics: An Introduction for Scientists and Engineers*, Oxford U. Press, New York, 1994
- Mikail B. Smirnov. *Clusters and Small Particles In Gases and Plasmas*, Graduate texts in contemporary physics, 2000
- Janos Kertesz, Imre Kondor. *Advances in Computer Simulation*, Technical University Budapest, Hungary 1998

2. Modelo Bicolor de Agregación Limitada por difusión (bicolor DLA)

2.1. Descripción del Modelo bicolor DLA

A partir del modelo original de Witten y Sander[12] que como ya se dijo fué la primera demostración de la naturaleza fractal de un agregado real, diversos investigadores han propuesto modificaciones y nuevos modelos ajustándolos a diferentes áreas del conocimiento.

Una amplia variedad de fenómenos naturales pueden considerarse como fenómenos cuyo crecimiento se ajusta a lo que conocemos como DLA; la coagulación de aerosoles, algunos crecimientos de cristales, flujo de fluidos en medios porosos, movimientos intermedios entre líquidos con viscosidad diferente, etc.[13]

Debido a la flexibilidad que presenta el modelo de Witten y Sander, el fenómeno DLA se ha expandido y diferentes investigadores han propuesto novedosos modelos cuyo objetivo ha sido estudiar el comportamiento con o sin enrejado, la dimensión fractal y también la simetría de las estructuras obtenidas experimentalmente[14]. La totalidad de los resultados han coincidido en que el estudio del fenómeno DLA no es sencillo y menos lo es si se pretende formular un modelo teórico. Por lo tanto es mucho más sencillo y rentable intentar formular un modelo computacional que permita obtener información importante del fenómeno.

Recientemente los investigadores Sergey Nechaev, Vladimir Tchijov y Suemi Rodríguez Romo propusieron el *modelo Bicolor DLA*[15], basados en el modelo original de Witten y Sander. Los resultados experimentales de sus simulaciones, pusieron especial atención a la distancia que separaban a dos semillas iniciales del fenómeno en estudio.

El *modelo bicolor DLA* es un desarrollo del modelo original de Witten y Sander e inicia con dos partículas (semillas) de diferente color situadas a una distancia d una de la otra, como se muestra en la figura 2.1.1. Una partícula se crea a una gran distancia de las dos partículas originales de diferentes colores. Esta partícula se genera con una probabilidad p de ser del color de una de las partículas utilizadas como semillas y una probabilidad de $1-p$ de ser del color de la otra semilla. La *partícula empieza a caminar en forma aleatoria, es decir, llevar* a cabo un movimiento de caminata al azar, hasta que una de las siguientes cosas ocurren:

1. Llegando a límites predeterminados del enrejado, es eliminada y una nueva partícula se genera en algún otro lugar aleatorio del enrejado. Los límites son arbitrarios y dependen del tamaño y forma de la estructura fractal que se va formando.

2. En su caminata la partícula ocupa un lugar adyacente a la semilla de color diferente, por lo cual esta partícula es eliminada y se genera una nueva para repetir la secuencia.

3. En su camino aleatorio, la partícula ocupa un lugar adyacente a la semilla de su mismo color, entonces esta partícula se adhiere a la semilla dando con esto origen a una estructura cuya forma será también aleatoria, y genera otra partícula para continuar con el proceso.

Este proceso se repite hasta que alguna de las dos estructuras de diferente color alcanza un tamaño considerablemente grande que incite al análisis de estos resultados.

Las variables de este modelo son; la distancia d entre las dos semillas y la probabilidad p para la determinación del color de una partícula.

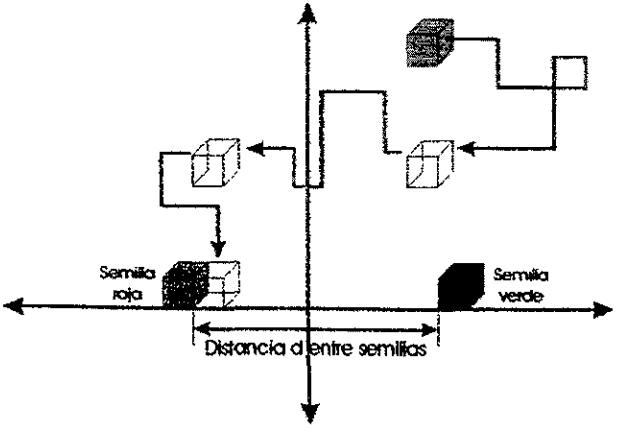


Fig. 2.1.1 Modelo Bicolor DLA

Los vértices inferior derecho de la semilla roja y el vértice inferior izquierdo de la semilla verde, están a la misma distancia del origen cartesiano como se muestra en la figura 2.1.1.

Las partículas de dos colores diferentes se suponen colocadas dentro de un enrejado a una distancia d una de otra.

A una gran distancia de las semillas (también partículas), en un punto aleatorio sobre un enrejado circular de radio R_d, una nueva partícula es introducida con probabilidad p de ser de color rojo y una probabilidad 1-p de ser de color verde. El radio R_d es mucho más grande que la distancia que separa a las semillas, $R_d >> d$. De hecho, es uno de los parámetros implícitos del modelo computacional, observe la figura 2.1.2.en la cual se muestra el radio circular R_d del enrejado.

Como ya se mencionó, los variables del *modelo Bicolor DLA* son la distancia *d* entre semillas y la probabilidad *p* de que la partícula caminante sea de un color u otro. En el caso del programa de *enumeración exacta aplicado al modelo bicolor DLA*, se restringen los valores de estudio a *p* entre 0 y 0.5 por simetría, y valores de *d* iguales a 2, 4, 6, 8 y 10 unidades de enrejado. En este rango de distancias se presentan fenómenos que nos interesan estudiar.

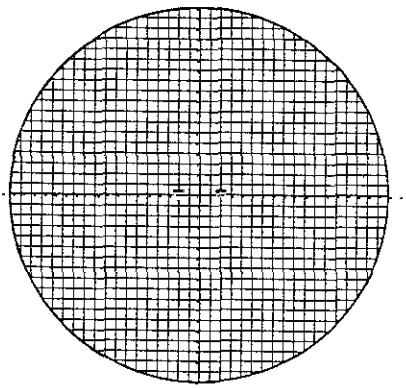


figura 2 1 2 Enrejado de radio circular del modelo Bicolor DLA

Las siguientes figuras muestran diversas estructuras obtenidas para los modelos DLA y Bicolor DLA

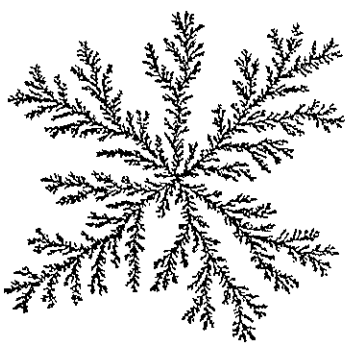


Figura 2 1 3 DLA con cien millones de partículas

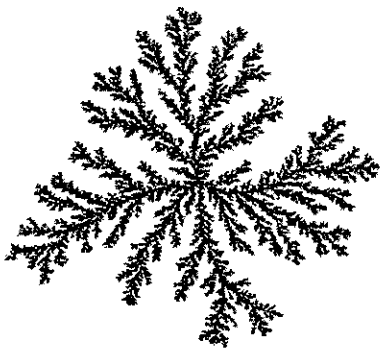


Figura 2.1.4 DLA con un millón de partículas

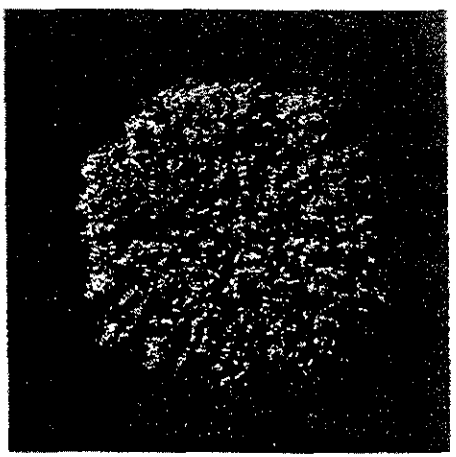
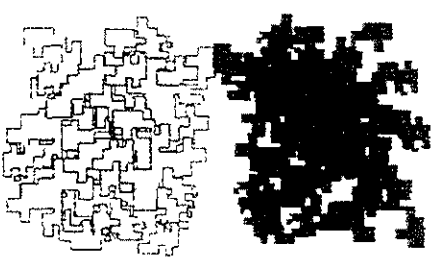


Figura 2.1.5 DLA en tres dimensiones



Con p= 0 5 y d=100, las dos estructuras son básica mente simétricas

Figura 2 1 6

Figura 2.1.7

2.2. Simulación computacional del modelo Bicolor DLA

Una muy buena herramienta para la simulación de procesos, actualmente es la computadora personal. Mediante programas específicos desarrollados con algoritmos suficientemente rápidos, eficientes, que generen resultados confiables y cuyo tiempo de cálculo sea relativamente bajo, se realizan simulaciones de procesos muy complejos.

Para la simulación del modelo Bicolor DLA se utilizó un programa escrito en Borland C++, el cual ha sido ejecutado en computadoras personales con plataformas integradas (Linux y Windows), la mayoría de las ejecuciones se realizaron en Windows con un procesador Intel Pentium II a 450 Mhz, donde se consideraron principalmente probabilidades p con valores entre *0* y *0.5*, y distancia d entre semillas *2, 4, 6, 8 y 10* unidades de enrejado.

Los primeros experimentos se realizaron para cada valor de p y d mencionados anteriormente, con la condición de terminar cada experimento en el momento que alguna de las estructuras fractales alcanzará un número máximo de partículas de *5000*.

Algunos otros experimentos se repitieron en computadoras Pentium similares, con el sistema operativo Linux, observándose básicamente los mismos resultados (datos y desempeño).

Con el análisis de resultados obtenido a través de este programa se observó formación de estructuras en algunos experimentos cuya asimetría rompía el esquema tradicional de muchos otros experimentos con las mismas condiciones de probabilidades p y distancias d entre semillas, esta fue una de las principales razones que motivan la necesidad de rediseñar este programa para cuantificar con exactitud el crecimiento inicial de las estructuras fractales obtenidas y el impacto de estos resultados en la formación geométrica de dichas estructuras[III].

Cuantificar con exactitud los orígenes de las estructuras fractales en el modelo Bicolor DLA, significa generar resultados que arrojen datos básicamente importantes al inicio de las estructuras fractales y que sirvan para definir el comportamiento macroscópico de dichas estructuras.

[III] En el capítulo 5 comprenderemos y analizaremos los resultados que dan origen al rediseño del programa

Bibliografía

- Paul Meakin. *Fractals, scaling and growth far from equilibrium*, Cambridge University Press 1998
- Benoit B. Mandelbrot. *The fractal geometry of nature*, W. H. Freeman 1983
- Keblinski, P., Maritan, A ., Toigo, F., Banavar, J.R., *Continuum Approach to Diffusion-Limited-Aggregation Type of Growth*, Physical Review E, The American Physical Society, College Park, MD, Vol. 49(6), pp. 4795-4798, June, 1994.
- Vaario, J., *Structural Formation by Enhanced Diffusion Limited Aggregation Model*, Artificial Life V, Nara, May 14-16, 1996
- Meakin , P., *The diffusion-limited aggregation model and geological pattern formation In: Growth and dissolution in Geo-systems*. (eds. Jamtveit, B., and Meakin, P.) Kluwer Academic, 1999
- Masahiro Nakagawa, Hajime Namikata. *A Diffusion-Limited Aggregation Model with Elongated Particles*, Journal of the Physical Society of Japan, 1992
- Dietrich Stauffer, Germany H. Eugene Stanley, *From Newton to Mandelbrot*, University of Cologne, Germany, 1995
- Zhuming Ai, Guipeng Luo, and Yu Wei. *A diffusion-limited aggregation model in time variant non-uniform field*. Journal of Southeast University, 24(5):110-112, 1994

3. Tablas Hash

3.1. Algoritmo de tablas Hash

Un problema de gran importancia en la simulación computacional de los modelos de agregación limitada por difusión es una verificación rápida para saber sí una partícula viajera se acercó a la estructura ya existente o todavía esta lejos de ella. La técnica más eficiente de dicha verificación es el método de las tablas hash.

La tabla hash es una estructura de datos que nos permite realizar inserciones, eliminaciones y búsquedas en un tiempo promedio constante, bajo ciertas hipótesis. Sin embargo, con un diseño cuidadoso, es posible hacer que sea arbitrariamente pequeña la probabilidad de que el hashing (o dispersión) demande más de un tiempo constante para cada operación[16].

La idea del hashing es ejecutar alguna operación sobre el campo clave (el campo clave nos permite identificar de manera única los diferentes elementos) y obtener el valor que es la dirección (o índice) del elemento al que pertenece la clave. La técnica de búsqueda hashing intenta localizar el elemento deseado después de un intento. El número de intentos necesarios para localizar una entrada, depende de factores tales como el algoritmo de hashing, los datos a buscar, el tamaño de la tabla, así como otras consideraciones. Se aplica igual para buscar elementos en memoria principal o auxiliar. El hashing se usa generalmente si el rango de las claves es grande en comparación con el número real de elementos.

La estructura de datos ideal para la tabla hash[IV] es simplemente un arreglo de tamaño fijo. Cada elemento de la tabla consiste en la clave y la información asociada a ésta.

Cada elemento del arreglo se le llama bucket o celdas, y al proceso de implementación se le conoce como hashing. El hashing es una técnica utilizada para la construcción de tablas para búsquedas, inserciones y borrado altamente eficiente. Un buen diseño de una tabla hash, encuentra que una entrada puede tomar solo una o más comparaciones, sin importar el número de entradas presentes.

Cuando dos elementos señalan al mismo bucket sucede un fenómeno llamado colisión. Los métodos para el manejo de colisiones se conocen como estrategias de resolución de colisiones.

[IV] Se trata de un vector de tamaño fijo en el que se van a guardar los elementos Existe una funcion hash que convierte la clave en una posición en el vector. Así se consigue un acceso con retardo casi constante. Tienen el problema de la gestión de colisiones. Una colisión se produce cuando la función hash devuelve el mismo valor para distintas claves

3.2. Hashing, muestreo aleatorio invertido

Las funciones para el mapeo en la tabla se llaman funciones hash. Una función hash debe tomar una clave, que proviene de un gran rango de valores, e introducirlo dentro del menor rango de buckets. *Cuando se mapea de muchos a pocos, existen muchas colisiones*, para minimizar las colisiones se mezclan las claves en una forma que la salida de la función hash sea esencialmente aleatoria. Funciones con estas propiedades son conocidas como funciones hash uniformes. Idealmente, se quiere que cada bucket tenga $1/m$ oportunidad de ser seleccionada, dando un grupo aleatorio de claves, como en el caso de las coordenadas de las posiciones de las partículas adheridas, y donde m es el número de partículas en la tabla.

Es posible tener malas funciones hash, como el caso en donde todas las claves señalan al mismo bucket.

El hashing esta relacionado a un muestreo aleatorio. Recordando que en una muestra aleatoria, las k selecciones se eligen aleatoriamente de un grupo de r elementos. Supongamos que asignamos un número único en el rango de 0 a $m-1$ a cada muestra en un grupo de m muestras aleatorias únicas. Estos números son equivalentes a índices de buckets y las selecciones en cada muestra son las entradas cuyas claves mapean al mismo bucket.

Con hashing se tienen muchas muestras, escogidas a priori, y debemos determinar de entre el grupo de muestras a cual muestra pertenece una clave dada.

3.3. Funciones Hash

Una función hash debe satisfacer dos requerimientos, que usualmente están en competencia. Es deseable que la función hash sea rápida, tomando solo un poco de cálculos. Además la función debe ser bastante hábil para minimizar colisiones.

La idea general es usar la clave para determinar la dirección del registro. Para ello hay que encontrar una aplicación adecuada H del conjunto K de claves sobre el conjunto D de direcciones

$$H : K \rightarrow D$$

Los dos criterios principales al seleccionar una función hash H son los siguientes:

- H deberá producir *tan pocas colisiones como sea posible*. Es decir, en la medida de lo posible, deberá distribuir uniformemente las direcciones hash sobre todo el conjunto D. Por supuesto, a menos que las claves se conozcan con anterioridad, no se puede determinar si una función hash las dispersa de manera adecuada. Sin embargo, aunque es raro conocer las claves antes de seleccionar una función de dispersión, es bastante común conocer algunas

propiedades de las claves. Por ejemplo, si todas las claves terminan en los mismos tres dígitos y el tamaño de la tabla es 1000, si utilizamos el método de división, entonces la función hash nos producirá el mismo valor para todas las claves.

- *H* debe ser fácil y rápida de calcular. Aunque la función hash proporcione direcciones únicas, no es buena, si lleva más tiempo calcular la dirección que buscar directamente de la tabla.

3.4. Funciones hash más utilizadas

A continuación veremos algunas de las funciones hash más utilizadas.

- Método de división. Una de las transformaciones más antiguas y sencillas, es la de dividir por el número de direcciones posibles. La transformación de un número a su resto después de dividir por un número fijo, es la idea central en la aritmética modular. La función hash *H* se define por cualquiera de las siguientes dos consideraciones:

$$H(k) = k \bmod m$$
$$H(k) = (k \bmod m) + 1$$

La segunda fórmula se usa cuando queremos que las direcciones hash vayan de *1* a *m* en vez de desde *0* a *m-1*.

Una buena idea es que el tamaño de la tabla sea algo más grande que el número de elementos que se desean insertar. Cuanto mayor sea el tamaño de la tabla, mayor es el rango de la función hash, y por lo tanto, disminuye la probabilidad de que se produzcan colisiones. Por supuesto, esto conlleva un compromiso entre espacio y tiempo. Dejar posiciones en blanco es ineficiente en términos de espacio, pero reduce el número de colisiones, y por lo tanto, es más eficiente en términos de tiempo.

Es crucial que se escoja un buen valor de *m*, o los resultados no serán buenos. Si se hace *m* un número par, entonces las claves pares se introducirán en los buckets pares y las claves impares en buckets impares. De esa manera, si todas las claves fueran impares, la mitad de los buckets no se utilizarán y se esperarían más colisiones.

Los mejores resultados con el método de división se producen cuando el tamaño de la tabla es primo, ya que así, en general, la función hash se dispersa de manera más uniforme sobre todas las posibles posiciones de la tabla y el número de colisiones se minimiza.

La ventaja de este método es su simplicidad, aunque para determinados conjuntos de claves se debería usar otra función de hash. Un ejemplo de esto sería aquel en el que las claves terminan con los mismos 3 dígitos y el tamaño de la tabla es 1000.

- Método de plegado. La clave k se divide en varias partes k_1, k_2,..., k_r. Donde cada parte, excepto posiblemente la última, tiene el mismo número de dígitos que la dirección requerida. Entonces, a cada una de las partes se la aplica el operador "*OR exclusivo*" para formar la dirección (El resultado del operador *OR exclusivo* de 2 bits es 1 si los bits son diferentes y 0 si son iguales, Equivale a la suma binaria de bits si se ignora el acarreo). La función hash quedaría como:

$$H(k) = k_1 + k_2 + \cdots + k_r$$

donde se ignoran los dígitos más significativos que se den por el acarreo. A veces a las partes pares, k_2, k_4, ..., se las invierte antes de sumarlas, con el fin de conseguir valores más aleatorios y únicos.

El problema del método del plegado de bits es cuando las claves contienen los mismos grupos de bits pero en orden diferente. Estas claves se dispersan en el mismo valor.

Este método se combina frecuentemente con otros métodos hash

- Método de análisis de dígitos. Sólo se seleccionan los dígitos que tienen la distribución más uniforme. Los que no son tan uniformes (homogéneos) se borran de la clave, hasta que se obtiene el número deseado de dígitos. Este método implica un conocimiento de las claves. Un cambio en el conjunto de claves implica un nuevo análisis de los dígitos, por tanto este método no se ajusta a las exigencias de rapidez de inserción y posibilidad de borrado.

Si las claves no son enteros, se tienen que convertir a enteros antes de aplicar una de las funciones hash anteriores

3.5. Hashing, cadenas de caracteres

Es muy común que las claves sean cadenas de caracteres. Algunos ejemplos son nombres, domicilios, o identificadores.

Para el caso de cadena de caracteres, hay varias maneras de hacerlo:

1. Se puede utilizar la representación interna con bits de cada carácter interpretado como número binario. Una desventaja de esto, es que las representaciones con bits de todas las letras o dígitos tienden a ser muy similares en la mayoría de las computadoras.
2. Otra función hash polinómica con base en la regla de Horner.

$$A0 + X(A1 + X(A2 + ... + X(AN)...))$$

Para estos casos se tiene una posibilidad; sumar los valores enteros de cada carácter en la cadena y usar el resultado como el índice del bucket.

Debido a que es posible que el índice resultante sea más grande que el número de buckets, es común tomarle al resultado el módulo *m*.

En otras palabras, una vez que una cadena de caracteres es convertida a una forma numérica, se usa el método de la división tanto para aplicar más tarde hash al resultado como para mantener el valor en el rango. Como en el caso anterior, lo mejor es si *m* es un número primo.

3.6. Funciones hash perfectas

Dado un conjunto de claves $k = k_1, k_2, ..., k_m$ una función hash perfecta es aquella función de dispersión h, tal que;

$$h(k_i) \neq h(k_j)$$

Para todo $i \neq j$. Es decir no se producen colisiones[17]

En general, es difícil encontrar una función hash perfecta para un conjunto particular de claves. Además, una vez que se añaden unas cuantas claves más al conjunto para el cual se había encontrado una función hash perfecta, ésta deja de serlo en general para el nuevo conjunto. Así, aunque es deseable encontrar una función hash perfecta para asegurar la recuperación inmediata, esto no es práctico a no ser que el conjunto de claves sea estático y se busque en él con frecuencia. El ejemplo más claro de ésto, es un compilador. El conjunto de palabras reservadas del lenguaje de programación que se está compilando no cambia y se tiene que acceder muchas veces a este conjunto. Por lo tanto, una función hash perfecta permitiría al compilador determinar rápidamente si una palabra es reservada o no.

Cuanto más grande sea la tabla hash más difícil será encontrar una función hash perfecta para un conjunto de claves dado.

En general es deseable tener una función hash perfecta para un conjunto de *n* claves en una tabla de sólo *n* posiciones. A este tipo de función de dispersión perfecta se le llama mínima.

A continuación veremos algunos ejemplos de funciones de dispersión perfectas

a) Funciones hash perfectas por reducción al cociente. Son de la forma:

$$H(Key)=\left\lfloor \frac{Key+s}{d} \right\rfloor$$

donde $\lfloor\ \rfloor$ significa "truncar al entero más próximo inferior", y s y d son enteros.

b) Funciones hash perfectas por reducción de residuo. Son de la forma:

$$H(Key)=\left\lfloor \frac{(r+s*Key)\bmod x}{d} \right\rfloor$$

y un algoritmo para calcular los valores enteros r, s, x y d.

Estas dos funciones fueron desarrolladas por *Sprugnoli*[18].

c) Otro método para funciones hashing perfectas nos lo ofrece Cichelli[19]. Su fórmula es:

h(k) = longitud de k + valor asociado del primer carácter de k + valor asociado del último carácter de k

donde k es la clave. Para un conjunto de claves en particular, se deben calcular los valores asociados para los caracteres.

3.7. Clases universales de funciones hash

Como hemos visto es difícil obtener una función hash perfecta para un conjunto de claves. Tampoco es posible garantizar que una función hash específica, minimice colisiones sin conocer el conjunto preciso de claves que deben ser dispersadas.

Carter y Wegman[20] introdujeron el concepto de clase universal de funciones de dispersión. Tal clase consta de un conjunto de funciones de dispersión *h(key)*. Aunque una función individual en la clase puede funcionar mediocremente en un conjunto particular de claves de entrada, hay suficientes funciones que trabajan bien para cualquier conjunto de entrada aleatorio que si se elige una función de manera aleatoria de la clase, es probable que ésta ejecute bien la dispersión sobre cualquier conjunto de entrada que se le presente[21].

3.8. Resolución de colisiones

Puesto que el objetivo del hashing es el comprimir un posible espacio de claves grande en uno pequeño, es inevitable que ocurran colisiones, o sea que dos o más claves se transformen en la misma dirección.

A los registros que tienen la misma dirección se les denomina *sinónimos*.

El procedimiento particular para la resolución de colisiones que se escoja dependerá de muchos factores. Uno de los más importantes es el factor de carga *(λ)*. Este se define como la relación entre el número de elementos en la tabla de dispersión *(n)* y el tamaño de ésta *(m)*.

$$\lambda = n/m$$

Si el factor de carga es bajo, significa que hay un gran número de espacio libre, pero significa también que el tiempo de recuperación será bueno. Los factores de carga grandes, requieren un número mayor de accesos para recuperar un registro, pero ahorran memoria. A la hora de decidir cuál debe ser el factor de carga, tendremos que tener en cuenta el costo de almacenamiento frente a la frecuencia de recuperaciones.

Los procedimientos para la resolución de colisiones se pueden clasificar en dos tipos:

a) Dispersión abierta, externa o encadenamiento por separado

b) Dispersión cerrada, interna o direccionamiento abierto

La diferencia entre ambos es que, mientras en la dispersión abierta las colisiones se almacenan fuera de la tabla de dispersión, en la dispersión cerrada se almacenan dentro

3.8.1. Dispersión abierta

La dispersión abierta consiste en tener una lista de todos los elementos que se dispersan en el mismo valor. Si se desea tener *B* listas, numeradas de *0* a *B - 1*, se usa una función de dispersión *h* tal que para la clave *x*, *h(x)* de un valor entre *0* y *B - 1*. Lógicamente, el valor *h(x)* es la lista a la cual *x* pertenece. A las "listas" se las llama buckets y a *h(x)* valor de dispersión.

En un vector llamado tabla de buckets, indexado por los números de bucket *0, 1, B-1*, se tienen los encabezamientos de *B* listas (si hay poco espacio podría ser preferible evitar el uso de las cabeceras). Los elementos de la *i-ésima* lista son los que pertenecen a la clase *i*, es decir, aquellos elementos *x* tales que $h(x) = i$.

Además de utilizar listas enlazadas, se podría usar cualquier otro esquema para resolver colisiones como: un árbol binario de búsqueda u otra tabla de dispersión. Pero esperamos que si la tabla es grande y la función de dispersión correcta, todas las listas deben ser cortas, por lo que no vale la pena intentar nada complicado.

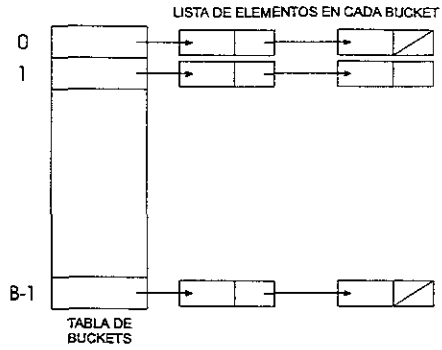


Figura 3 8.1.1 Dispersión abierta o encadenamiento por separado

Si hay *B* listas y *N* elementos almacenados en la tabla hash, las listas tienen en promedio *N/B* elementos, ya que los elementos se distribuyen uniformemente sobre las *B* listas.

Como ya hemos visto anteriormente, *N/B* es lo que hemos definido como factor de carga (λ).

Si el elemento no está en la lista , le tendremos que comparar con todos los elementos de esa lista. Por lo tanto, se puede esperar que la búsqueda lleve un tiempo $O(1+\lambda)$. La constante *1* representa el tiempo necesario para hallar la lista.

Ahora, supongamos que la búsqueda es exitosa, es decir, el elemento está en la lista. Del análisis de la búsqueda secuencial sabemos que el número promedio de comparaciones es $1/2\ (k+1)$, donde *k* es la longitud de la lista que contiene el elemento que estamos esperando. Pero la longitud de la lista ya no es *N/B*, puesto que sabemos por anticipado que debe contener al menos un elemento (el que estamos buscando). Los otros *N-1* se distribuyen uniformemente sobre las *B* listas. Por tanto, el número esperado de elementos de la lista donde está el elemento que buscamos es $1+(N-1)/B$. Excepto, para las tablas de tamaño trivialmente pequeño podemos aproximar $(N-1)/B$ por (N/B). De aquí que el número promedio de pruebas (número de posiciones de la tabla que deben ser examinadas en la búsqueda de un elemento) para una búsqueda exitosa esté muy cercano.

$$\frac{1}{2}(k+1) \approx \frac{1}{2}(1+\lambda+1) = \left(1+\frac{1}{2}\lambda\right)$$

En el caso de la operación de borrado, el análisis sería el mismo.

El tiempo que requiere la operación de inserción sería $O(1)$, en el supuesto de que se inserten los elementos por el principio de la lista o $O(1+\lambda)$, si se insertan al final. En el caso de que tengamos un puntero al último, insertar el elemento por el final tendría un tiempo de $O(1)$. En el caso de que no se permitan elementos repetidos, tenemos que comprobar antes que el elemento no está. Por lo tanto, si el elemento está ya en la tabla, el tiempo que requiere la operación de inserción sería igual al tiempo de una búsqueda exitosa Si el elemento no está repetido, el tiempo que requiere entonces la operación de inserción sería igual al tiempo de una búsqueda no exitosa más el tiempo de inserción.

Este análisis demuestra que el tamaño de la tabla, por si solo, no es realmente importante, ya que los elementos no se almacenan en la tabla de dispersión en sí, sino en los nodos de las listas. Pero sí lo es el factor de carga . La regla general de una dispersión abierta es hacer el tamaño de la tabla casi tan grande como el número de elementos esperados. En otras palabras, $\lambda = 1$. También es buena idea, como se mencionó al principio, conservar primo el tamaño de la tabla para asegurar la buena distribución.

Todo este análisis está basado en que las listas no están ordenadas. Si lo estuvieran, el tiempo de búsqueda infructuosas se podría reducir, ya que en promedio, sólo la mitad de la lista necesita recorrerse para determinar si el elemento está o no en la lista.

3.8.1.1. Ventajas de la dispersión abierta

- La dispersión abierta permite que el conjunto se almacene en un espacio potencialmente ilimitado, por lo que no impone un límite al tamaño del conjunto.
- La resolución de colisiones es sencilla. No tendremos problemas de agrupamiento primario.
- La eliminación es una tarea fácil y eficaz.
- Si los registros son bastantes extensos se ahorra espacio. Si los registros estuviesen en la tabla, entonces si hay muchas posiciones vacías (como es deseable para ayudar a evitar el coste de las colisiones);éstas consumirán espacio que podría utilizarse en otro lugar. Por otra parte, al contener la tabla sólo los apuntadores, éstos requieren menos espacio que los registros y se podría reducir el tamaño de la tabla.

3.8.1.2. Desventajas de la dispersión abierta

- Ocupa espacio adicional al de la tabla. Los enlaces requieren espacio. Si los registros son extensos, el espacio es despreciable en comparación con el necesario para los registros mismos, no así si los registros son pequeños.
- Requiere el uso de memoria dinámica. Esto tiende a hacer más lento el algoritmo, debido al tiempo necesario para asignar celdas nuevas, y también requiere la implantación de una segunda estructura.
- Exige el manejo de listas enlazadas. Si las listas crecen demasiado se perderá la facilidad de acceso directo del método hash, ya que al realizar la operación de búsqueda, tendrá que recorrer toda la lista.

3.8.2. Dispersión cerrada

En el procedimiento de dispersión cerrada se utiliza la tabla para almacenar los elementos en lugar de almacenar los apuntadores a las listas.

En un sistema de dispersión cerrada, si ocurre una colisión, se intenta buscar celdas alternativas hasta encontrar una vacía. Si ninguna está vacía, la tabla está llena y no es posible insertar. Más formalmente, se busca sucesivamente en las celdas $h0(x)$, $h1(x)$, ., donde $hi(x) = ((Dispersión(x) + f(i)) \bmod TAMAÑO\_TABLA$, con $f(0) = 0$. La función f es la estrategia de resolución de colisiones. Como todos los datos se meten en la tabla, se necesita una tabla más grande para la dispersión cerrada que para la abierta. En general, el factor de carga debe estar por debajo de $fc = 0\ 5$ para la dispersión cerrada.

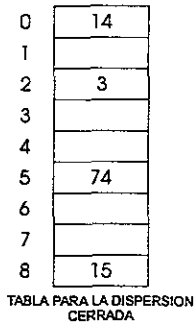


Fig. 3.8.2.1 Dispersión cerrada o direccionamiento abierto

3.9. Estrategias para la resolución de colisiones

Estas estrategias son conocidas como estrategias de redispersión. Las más comunes son:

3.9.1. Exploración lineal

Si la dirección calculada está ocupada, se comprueba la siguiente y así se recorren todas las celdas en secuencia (la siguiente de la última es la primera) en busca de una celda vacía. Aquí la función f es una función lineal, por lo general $f(i) = i$.

Este esquema es muy sencillo pero tiene algunos inconvenientes. En tanto que la tabla sea suficientemente grande, y no está llena, siempre se puede encontrar una celda vacía, pero ello puede llevar demasiado tiempo. Pero la desventaja fundamental, aun si la tabla está relativamente vacía, es que los registros tienden a agruparse, o sea, a aparecer unos juntos a otros, cuando el factor de carga es mayor del 50%. Este efecto se denomina agrupamiento primario y aumenta el tiempo medio de búsqueda para un elemento. Este agrupamiento sucede cuando la función hash no distribuye las direcciones uniformemente por todas las direcciones.

3.9.2. Exploración cuadrática

Este método elimina el problema del agrupamiento primario. En la exploración cuadrática la función de colisión es cuadrática. La elección común es $f(i) = i^2$. Ahora buscamos linealmente en las posiciones con direcciones: h, $h + 1$, $h + 4$, $h + 9$, .. , $h + i^2$.

En la exploración lineal es mala idea dejar que la tabla de dispersión esté casi llena, porque se degrada el rendimiento. Para el sondeo cuadrático, la situación aún es más drástica. No hay garantía de encontrar una celda vacía una vez que la tabla se llena a más de la mitad, o aun antes, si el tamaño de la tabla no es primo. Esto se debe a que a lo más la mitad de la tabla se puede usar como posiciones alternativas al resolver las colisiones. Lo que sí sucede es que si la tabla está medio vacía y su tamaño es primo, podemos contar siempre con la seguridad de poder insertar un elemento nuevo[22].

Aunque la exploración cuadrática elimina el agrupamiento primario, los elementos que se dispersen a la misma posición irán probando en las mismas celdas alternas. Esto se denomina agrupamiento secundario.

3.9.3. Desplazamiento cociente

Para calcular la dirección o posición inicial se utiliza el método de división. Cuando ocurre una colisión, es necesario calcular la siguiente posición, con lo cual a la dirección inicial se le suma un desplazamiento. Este es igual al resultado de dividir la clave por el tamaño de la tabla. Cuando el desplazamiento es múltiplo del tamaño de la tabla, entraríamos en un bucle infinito, ya que siempre iríamos a la misma posición. Cuando esto ocurre, se hace que el desplazamiento sea 1.

Con el desplazamiento cociente se evita el agrupamiento secundario. El cociente es independiente del resto. Por lo tanto, dos claves con el mismo resto tienen más probabilidades de tener diferentes cocientes y, por lo tanto, distintos desplazamientos.

El agrupamiento primario se minimiza (aunque no se elimina por completo), dependiendo del cálculo de la dirección inicial y de cómo ésta se disperse a lo largo de toda la tabla.

Las diferentes posiciones que vamos comprobando siguen la fórmula *hi(x) = ((Dispersión(x) + c) mod TAMAÑO_TABLA)*. Donde *c* es una constante que es igual al desplazamiento. Con este tipo de funciones no tenemos la seguridad de que recorramos todas las posiciones de la tabla. Por ejemplo, si *c* es igual a *2*, cualquier clave cuya dirección inicial fuese impar, se redispersaría en posiciones impares. Si todas éstas estuviesen ocupadas, no se encontraría ninguna posición libre, aunque todas las posiciones pares estuviesen vacías.

Para que esto no ocurra, *c* y el tamaño de la tabla deben ser primos relativos, es decir, no pueden ser ambos divididos por un entero que no sea 1. Una forma de garantizar ésto, es que el tamaño de la tabla sea primo[23].

3.9.4. Dispersión doble

Para este método la elección común es *f(i) = i * H' (x)*. Aquí se usa una segunda función hash *H'* para resolver la colisión de la siguiente forma. Si la clave *k* tiene las direcciones *hash H(k) = h* y *H'(k) = h'* (distinto de *m*), entonces buscamos linealmente en las posiciones con direcciones:

$$h, h+h', h+2h', h+3h' \ldots\ldots$$

Si el tamaño de la tabla m es un número primo, entonces la secuencia anterior accederá a toda la tabla. En caso contrario, es posible que no encontremos posiciones donde insertar el elemento, aunque haya posiciones vacías.

Veamos un ejemplo para ilustra esto[24]. Supongamos que tenemos una tabla de tamaño 10 y una función *H'(x) = R - (x mod R)* con *R* un número primo menor que el tamaño de la tabla. Escogemos *R* igual a *7*. Insertamos el *89* en la posición *9*. Insertamos el *18* en la posición *8*. Insertamos el *58*. Éste tendría que ir en la posición *8*, pero ya está ocupada. Calculamos *H'(58) = 7 - 2 = 5*, con lo cual el *58* iría a la posición *3*. Si ahora queremos insertar el *23*, entraría en colisión con el *58*.

Calculamos $H'(23) = 7 - 2 = 5$, con lo cual sólo tenemos una posición alternativa (la 8) y ésta ya está ocupada.

Otra posible elección de $H'(k)$, si m es primo y $H(k) = k \bmod m$, es hacer $H'(k) = 1 + (k \bmod (m-1))$. Pero como $m-1$ es par sería mejor $H'(k) = 1 + (k \bmod (m-2))$. Se sugiere la elección de m, tal que m y $m-2$ sean "parejas de primos", como *1021 y 1019*.

Una mala elección de $H'(x)$ sería desastrosa. Si se implementa correctamente la dispersión doble, el número esperado de intentos es al menos el mismo que en la estrategia aleatoria de resolución de colisiones. Esto hace interesante la dispersión doble desde un punto de vista teórico. La exploración cuadrática, sin embargo, no requiere el uso de una segunda función hash y así es, probablemente más simple y fácil de calcular (no requiere multiplicaciones ni divisiones adicionales).

3.9.5. Estrategia aleatoria

Se utiliza un generador de números aleatorios para obtener *i*. El generador utilizado debería ser uno que siempre genere la misma serie proporcionada que comience con el mismo origen. El origen, entonces, puede especificarse como alguna función llave. Este método es excelente para evitar la agrupación primaria, pero probablemente más lento que otros.

3.9.6. Estrategia uniforme

Dada una clave probaríamos con una permutación de elementos entre *0* y *TAMAÑO_TABLA - 1*. Cada permutación es distinta y depende de la clave. Dentro de cada permutación los elementos no se repiten.

3.10. Rehashing

Con las técnicas que se han presentado, existe un problema cuando se trata de agregar más entradas, especialmente para tablas con un tamaño fijo. Aun con la dispersión abierta, que permite agregar cualquier número de entradas , eventualmente se tienen problemas. Como las cadenas que llegan a ser muy largas para ser prácticas, la tabla debe ser diseñada acorde a las entradas que se esperan, o de lo contrario prever de alguna manera el crecimiento de la misma.

Una técnica que se usa cuando una tabla se llena, es crear otra. Una más larga, quizá dos veces el tamaño, e insertar las entradas desde la primera tabla. Esto es los que se llama Rehashing[25]. Para cada clave se debe aplicar la función hash, debido a que el número de los buckets es diferente. Dependiendo del tamaño de la tabla aumenta el costo.

Existen muchas técnicas que permiten crecer la tabla sobre la marcha, pero son complicadas, y casi todas las técnicas sufren pequeños problemas de desempeño aun si la tabla no necesita crecer. Otra técnica popular se llama hashing extendible, la cual arma una estructura de directorio jerárquica basada en los patrones de bit de las claves hashed[26].

3.11. Hashing extendible

En el caso en que la cantidad de datos sea demasiado grande para caber en memoria principal, se puede utilizar el hashing extendible (también conocido como ampliable). Este es un método de acceso a ficheros, especialmente indicado para aquellos que sufren cambios y actualizaciones frecuentes. Está diseñado para localizar cualquier registro de datos como máximo en *2* accesos al medio externo de almacenamiento en que se encuentre el fichero. Es una combinación perfecta del hashing y de la búsqueda de raíz. Lo que caracteriza a este método es que el índice del hashing ampliable puede expandirse y contraerse.

Los métodos hashing de almacenamiento interno que se han descrito antes comienzan con una tabla hash de tamaño fijo y una función hashing que depende del tamaño de esa tabla. Si el factor de carga de la tabla se acerca mucho a 1 y hay muchas colisiones, no es posible incrementar el tamaño de la tabla sin tener que pasar todos los registros a la nueva tabla y calcular su nueva dirección con el nuevo tamaño. Por el contrario, el hashing ampliable permite que crezca la tabla hash cuando sea necesario sin tener que mover los registros.

La estructura entera del hashing ampliable está formado por páginas, cajones o buckets. Una página es un área de tamaño fijo que puede contener registros de datos o punteros a otras páginas. Una página-directorio sólo contiene punteros. Una página-hoja contiene claves y sus registros o punteros hacia otros registros.

El directorio puede constar de una o más páginas, y sus entradas apuntan a páginas hojas. El número de entradas del directorio será una potencia de 2. La función hashing usada puede ser cualquiera que sea apropiada, es decir, que distribuya las claves de forma uniforme sobre el espacio de direcciones.

El tiempo para acceder a un elemento es el mismo que para acceder a cualquier otro. No existe la circunstancia de ejecución en mejores o peores condiciones. Los 2 accesos para localizar un elemento son: uno para acceder al directorio y otro para acceder a la página-hoja. Si la hoja apuntaba al registro de datos en vez de contener el registro, habrá un acceso adicional. Lo que hace eficaz el hashing ampliable, es la partición de los datos en grupos sobre la base del valor hash de su clave.

A la hora de trabajar con el hashing ampliable debemos tener en cuenta dos factores.

- Un número "D" que nos indica la porción de la dirección hash que vamos a utilizar como índice del directorio. A este número se le suele llamar profundidad del directorio
- Un número "D'", tal que $D' <= D$, que nos indica el número de dígitos de la dirección hash comunes a todos los registros de una página. A este número se le denomina profundidad de una página

Teniendo esto en cuenta, el tamaño del directorio será 2^D.

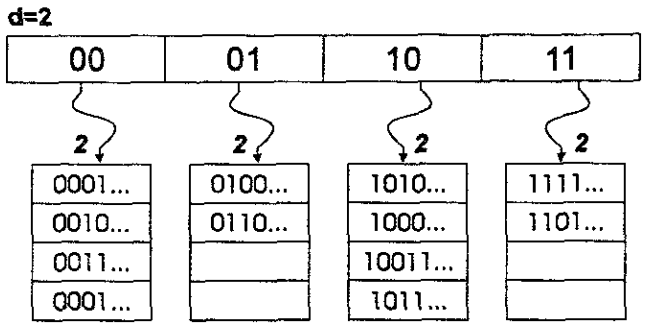


Fig. 3.11.1 Hashing extendible

3.12. Tablas hash en la agregación limitada por difusión

El problema que plantea el modelo bicolor DLA, es la necesidad de calcular de forma rápida las posiciones de las partículas caminantes[V]. Para cada una de estas posiciones es necesario saber si las partículas caminantes están ya unidas a alguna semilla o no lo están. Debido a que el número de posiciones que toma la partícula caminante son muchas, se requiere de un algoritmo que permita hacer cálculos rápidos.

Cuando se trata de tres dimensiones, la posición de la partícula se almacena en arreglos tridimensionales que van creciendo de la misma forma en que las partículas forman estructuras más grandes.

Mientras mas partículas se unen, más cálculos se requieren para saber si una partícula caminante se unirá o no a alguna estructura fractal, esto influye directamente en las características de la computadora que se utiliza.

El algoritmo de la tabla de hash puede aplicarse a cualquier dimensión y el tiempo no depende da la cantidad de partículas que ya estén formando la estructura, este tiempo se mantiene casi constante.

Consideremos una estructura del modelo DLA que ya ha crecido, teniendo como única una semilla la partícula localizada en las coordenadas *(0,0,0)*;

(1,0,0)
(1,1,0)
(1,1,1)
(2,1,1)

[V] Recuérdese que dicha partícula toma posiciones aleatorias

Y sea *(3,-1,2),* las coordenadas de una partícula caminante, como se observa en la figura *3.12.1*:

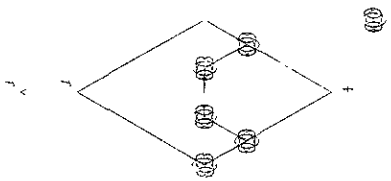


Figura 3.12 1

Considerando a *M* como la cantidad máxima de partículas caminantes

El algoritmo consiste en obtener un índice para cada partícula que ya esta integrada a la estructura, así como de la partícula caminante

(Ax + By +Cz) es el índice y *x,y,z* son las coordenadas de las partículas, como se muestra:

Coordenadas	(Ax + By +Cz)
(1,0,0)	A
(1,1,0)	A + B
(1,1,1)	A + B + C
(2,1,1)	2A + B +C

Las constantes *A, B* y *C*, se obtienen como sigue:

$$A \approx M^{1/D}$$

$$B \approx M^{2/D}$$

$$C \approx M^{3/D}$$

En donde *D* es la dimensión de trabajo + *1*, en esta caso *D= 3+1 = 4*

En los cálculos *M* y *D* son conocidos, así como las coordenadas tanto de las partículas que ya forman la estructura como de la partícula caminante.

Para evitar colisiones el algoritmo indica que las constantes *A, B, C* (reales), se deben aproximar al número primo más cercano.

Consideremos el siguiente ejemplo:

DIMENSIONES = **3**
D = 4
M = **1000**
A = 5.62341325
B = 31.6227766
C = 177.827941

Los valores primos cercanos para las constantes *A*, *B* y *C* son *5*, *31* y *177* respectivamente

Consideremos que se inicia con una semilla en el centro de un enrejado tridimensional[VI], y que las coordenadas del vértice inferior son *(0,0,0)*. Donde la función hash correspondiente es:

$$(Ax + By + Cz) \bmod M$$

Para este ejemplo *M=1000* (véase la figura *3.12.2*)

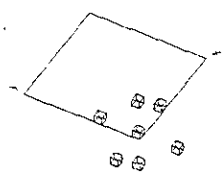


Figura 3.12 2

En la figura *3 12.2* observe cuidadosamente las *6* alternativas de crecimiento, como lo indica la tabla *3.12 1*:

Alternativas	*(X,Y,Z)*	*(0,0,0)*
1ª	*(X+1,Y,Z)*	*(1,0,0)*
2ª	*(X,Y+1,Z)*	*(0,1,0)*
3ª	*(X,Y,Z+1)*	*(0,0,1)*
4ª	*(X-1,Y,Z)*	*(-1,0,0)*
5ª	*(X,Y-1,Z)*	*(0,-1,0)*
6º	*(X,Y,Z-1)*	*(0,0,-1)*

Tabla 3.12 1

Considerando que aparece una partícula caminante cuyas coordenadas son *(1,0,0)*, es necesario verificar si la función hash aplicada a las seis alternativas de la unión de la partícula, tiene un residuo igual al de la semilla[VII].

[VI] Para el modelo DLA

Al aplicar la formula *(Ax+By+Cz)mod M*, obtenemos los residuos mostrados en la tabla *3.12.2*:

Alternativas		x	y	z	A	B	C	RESIDUO
1ª	(X+1,Y,Z) = (2,0,0)	2	0	0	5	31	177	10
2ª	(X,Y+1,Z) = (1,1,0)	1	1	0	5	31	177	36
3ª	(X,Y,Z+1) = (1,0,1)	1	0	1	5	31	177	182
4ª	(X-1,Y,Z) = (0,0,0)	0	0	0	5	31	177	0
5ª	(X,Y-1,Z) = (1,-1,0)	1	-1	0	5	31	177	-26
6º	(X,Y,Z-1) = (1,0,-1)	1	0	-1	5	31	177	-172

Tabla 3.12.2

La 4ª alternativa tiene el mismo residuo que la semilla, por lo tanto se agrega el residuo de la partícula caminante al arreglo de residuos y se incrementa la estructura en una partícula.

En cada nueva partícula caminante, se deberán verificar sus seis alternativas de unión con las dos partículas (semilla y primer partícula unida), en caso de que el residuo de alguna alternativa, este contenido en el arreglo de residuos, se agregara el residuo de la partícula caminante al arreglo de residuos (cada vez sé ira llenando la tabla de registros), como se muestra en la figura *3.12.3*;

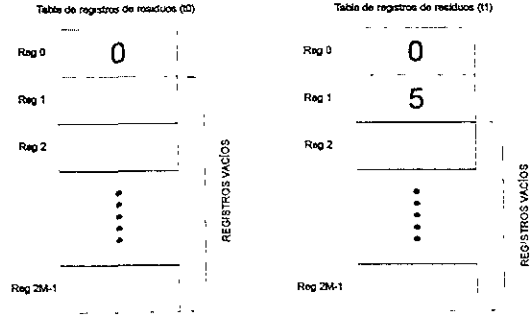


Figura 3.12.3

[VII] Recordar las constantes *A*, *B* y *C* para este ejemplo donde como ya dijimos son los valores primos cercanos 5, 31 y 177 respectivamente

Continuemos para la siguiente partícula caminante que aparece en las coordenadas *(0,2,0)*, realizando la prueba en sus seis alternativas, tenemos:

Alternativas		X	y	z	A	B	C	RESIDUO
1ª	(X+1,Y,Z) = (1,2,0)	1	2	0	5	31	177	67
2ª	(X,Y+1,Z) = (0,3,0)	0	3	0	5	31	177	93
3ª	(X,Y,Z+1) = (0,2,1)	0	2	1	5	31	177	239
4ª	(X-1,Y,Z) = (-1,2,0)	-1	2	0	5	31	177	57
5ª	(X,Y-1,Z) = (0,1,0)	0	1	0	5	31	177	31
6º	(X,Y,Z-1) = (0,2,-1)	0	2	-1	5	31	177	-115

Como puede observarse en la tabla anterior, ningún residuo encontrado en las seis alternativas de esta partícula caminante, se encontraba en la tabla de registros, por lo cual esta partícula aun no se agrega a la estructura y por lo tanto el residuo de la partícula caminante no se agrega[VIII]:

Del mismo ejemplo anterior, supondremos que la partícula caminante se desplaza un paso a la coordenada *(0,1,0)*, veamos ahora que sucede evaluando sus seis alternativas, como se muestra a continuación:

Alternativas		x	y	z	A	B	C	RESIDUO
1ª	(X+1,Y,Z) = (1,1,0)	1	1	0	5	31	177	36
2ª	(X,Y+1,Z) = (0,2,0)	0	2	0	5	31	177	62
3ª	(X,Y,Z+1) = (0,1,1)	0	1	1	5	31	177	208
4ª	(X-1,Y,Z) = (-1,1,0)	-1	1	0	5	31	177	26
5ª	(X,Y-1,Z) = (0,0,0)	0	0	0	5	31	177	0
6º	(X,Y,Z-1) = (0,1,-1)	0	1	-1	5	31	177	-146

Observemos que el residuo de la quinta opción, ya se encuentra en la tabla de registros, por lo cual esta partícula se agrega a la estructura y se adiciona a la tabla de registros el residuo de la partícula caminante evaluado en las coordenadas *(0,1,0)*, esto es:

	x	Y	z	A	B	C	RESIDUO
(X,Y,Z) = (0,1,0)	0	1	0	5	31	177	31

Quedando la siguiente tabla de registros de residuos como lo indica la figura *3 12.4*:

[VIII] El residuo de esta partícula caminante en las coordenadas (2, 2, 0) es 62, aplicando la formula $(A_x + B_y + C_z) mod\ M$

Bibliografía

- Alexander Shen, *Algorithms and Programming*, Inst. for Problems of Info. Transmission, Moscow Price, 1996
- Donald E. Knuth, *Mathematics for the Analysis of Algorithms*, Stanford University, Stanford CA, 1990
- Josef Pieprzyk, Babak Sadeghiyan, *Design of Hashing Algorithms*, , Lecture notes in computer science
- A. V. Hopcroft, J. E. Ullman, *Data Structures and Algorithms*, Addison Wesley, Reading mass
- Donald E. Knuth, *The Art of Computer Programming vol. 1: Fundamental Algorithms*, Addison Wesley 1979
- Donald E. Knuth, *The Art of Computer Programming vol. 3: Sorting and Searching*, Addison Wesley 1979

4. Enumeración Exacta

4.1. Objetivos

A partir de un programa de simulación que se aplique al modelo bicolor DLA para cualquier número de partículas, desarrollar otro que proporcione datos en el crecimiento inicial de estructuras fractales aplicados a este modelo, como son: Distribuciones de probabilidad, números y posiciones exactas de las partículas en el crecimiento inicial para diferentes valores de distancia *d* entre semillas y probabilidad *p* de que la partícula caminante sea de un color u otro.

4.2. Ejemplo de una prueba simple del modelo computacional

Cada prueba realizada mediante el desarrollo computacional de enumeración exacta para el modelo bicolor de agregación limitada por difusión, se basa principalmente en ejecutar hasta 5000 experimentos y cuantificar en cada uno de ellos, las partículas agregadas y las posiciones de agregación en que se van adhiriendo para la formación de dos diferentes estructuras fractales.

La ejecución del programa requiere que el usuario especifique;

a) la distancia de separación entre semillas (*1* a *5*)

b) la probabilidad de que las partículas que se generan sean de *color 1* (entre *0* y *0.5*)

c) la cantidad de experimentos a realizar (como máximo *5000*)

d) La cantidad máxima de partículas que deben integrar cualquiera de los dos fractales y que sirve como referencia para indicarle al programa que la estructura ya alcanzó la cantidad de partículas previamente especificadas.

Como resultado final, el programa entregara dos archivos en ASCII (formato *.dat), uno de ellos nos proporciona la información general de la prueba y los resultados específicos de cada uno de los experimentos: Posición de la Semilla de *color 1* y *2* respectivamente, así como la cantidad y posición de cada una de las partículas que se van integrando a cada estructura fractal. El otro archivo proporciona datos y resultados globales de cada experimento, de tal forma que estos puedan ser analizados y procesados en forma independiente.

De los resultados relevantes de cada experimento destacan, la cantidad de partículas generadas en forma total por experimento, sin importar aquellas que mueren ya sea porque se salen de los límites del enrejado o bien porque tratan de adherirse a una estructura de diferente color, las cantidad de partículas que se integran a cada estructura fractal y la razón de asimetría entre las dos estructuras fractales, definiendo la razón de asimetría como el $log\ (N_{max}/N_{min})$, donde $N_{max} = max\ (N1, N2)$ y N_{min}

= *min (N1,N2)*, y donde *N1* y *N2* son la cantidad de partículas que integran cada una de las estructuras fractales.

Para esta prueba en particular se tienen los siguientes datos:

Distancia de la *semilla 1* al origen: *3*

Por lo cual la distancia entre semillas es: *6*

Probabilidad de que la partícula creada sea de *color 1* (*0 a 0.5*): *0 462564* [IX]

Experimentos a realizar (*1 a 5000*): *5*

Número máx. de partículas por fractal (*2 a 10 máx.*): *2*

Con los datos anteriores, al finalizar la ejecución del programa se obtienen los siguientes datos:

Probabilidad de Color 1 = *0.462564*
Número de Experimentos a realizar = *5*
Distancia entre semillas = *6*
Número máx de partículas por fractal = *2*

No. Exp.	Cantidad de partículas en la estructura	X	Y	
1	1	-3	0	***Semilla Color 1***
1	1	3	0	***Semilla Color 2***
1	2	4	0	Fractal Color 2
2	2	2	0	Fractal Color 2
3	2	-3	1	Fractal Color 1
4	2	-3	1	Fractal Color 1
5	2	3	1	Fractal Color 2

Los resultados anteriores es la parte sustancial del primer archivo de resultados de esta prueba simple y los siguientes resultados forman la parte global de esta prueba.

Probabilidad de Color 1	No. Experimento	Distancia entre semillas	Total de Partículas generadas	Total de Partículas integradas al Fractal 1	Total de Partículas integradas al Fractal 2	Razón de Simetría entre Fractales log(Nmax/Nmin) (base 10)
0 462564	1	6	5	2	1	0.30103
0.462564	2	6	4	2	1	0.30103
0 462564	3	6	6	1	2	0.30103
0.462564	4	6	6	1	2	0 30103
0.462564	5	6	1	2	1	0 30103

Ahora bien, regresando a los datos que dan origen a esta prueba de ejemplo. El primer dato en el que indicamos que la distancia de la *semilla 1* al origen es *3*, sirve para que el programa establezca

[IX] Este valor se escoge por ser de relevancia según un análisis teorico previamente hecho usando el grupo de renormalizacion.

las coordenadas donde se sitúan las semillas de *color 1* y *2* respectivamente, en este caso las coordenadas son las siguientes *(-3 , 0) y (3 , 0)*[X].

Como se puede observar en la primer tabla de resultados de esta prueba, en el primer experimento se adhiere la 2ª partícula a la estructura del fractal de *color 2* en la posición *(4 , 0)*, es ahí donde finaliza este primer experimento, ya que con los datos iniciales le indicamos al programa que la cantidad máxima de partículas por fractal debería ser *2* (en cualquiera de las dos estructuras), en esta misma tabla se observa que la primera partícula que integra cada fractal son por si mismas las semillas. En el siguiente experimento No. 2, se inicia nuevamente con las *semillas 1* y *2* en las posiciones *(-3 , 0) y (3 , 0)* respectivamente y se observa que la segunda partícula se adhiere en (*2 , 0*) correspondiente también a la estructura del fractal de *color 2*, así sucesivamente se obtienen los puntos de agregación para los experimentos *3, 4 y 5* respectivamente.

La otra tabla de resultados nos indican como ya dijimos los aspectos globales de cada experimento. Para el experimento *1* nos indica que el número total de partículas generadas fué de *5*, esto quiere decir que si solamente una se integró al fractal *2* y se dio por finalizado el experimento, entonces *4* partículas murieron, ya sea por alejarse mucho de las estructuras fractales o bien porque trataron de integrarse a estructuras de diferente color. Otros resultados importantes obtenidos en esta prueba son la cuantificación de partículas que forman cada estructura al finalizar cada experimento, sin embargo para el caso de este ejemplo siempre una estructura va a tener *2* partículas integradas y la otra solamente *1*. Por último en esta última tabla se observa también lo que denominamos razón de simetría fractal entre estructuras que como ya se mencionó, es la relación logarítmica entre la cantidad máxima de partículas que integra una estructura fractal con relación a la otra.

El ejemplo anterior es realmente simple ya que siempre que se adhiera alguna partícula a cualesquiera de las estructuras fractales, esto originará que se finalice dicho experimento. Sin embargo esta aparente simplicidad se convierte en un proceso realmente complejo al indicarle al programa que la prueba consistirá de *5000* experimentos, que la distancia entre semillas es *10*, que la cantidad máxima de partículas por fractal es de *10* y que es necesario cuantificar todas las trayectorias y formas en la formación de las estructuras fractales, este programa computacional puede proporcionarnos resultados relevantes para este tipo de problemas complejos.

El análisis de los resultados obtenidos nos proporciona un conocimiento detallado en el crecimiento y fluctuaciones de agregación de las partículas al inicio de la formación de estructuras fractales complejas, se espera además que estos resultados sean una aportación en la descripción cuantitativa

[X] Ver anexo 2, análisis de radios de giro y de muerte

y probabilística de patrones complejos y desordenados, como son la mayoría de fenómenos naturales.

4.2.1. El detalle de esta prueba

Análisis sobre la posición inicial de las semillas

Cuando el programa le solicita al usuario que proporcione la distancia de la *semilla 1* al origen, se le sugiere introduzca un número entre *1* y *5*, el usuario puede escoger el número que le sea conveniente. Sin embargo el programa primeramente asigna las coordenadas de cada una de las semillas de acuerdo al dato proporcionado por el usuario, en la tabla 4.2.1.1 se muestran las asignaciones de coordenadas que el programa proporciona de acuerdo a distancias de la semilla 1 al origen que van de 1 a 5.

Distancia de la semilla 1 al origen	Coordenadas de la semilla 1	Coordenadas de la semilla 2	Distancia entre semillas
1	(-1,0)	(1,0)	2
2	(-2,0)	(2,0)	4
3	(-3,0)	(3,0)	6
4	(-4,0)	(4,0)	8
5	(-5,0)	(5,0)	10

tabla 4.2 1.1 Asignaciones de coordenadas en función de la distancia entre semillas

Caso 1. En el caso de que el usuario indique que la distancia de semilla 1 al origen es 1, entonces se tiene lo siguiente:

Siempre el número de partículas al inicio en las dos estructuras fractales es uno, ya que las semillas son por sí mismas partículas.

Para la semilla de *color 1* cuya única partícula integrada tiene coordenadas *(-1, 0)*, se calcula la distancia R^2, en donde

$$R^2 = x^2 + y^2$$

en este caso $R^2 = 1$

También al inicio del programa existirá una variable inicial *CurrentDist_2* con valor de *0*, que se tratará como sigue:

if (R2 > CurrentDist_2) CurrentDist_2 = R^2, de lo que se obtiene que *CurrentDist_2 = 1*

A continuación se calcula un radio máximo como sigue

$MaxRadius = 2 + \sqrt{(CurrentDist\_2)}$, y se obtiene que MaxRadius es igual a 3

$MaxRadius\_2 = MaxRadius^2$, con lo cual MaxRadius_2 es igual a 9

Para la semilla de *color 2* cuya única partícula integrada tiene coordenadas *(1, 0)*, se realizan exactamente los mismos pasos anteriores.

De lo anterior se obtienen otros dos radios mediante las siguientes formulas

$Radius = MaxRadius + 5$
$Radius\_Kill = MaxRadius + 15$

con lo que se obtiene que *Radius = 8* y *Radius_Kill= 18*

Como se puede observar hasta ahora, las posiciones de las semillas son determinantes en el cálculo de los radios máximos y de muerte al inicio del experimento, estas variables seguirán sufriendo modificaciones en sus valores conforma las estructuras fractales vayan creciendo.

La primera partícula caminante que se genera aparecerá en un enrejado cuyo radio estará delimitado por el valor de Radius calculado previamente y cuyas coordenadas estarán dadas por:

*X0=(int)ceil(Radius*cT);*

*Y0=(int)ceil(Radius*sT);*

Lo que significa que *X0* esta dado por el techo del número (redondeo de *X0* al entero más pequeño no menor que *X0*), que se obtiene del producto de *Radius* por *cT*, donde *cT* es el coseno de *Theta* y theta proviene del producto de dos números, uno de ellos se obtiene mediante el generador de números aleatorios ran3[XI].

Y0 se obtiene en forma similar, solo que *sT* en este caso es el seno de *Theta*, y al igual que en el caso anterior *theta* proviene del producto de dos números, uno de ellos también obtenido mediante el generador de números aleatorios ran3.

Supongamos que las coordenadas de la primer partícula caminante son *(8, 2)*, el color de dicha partícula se obtiene mediante la comparación del valor de probabilidad de que la partícula generada sea de color *1* con el número obtenido a través del generador de números aleatorios ran3[XII], de tal

[XI] Ver anexo 1, en donde se explica algoritmo del generador de numeros aleatorios ran3
[XII] Este método genera numeros aleatorios con distribución uniforme en el rango de 0.0 a 1.0

forma que si el número generado es menor al valor de la probabilidad proporcionado por el usuario, entonces la partícula caminante será de *color 1*, en cualquier otro caso la partícula será de *color 2*.

Regresando al ejemplo anterior, sea que la partícula caminante localizada inicialmente en *(8, 2)* es de *color 1*, el sistema computacional creará un objeto cuyas propiedades serán posición en x, posición en y, y el *color*.

A partir de este momento la partícula caminante inicia sus movimientos, veamos la tabla *4.2.1.2* para visualizar los diferentes caminos que puede tomar esta partícula caminante en particular:

Caso	Movimiento	Coordenada Inicial	Coordenada después del movimiento
1	Derecha	(8, 2)	(9, 2)
2	Izquierda	(8, 2)	(7, 2)
3	Arriba	(8, 2)	(8, 3)
4	Abajo	(8, 2)	(8, 1)

Tabla 4.2.1 2

Supongamos que el primer movimiento lo hace hacia arriba, ahora la partícula se localizara en las coordenadas *(8, 3)*, en ese momento el programa debe realizar;

a) el cálculo de la distancia la distancia a la que ahora se encuentra la partícula al centro del enrejado

b) Con los valores anteriores deberá determinar que magnitud será el paso en el siguiente movimiento aleatorio de la partícula, brevemente esto nos indica que conforme la partícula se aleja demasiado, entonces el paso se debe incrementar, todo con el fin de hacer más eficiente el programa y forzar así a la partícula a alejarse y en su caso salir del enrejado o bien a acercarse mas rápidamente a las estructuras fractales. Lo anterior se logra primeramente calculando a que distancia se encuentra la partícula y comparándola con el radio máximo como se indica en las siguientes fórmulas:

$w = \sqrt{(x^2 + y^2)}$ ahora este valor obtenido es comparado como sigue

```
if (w < MaxRadius+10) Paso=1;
  else
    if (w < MaxRadius+20) Paso=2;
```

```
else
  if (w < MaxRadius+40) Paso=4;
    else
      if (w < MaxRadius+80) Paso=8;
      else Paso=16;
```

En el caso de nuestro ejemplo, la magnitud a la cual se encuentra la partícula caminante en la posición *(8,3)*, es la siguiente;

$$w = \sqrt{(8^2 + 3^2)} = 8.5440$$

MaxRadius = 8, entonces se cumple que;

8.544 < [*MaxRadius* + *10*], por lo cual la magnitud del siguiente paso de la partícula caminante será de *1*.

Sí la partícula continua alejándose de las estructuras fractales y después de varios movimientos llega a la posición *(12, 4)*, la distancia al centro del enrejado sería;

$$w = \sqrt{(12^2 + 4^2)} = 12.649,$$

este valor de *w* siendo menor a *13*, por lo cual su siguiente movimiento también será de una unidad. Sea que la partícula caminante en su movimiento aleatorio se desplaza a las coordenadas *(13, 4)*, entonces w estaría dada por;

$$w = \sqrt{(13^2 + 4^2)} = 13.601,$$

donde $w = 13.601$ es mayor que [*MaxRadius* +*10*], pero menor que [*MaxRadius* + *20*], dado lo anterior, el siguiente movimiento de la partícula debe ser de *2* unidades.

Estas comparaciones las realiza en todas las partículas que se van generando para determinar la magnitud de sus movimientos, sin embargo no son en sí las únicas comparaciones, ya que el sistema también debe determinar cuando dá por muerta una partícula si se ha alejado una distancia considerablemente grande, para lo cual también realiza la siguiente comparación, después de cada movimiento.

Cuando $x^2 + y^2 \rangle Radius\_Kill^2$

Cuando se cumple la anterior desigualdad, se toma por muerta la partícula por rebasar el radio de muerte, en el caso de nuestro ejemplo, tenemos que *Radius_Kill* = *18*, por lo cual veamos en la tabla *4.2 1.3* en cuales coordenadas del enrejado la partícula caminante se considera como muerta, por rebasar el radio de muerte.

Coordenada	$x^2 + y^2$	Radius_Kill^2	Posición de la partícula respecto al radio de muerte
(9, 14)	277	$18^2 = 324$	Adentro
(10, 14)	296	$18^2 = 324$	Adentro
(10, 15)	325	$18^2 = 324$	Afuera, por lo tanto muere la partícula
(13, -13)	338	$18^2 = 324$	Afuera, por lo tanto muere la partícula

Tabla 4.2.1.3

Otra comparación muy importante la realiza para determinar la cercanía de la partícula caminante a alguna estructura fractal, y de ser así realiza un proceso muy cuidadoso para probar si la partícula en alguno de sus siguientes movimientos se adhiere o no a alguna estructura fractal. Esta comparación es parte sustancial del alto desempeño del programa, dado que si se determina que la partícula caminante esta alejada de alguna estructura fractal, el programa deberá omitir algunos cálculos que no son importantes dado que por lo lejano de la partícula podemos se sabe que no esta unido a ninguna estructura fractal, sin embargo en el caso contrario en el que la partícula este lo suficientemente cercana a alguna estructura fractal, es muy conveniente observarla cuidadosamente para determinar rápidamente cuando la partícula caminante se adhiere o muere por estar en contacto con una estructura fractal. Estas comparaciones las realiza como sigue:

Cuando se cumple la siguiente desigualdad de cercanía

$$x^2 + y^2 \langle MaxRadius\_2,$$

Se considera que una partícula esta cercana a alguna estructura fractal.

Consideremos que ahora la partícula caminante se encuentra en la posición (-3, -1), se tiene;

$MaxRadius\_2 = 9$ (en este ejemplo, esto es valido solo al inicio del experimento[XIII])

$$x^2 + y^2 = (-3)^2 + (-1)^2 = 10$$

Se observa que no se cumple la desigualdad de cercanía, por lo cual no se considera todavía cercana la partícula a alguna estructura fractal.

Sea que esta misma partícula se desplaza en su siguiente movimiento hacia la derecha, se localizará ahora en las coordenadas *(-2, -1)*, en donde

$$x^2 + y^2 = (-2)^2 + (-1)^2 = 5,$$

En este momento si se cumple la desigualdad y se considera que dicha partícula se encuentra muy cercana a alguna estructura fractal.

[XIII] Dentro del ejemplo, este valor es correcto, siempre y cuando las dos estructuras fractales solo consten de sus semillas y estas se encuentren en las posiciones (-1, 0) y (1, 0) como lo indica este ejemplo, en cualquier otro caso, es muy probable que este valor sea diferente.

En nuestro ejemplo particular la semilla de color 1 se localiza en las coordenadas *(-1, 0)*, por lo cual una partícula localizada en la posición *(-2, -1)* no se encuentra pegada a la estructura, sin embargo se le considera que tiene muchas probabilidades de agregarse a la semilla.

Ahora es importante entender brevemente qué comparaciones y pruebas hace el programa cuando considera que la partícula caminante se encuentra muy cercana a una estructura.

Cuando el programa determina que la partícula esta muy cercana a una estructura fractal, utiliza la instrucción de control *switch* para indagar el color de la partícula caminante[XIV]. Una vez que determina que color tiene la partícula caminante, compara y verifica en las posiciones adyacentes a la posición de la partícula caminante, con el fin de ver si alguna de estas posiciones adyacentes esta ocupada por alguna estructura fractal..

Regresemos al ejemplo de la partícula caminante situada en las coordenadas *(-2, -1)*, considerando además que las dos estructuras fractales solo estaban integradas por sus semillas propias, o sea que no habían integrado ninguna partícula caminante. Es muy fácil determinar entonces que hasta este momento la partícula caminante en cuestión no esta agregada a ninguna partícula, ya que sus posiciones adyacentes no contemplan las posiciones de las semillas (véase la tabla *4.2.1.4*).

Renglón	Objeto o posición	X	Y	Color
1	Semilla 1	-1	0	Color 1
2	Semilla 2	1	0	Color 2
3	partícula caminante	-2	-1	No se sabe hasta el momento[XV]
4	Posición a la derecha de la partícula caminante	-1	-1	
5	Posición arriba de la partícula caminante	-2	0	
6	Posición a la izquierda de la partícula caminante	-3	-1	
7	Posición abajo de la partícula caminante	-2	-2	

Tabla 4.2 1.4

De esta tabla se puede observar lo siguiente:

1. Los renglones *1*, *2* y *3* corresponden propiamente a las dos estructuras fractales y a la partícula caminante en cuestión

[XIV] Recuérdese que previamente habíamos dicho que se creaba un objeto inicial por cada partícula caminante con las propiedades de posición inicial y color

[XV] No se conoce el color o bien hasta este momento no es de importancia

2. Los renglones *4*, *5*, *6* y *7* son los puntos adyacentes a la partícula caminante
3. En la fila *3* que corresponde a la partícula caminante no se indica el color de dicha partícula, esto se debe meramente a que hasta el momento no habíamos necesitado conocer con precisión el color de la partícula caminante.
4. Por la posición que tiene la partícula caminante, se observa que se encuentra más cerca de la *semilla 1*, correspondiente a la estructura fractal *1* de *color 1*.
5. Supongamos que la partícula caminante es de *color 1*, entonces con un movimiento a la derecha o hacia arriba se encontraría en un lugar adyacente a la semilla de su mismo color y entonces se agregaría y por lo tanto aumentaría el tamaño de esta estructura fractal.
6. En el caso de que la partícula caminante fuera de *color 2*, también con un movimiento a la derecha o bien hacia arriba, estaría en una posición adyacente a la semilla de *color 1*, pero en este caso se causaría la destrucción de la partícula caminante por tratar de integrarse a una estructura fractal de diferente color.

Básicamente cuando la partícula caminante esta cercana a alguna estructura fractal, se realizan comparaciones de los puntos adyacentes a esta partícula con los puntos que integran cada estructura fractal, sin embargo ya en este punto también es necesario conocer con precisión el color de la partícula caminante, ya que esto finalmente es determinante para saber el futuro de la partícula.

Estas comparaciones de los puntos adyacentes a la partícula caminante con los puntos que integran las estructuras fractales, se realizan mediante comparaciones de índices entre las partículas que integran las estructuras fractales y la partícula caminante. Dichos índices se obtienen con la utilización de algoritmos de hashing numérico que realizan cálculos muy rápidos en tiempos básicamente constantes para arreglos de cualquier dimensión.

Como ya se mencionó, una vez que la partícula caminante está en un lugar adyacente a cualquiera de las estructuras fractales, se determina el futuro de esa partícula, ya sea integrándose a la estructura fractal o muriendo por haberse pegado una estructura de color diferente. De lo anterior derivan las siguientes dos posibilidades:

1. Sea que la partícula caminante murió por tratar de integrarse a una estructura fractal de diferente color, entonces se inicia nuevamente la generación de una nueva partícula caminante, iniciando con la obtención del color en forma aleatoria y respetando la probabilidad que de inicio fué parte de los datos del experimento proporcionados por el usuario. También se incrementan los contadores de partículas generadas por experimento en forma general y por color, para posteriormente ser utilizados en el análisis estadístico de resultados. Se crean en forma aleatoria las coordenadas iniciales correspondientes a la posición de la partícula caminante y se vuelve a crear un objeto con las propiedades de color, posición x y posición y, e inicia el ciclo de movimientos aleatorios.
2. Sí la partícula se integra a una estructura fractal de su mismo color, entonces se adhiere e incrementa el tamaño de ésta, el programa incrementa el número de partículas que integran dicha estructura. También compara el número de partículas que integran ahora la estructura fractal, con el número total de partículas que queremos que integren la estructura fractal, en caso de que el número de partículas que están integradas sea menor del que se pretende alcanzar, entonces se debe agregar este nuevo sitio a la estructura fractal y también a una tabla en el hashing numérico. También se deben calcular el radio máximo y su cuadrado en función de la posición de la nueva partícula que integra la estructura, y de estos valores calcular el radio y el radio de muerte respectivamente. Lo último que se realiza es verificar si alguna de las dos estructuras fractales ha completado el número de partículas solicitadas por el usuario, de ser así da por finalizado este experimento e inicia el siguiente o bien

termina el programa en caso de que también se hayan completado el número de experimentos solicitados. En el caso de que ninguna estructura haya completado el número de partículas solicitadas, se genera una nueva partícula caminante con las propiedades de sus coordenadas y color y se inicia nuevamente un ciclo como el que brevemente describimos en los párrafos anteriores.

Básicamente este modelo computacional de enumeración exacta realiza los pasos previamente narrados, arrojando datos cuantitativos que posteriormente deben ser analizados en forma profunda para entender el comportamiento de estos procesos, que si bien por el momento no sabemos si corresponden o no a algún proceso natural en particular, una vez se tenga el análisis detallado de los resultados, estos se podrán encausar a los procesos naturales que más se adapten.

4.3. MODELO COMPUTACIONAL

Este proyecto esta diseñado en lenguaje estándar C++, puede ser compilado y ejecutado en las plataformas Linux (g++), o bien en Windows (Borland C++ u otro compilador de C++), de acuerdo con el estándar preliminar ANSI/ISO.

El programa computacional es un proyecto el cual consta de los siguientes archivos fuente de usuario indicados en la tabla *4.3.1*

Archivos Fuente (*.cpp para Windows y *.cc en Linux)	**Archivos de Encabezado de la Biblioteca Estándar (< *.h >)**	**Archivos de Encabezado propios del programa (" *.h ")**
enum2d.cpp	iostream.h	cabeza.h
fract2d.cpp	Fstream.h	fractal.h
cabeza.cpp	Stdlib.h	hash2d.h
fractal.cpp	Math.h	define.h
define.cpp	Mem.h	spis2d.h
hash2d.cpp	Stdio.h	random.h
random.cpp	Time.h	screen2d.h
spis2d.cpp	Alloc.h	primo.h
screen2d.cpp		

Tabla 4 3 1

El archivo principal del proyecto, el cual incluye la función Main, es enum2d.cpp, el cual a su vez define los demás archivos *.cpp como unidades del proyecto. Por lo tanto, este archivo inicia la ejecución del programa.

En la siguiente sección se explicará como esta organizado a bloques este programa y se profundizará más a cerca de cada uno de ellos.

4.4. Diagrama de bloques del programa computacional

A continuación muestro se muestra un diagrama a bloques general de la estructura principal del proyecto (figura *4.4.1*)

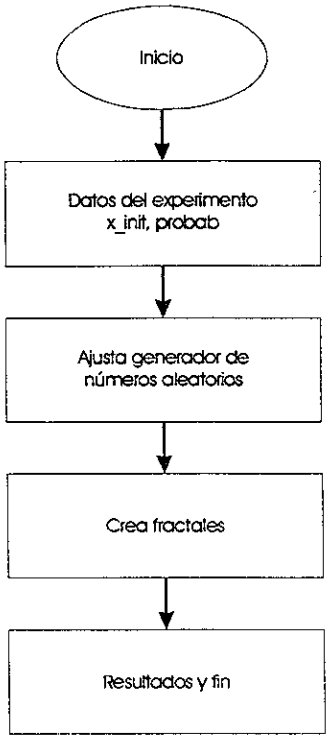


Figura 4.4.1 Diagrama a bloques general del modelo de enumeración exacta

El archivo principal del proyecto es enum2d.cpp, el cual incluye la función main e inicia la ejecución del programa.

El proyecto a su vez cuenta con los demás archivos fuentes mencionados en la tabla anterior.

1. Datos del experimento

En esta etapa se dá inicio a la ejecución del programa de enumeración exacta, solicitando al usuario que indique los siguientes datos;

a. Distancia de la semilla 1 al origen (*1 a 5*)[XVI]

b. Probabilidad de que las partículas caminantes sean de color 1 *(0 0 a 0 5)* [XVII]

[XVI] Con este dato, el programa calculara inmediatamente cual es la distancia entre semillas

[XVII] Recuerde que la probabilidad de que la partícula sea de color 1 es complementaria de que sea de color 2

2. Inicialización del Generador de Números Aleatorios

La mayoría de las implementaciones en C, tienen escondidas algunas rutinas para la inicialización y generación de números aleatorios. En ANSI C, se tiene:

```
#include <stdlib.h>
#define RAND_MAX ...
void srand(unsigned seed);
int rand(void);
```

La función *srand* toma un argumento entero *unsigned* (conocido como semilla) y siembra la función *rand* para que genere una secuencia diferente de números aleatorios cada vez que se ejecute el programa[XVIII]

Con instrucción *srand (time(0))*, la computadora lee su reloj, obteniendo de ahí el valor para la semilla. La función time con argumento 0, devuelve en segundos la hora calendario actual. Este valor se convierte a un entero sin signo y se utiliza como la semilla para el generador de números aleatorios.

Es importante mencionar que el mismo valor de semilla (*seed*), siempre regresara la misma secuencia aleatoria.

Las funciones que se ejecutan en esta etapa son las siguientes:

AjustaGenNumAleatorio( ), esta función se define en el archivo fractal.cpp, inicializa las variables *IntSeed*[XIX], *IntColor*[XX] y *IntTheta*[XXI], asignándoles un valor entero negativo aleatorio diferente a cada uno de ellos.

Como ejemplo veamos la variable *IntSeed*, a la cual se le asigna un número negativo aleatorio, obtenido de utilizar la función *random (num)*, donde *num* en este caso proviene de la función *time(0)*, dicha función entrega como resultado la cantidad de segundos que han ocurrido desde el 1° de enero de *1970* al momento donde se invoca esta función. Esto es muy interesante ya que asegura que el generador de números aleatorios siempre tendrá un valor de inicialización diferente, con lo cual las secuencias de números aleatorios siempre serán también diferentes.

```
random(time(0)); //Genera un numero aleatorio entre 0 y time(0)
IntSeed = -random(time(0))-1;
```

[XVIII] Un ***unsigned int*** se puede almacenar en cuando menos dos bytes de memoria. Un ***unsigned int*** de dos bytes puede tener valores no negativos del 0 al 65535. Un ***unsigned int*** de cuatro bytes sólo puede tener valores no negativos en el rango de 0 al 4294967295 La función ***srand*** tomo como argumento un valor ***unsigned int***.

[XIX] La variable ***IntSeed*** es el identificador de una variable externa declarada en el algoritmo de generación de números aleatorios ran3

[XX] La variable ***IntColor*** es el identificador de una variable externa declarada en el algoritmo de generacion de números aleatorios ran3

[XXI] La variable ***IntTheta*** es el identificador de una variable externa declarada en el algoritmo de generación de números aleatorios ran3

Cuando ya se tiene el valor aleatorio de *IntSeed*, se realiza el mismo procedimiento para *IntColor* y posteriormente para *IntTheta*, sin embargo entre cada una de estas operaciones se establecen retrasos intencionales. Estos retrasos aseguran que el valor aleatorio para cada una de estas variables proviene de la misma función *random(time 0)*, pero con diferente periodo. Esto es que el *time (0)* en cada para el cálculo de cada variable es diferente por dos cosas principalmente;

a. Porque la cantidad de segundos del 1° de enero de *1970* al momento de la ejecución del programa siempre es diferente

b. Porque entre la generación del número aleatorio para cada variable existen estos retrasos intencionales.

Es de observarse que estamos hablando de 3 variables que se inicializar y que se les conoce como *IntSeed*, *IntColor* e *IntTheta*. En el modelo bicolor DLA se requiere determinar mediante el generador de números aleatorios tres aspectos fundamentales que son:

1. Posición aleatoria cuando se forma la partícula caminante
2. El color que tiene la partícula caminante, el cual también debe estar determinado en forma aleatoria
3. El movimiento aleatorio de la partícula, donde para el caso de 2 dimensiones una partícula puede tener 4 movimientos (arriba, abajo, derecha o izquierda) y para el caso de 3 dimensiones la cantidad de movimientos ahora es de 6.

3. Crea Fractales

Como este programa fué diseñado en base a la tecnología orientada a objetos OOP, se manejan clases y funciones. Las funciones están compuestas por estructuras de control que realizan acciones y decisiones. Además en este programa se determinan las clases necesarias para los atributos que necesitarían tener los objetos de estas clases, los comportamientos de las clases y la manera de interactuar los objetos.

La función *CreaFractales* como veremos es la más compleja del programa y es invocada desde el archivo principal como sigue:

CreaFractales(X_Init,Probab);

En donde *X_Init* es la distancia de la semilla 1 al origen y *Probab* es la probabilidad de que la partícula caminante sea de Color 1.

A partir de la figura *4.4.2* muestra el diagrama a bloques general de la función Crea fractales, el cual también integra el bloque de resultados y fin referido en la figura *4.4.1*.

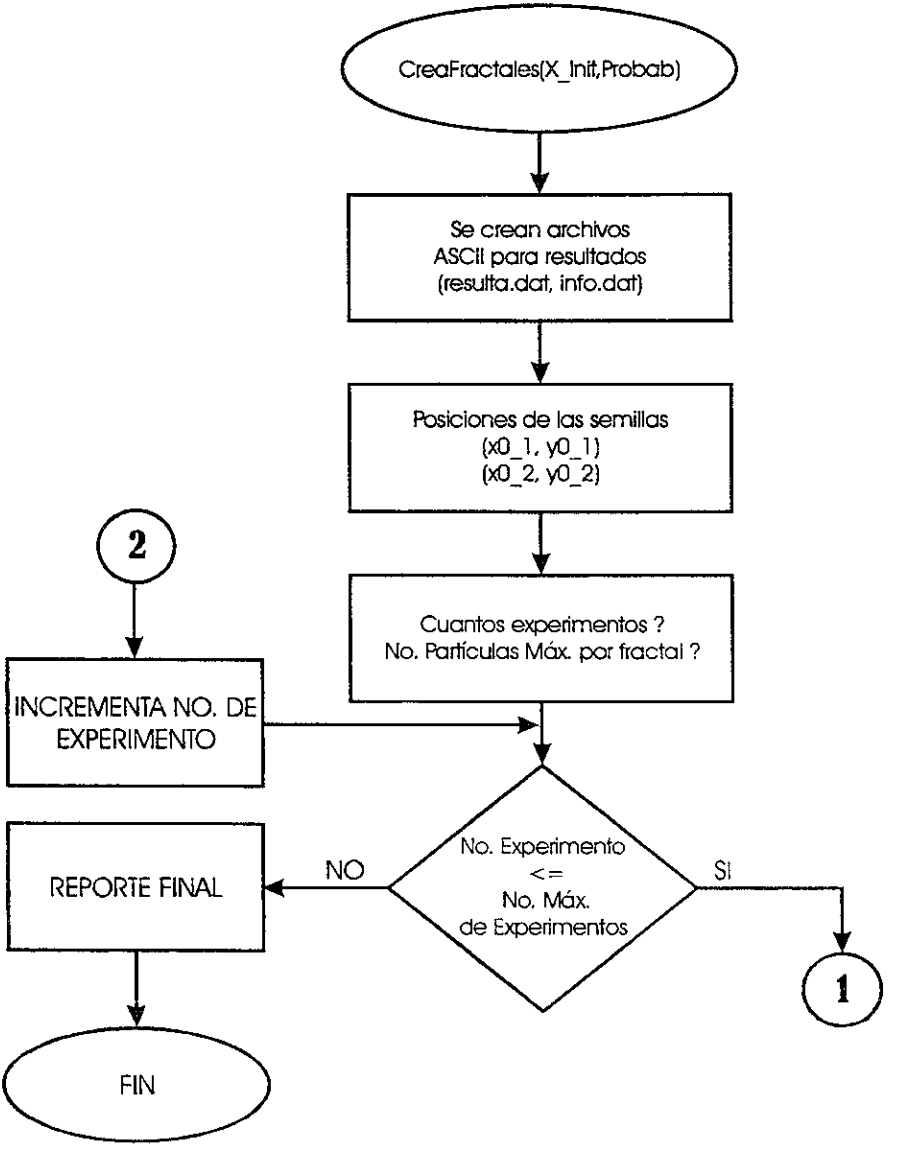

CreaFractales(X_Init,Probab)
Se crean archivos
ASCII para resultados
(resulta.dat, info.dat)
Posiciones de las semillas
(x0_1, y0_1)
(x0_2, y0_2)
Cuantos experimentos ?
No. Partículas Máx. por fractal ?
2
INCREMENTA NO. DE
EXPERIMENTO
No. Experimento
<=
No. Máx.
de Experimentos
NO
SI
REPORTE FINAL
1
FIN

Figura 4 4 2

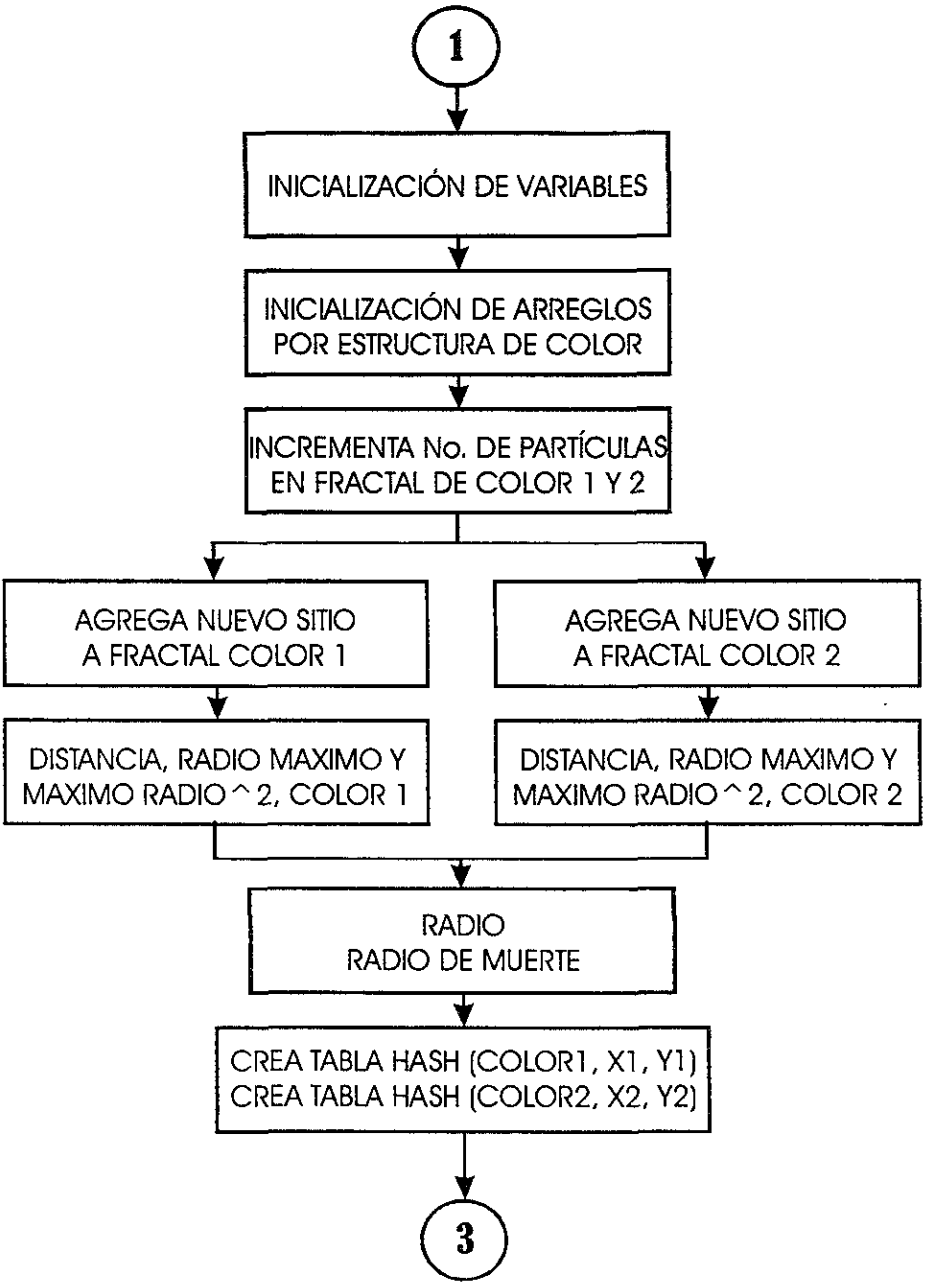

1
INICIALIZACIÓN DE VARIABLES
INICIALIZACIÓN DE ARREGLOS POR ESTRUCTURA DE COLOR
INCREMENTA No. DE PARTÍCULAS EN FRACTAL DE COLOR 1 Y 2
AGREGA NUEVO SITIO A FRACTAL COLOR 1
AGREGA NUEVO SITIO A FRACTAL COLOR 2
DISTANCIA, RADIO MAXIMO Y MAXIMO RADIO ^ 2, COLOR 1
DISTANCIA, RADIO MAXIMO Y MAXIMO RADIO ^ 2, COLOR 2
RADIO
RADIO DE MUERTE
CREA TABLA HASH (COLOR1, X1, Y1)
CREA TABLA HASH (COLOR2, X2, Y2)
3

Figura 4 4 3

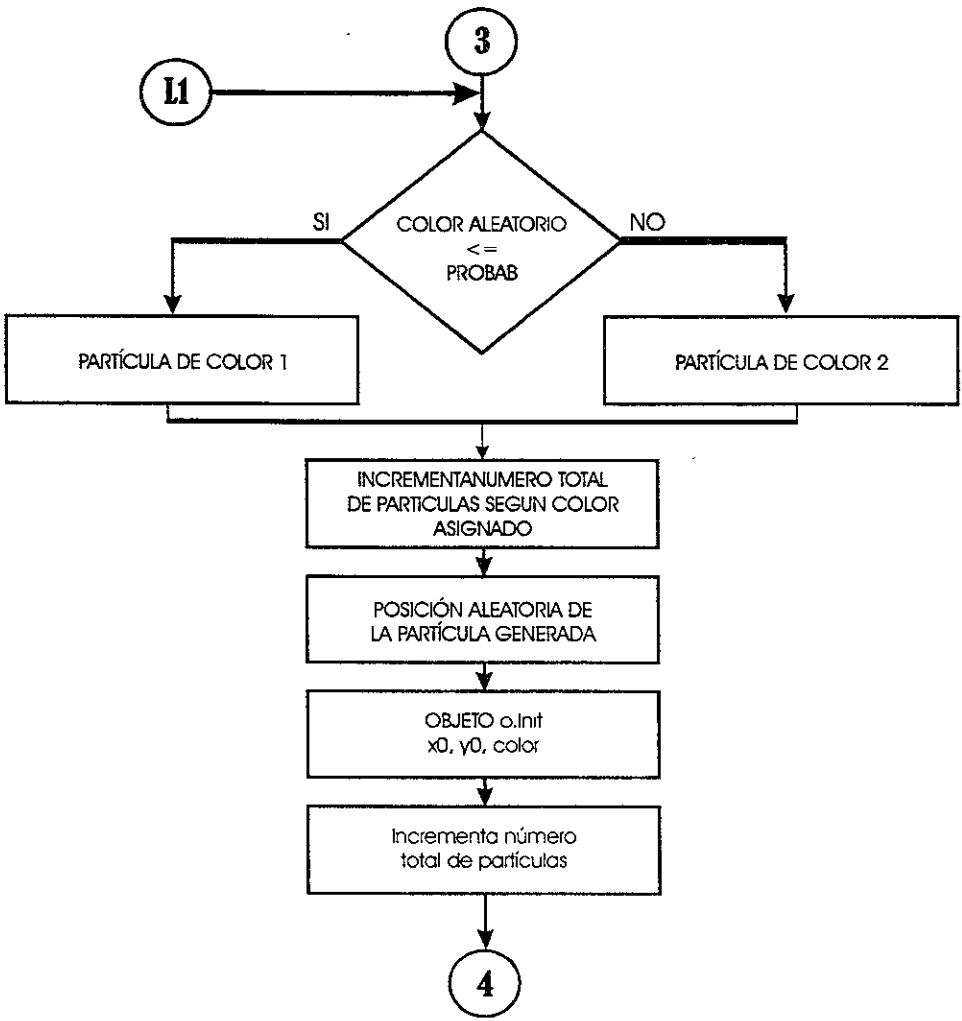

3
L1
SI
COLOR ALEATORIO
<=
PROBAB
NO
PARTÍCULA DE COLOR 1
PARTÍCULA DE COLOR 2
INCREMENTANUMERO TOTAL
DE PARTICULAS SEGUN COLOR
ASIGNADO
POSICIÓN ALEATORIA DE
LA PARTÍCULA GENERADA
OBJETO o.Init
x0, y0, color
Incrementa número
total de partículas
4

Figura 4 4 4

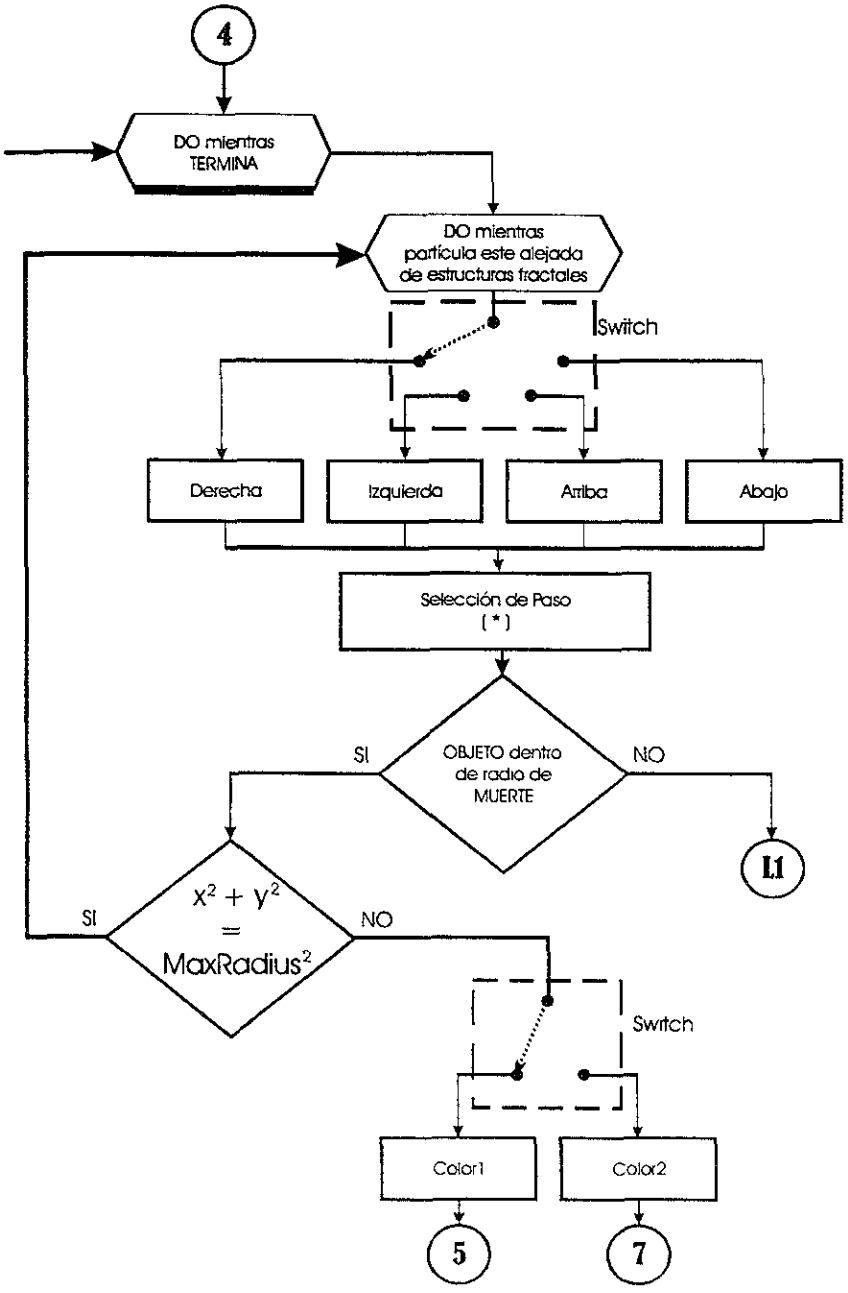

4
DO mientras
TERMINA
DO mientras
partícula este alejada
de estructuras fractales
Switch
Derecha
Izquierda
Arriba
Abajo
Selección de Paso
(*)
SI
OBJETO dentro
de radio de
MUERTE
NO
L1
SI
x² + y²
=
MaxRadius²
NO
Switch
Color1
Color2
5
7

Figura 4.1.5

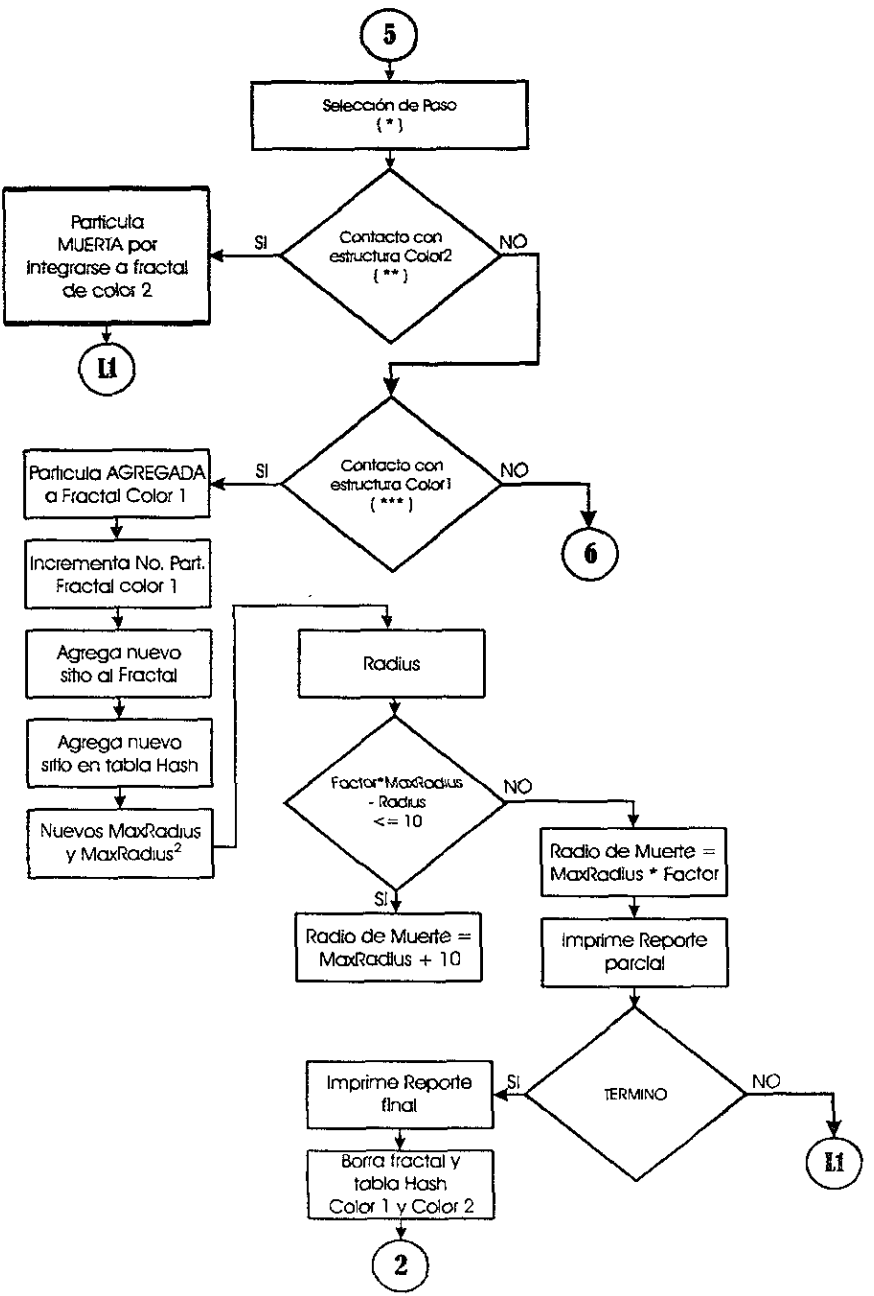

5
Selección de Paso
(*)
Particula
MUERTA por
integrarse a fractal
de color 2
SI
Contacto con
estructura Color2
(**)
NO
L1
Particula AGREGADA
a Fractal Color 1
SI
Contacto con
estructura Color1
(***)
NO
6
Incrementa No. Part.
Fractal color 1
Agrega nuevo
sitio al Fractal
Agrega nuevo
sitio en tabla Hash
Nuevos MaxRadius
y MaxRadius^2
Radius
Factor*MaxRadius
- Radius
<= 10
NO
SI
Radio de Muerte =
MaxRadius + 10
Radio de Muerte =
MaxRadius * Factor
Imprime Reporte
parcial
Imprime Reporte
final
SI
TERMINO
NO
L1
Borra fractal y
tabla Hash
Color 1 y Color 2
2

Figura 4 4 6

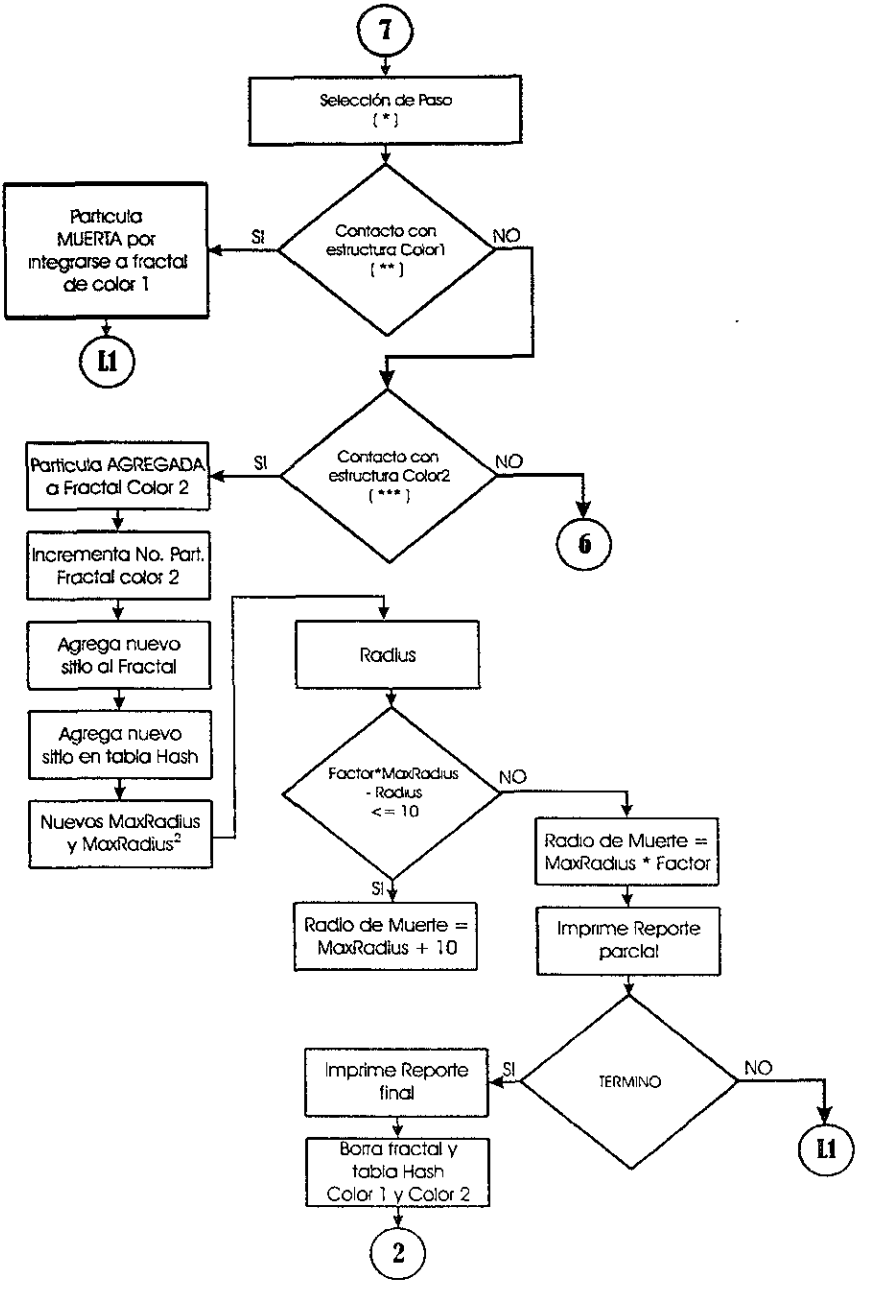

7
Selección de Paso
(*)
Particula
MUERTA por
integrarse a fractal
de color 1
SI
Contacto con
estructura Color1
(**)
NO
L1
Particula AGREGADA
a Fractal Color 2
SI
Contacto con
estructura Color2
(***)
NO
6
Incrementa No. Part.
Fractal color 2
Agrega nuevo
sitio al Fractal
Agrega nuevo
sitio en tabla Hash
Nuevos MaxRadius
y MaxRadius²
Radius
Factor*MaxRadius
- Radius
<= 10
NO
SI
Radio de Muerte =
MaxRadius + 10
Radio de Muerte =
MaxRadius * Factor
Imprime Reporte
parcial
Imprime Reporte
final
SI
TERMINO
NO
L1
Borra fractal y
tabla Hash
Color 1 y Color 2
2

Figura 4.1.7

4.5. Operaciones de Entrada y Salida a archivos en nuestro modelo computacional.

En el diagrama a bloques detallado del programa, fué necesario declarar clases para el procesamiento de *E/S* a través de archivos.

C++ proporciona el soporte para su sistema de *E/S* en el archivo de cabecera *iostream.h*, en este archivo se definen las jerarquías de clase para que admitan las operaciones de *E/S*.

Las jerarquías de clase con las que se trabaja normalmente derivan de la clase *ios*. Se emplea como una base para varias clases derivadas incluyendo *istream*, *ofstream*, *iostream*. Estas clases se utilizan para crear flujos que lleven a cabo *Entrada*, *Salida* y *Entrada / salida* respectivamente.

La clase *ios* contiene miembros y variables que controlan las operaciones fundamentales de un flujo, para tener acceso a esta clase, debe incluirse *iostream.h* en el programa.

Para realizar *E/S* en archivos debe incluirse en el programa el archivo de cabecera *fstream.h* .

En C++, un archivo se abre mediante el enlace a un flujo. Hay tres tipos de flujos: de *Entrada*, de *Salida* y de *Entrada / salida*. Antes que pueda abrirse un archivo debe obtenerse un flujo. Para crear un flujo de *Entrada*, este debe declararse de la clase *ifstream*. Para crear un flujo de *Salida*, este debe declararse de la clase *ofstream*.

Los flujos que realizan tanto operaciones de *Entrada* como de *Salida* deben ser declarados de la clase *fstream*.

Ejemplo:

- Ifstream *Archivo_Ent;*
- Ostream *Archivo_Sal;*
- Fstream *Archivo_Ent_Sal;*

Una vez creado un flujo, una forma de asociarle aun archivo es realizar la función *open()*, su prototipo es como sigue:

- Void open (const char**nombre*, int *modo*, int *acceso)*

nombre, es el nombre del archivo, el cual puede incluir un especificador de trayectoria o ruta donde se ubica dicho archivo.

modo, determina el modo de abrir un archivo. Su valor implícito ios::in para entrada (para *ifstream*) y ios::out para salida (para *ofstream*). Por omisión, todos los archivos se abren en modo texto. Pero, sí el valor de modo es ios::binary, dicho archivo será abierto en modo binario.

acceso, determina el modo de acceso al archivo, por omisión, acceso toma un valor que crea un archivo normal (no oculto, no del sistema y/o no de solo lectura).

Prácticamente nunca se verá una llamada a *open( )* con tres parámetros, porque los parámetros *modo* y *acceso* toman valores implícitos.

1. Un archivo normal de Salida

   Ofstream FileOut

   FileOut open ("test.dat"),

2. Un archivo normal de Entrada

   Ifstream Filen,

Filien.open ("test.dat");

3. Un archivo normal de Entrada / salida

Fstream FileInOut;

FileInOut.open("test.dat", ios::in | ios::out);

En esta situación, modo no toma ningún valor implícito.

Sí se produce un error en *open()*, el flujo será nulo. Antes de utilizar un archivo, sería necesario comprobar la operación de apertura:

```
ofstream Fout("datos.dat");
if (!Fout)
{
 cout << "Error al abrir archivo datos.dat" << endl;
 exit(1);
}
```

Para cerrar un archivo hay que emplear la función miembro *close()*:

Fout.close(),

La función *close()* no tiene parámetros, no devuelve ningún valor.

En el caso de nuestro modelo en particular se hace uso de la clase *ofstream* para las operaciones de salida de archivos. La clase *ofstream* utilizada en este programa pertenece a la biblioteca *iostream*, la cual maneja una amplia variedad de operaciones de *E/S*.

4.5.1. Entrada / salida a Consola

Aunque se pueden utilizar funciones como *printf()* y *scanf()*, C++ proporciona un método nuevo y mejor de realizar estas operaciones de *E/S*. En C++, La *E/S* se realiza usando operadores de E/S en vez de funciones de *E/S*.

El operador de salida es <<, y el operador de entrada es >>.

La forma general de salida por consola es:

cout << "expresión" ;

Aquí, expresión puede ser cualquier expresión válida de C++.

La forma general de entrada desde teclado

cin >> variable ;

Los operadores << y >> son ejemplos de sobrecarga de operadores

En la forma general de salida, se le indica a la computadora que imprima en la pantalla la cadena contenida entre comillas (expresión). La línea completa, incluyendo cout, el operador <<, la cadena "expresión" y el punto y coma (;), se le llama instrucción.

La salida y entrada en C++ se logra por medio de flujos de caracteres. Por lo tanto al ejecutarse la instrucción previa, envía el flujo de caracteres expresión al objeto de flujo de salida estándar, cout normalmente esta dirigido a la pantalla.

En la forma general de entrada desde el teclado se emplea el objeto de flujo de entrada cin y el operador de extracción >>, con el fin de solicitar al usuario introducir datos mediante el teclado.

En el programa se utilizan ambas formas generales para la entrada / salida a través de la consola (teclado y pantalla), con el fin de ingresar los datos de cada experimento y entregar visualmente los resultados de cada uno de ellos al usuario.

cout<<endl<<"Proporciona la Distancia de la semilla 1 al Origen (1 a 5) : ";

cin>>setw(10)>>X_Init;

Las dos líneas anteriores forman parte inicial del programa, en donde se le solicita al usuario a través de la pantalla ingresar el dato de la distancia de la semilla 1 al origen, y dicho valor será reconocido por la computadora a través de la variable X_Init.

Otros datos que se le solicitan al usuario son:

¿Probabilidad de que la partícula sea de color 1 (0 a 0.5) ?

¿Cuántos experimentos desea realizar (1 a 5000 máx.) ?

¿Número máximo de partículas por fractal (2 a 10 máximo) ?

Para cada una de estas preguntas en pantalla, se asocia una instrucción de entrada mediante el teclado para la introducción de estos datos solicitados.

4.6. Declaración de funciones

Este programa al igual que muchos otros programas grandes escritos en C++, ha sido desarrollado a partir de la construcción de pequeñas partes que facilitan su manejo en forma completa. Estos componentes pequeños los podemos dividir en funciones y clases.

Los programas en C++ generalmente se escriben combinando funciones nuevas que el programador escribe con las funciones disponibles en la biblioteca estándar de C++, y combinando clases nuevas que el programador escribe con clases disponibles en varias bibliotecas de clases.

La biblioteca de C++ contiene un vasto conjunto de funciones para realizar cálculos matemáticos, manipulación de cadenas, manipulación de caracteres, entrada / salida, comprobación de errores y muchas otras operaciones útiles. Las funciones de la biblioteca estándar de C++ son parte del entorno de programación que simplifican el trabajo del programador ofreciendo muchas de las capacidades que se necesitan.

El programador puede escribir funciones para definir ciertas tareas específicas que podrán utilizarse en muchas partes de su programa, dichas funciones son conocidas como funciones definidas por el programador. Estas funciones solo se escriben una sola vez y se ocultan de las demás funciones.

Se puede invocar a una función mediante una llamada de función. Esta especifica el nombre de la función y le proporciona información (en forma de argumentos) que necesita para trabajar.

Las funciones le permiten al programador modularizar sus programas. Todas las variables declaradas en la definición de función son variables locales, solo se conocen en la función en la que se definieron. La mayoría de las funciones tienen una lista de parámetros, que proporciona el medio para la comunicación de información entre las funciones. Los parámetros de una función también son variables locales.

C++ permite definir varias funciones con el mismo nombre, siempre y cuando tengan diferentes grupos de parámetros. Esta capacidad se llama sobrecarga de funciones. Cuando se llama a una función sobrecargada, el compilador de C++ selecciona la función adecuada examinando el número, tipos y orden de los argumentos de la llamada. La sobrecarga de funciones se utiliza normalmente

para crear varias funciones con el mismo nombre que realicen tareas similares, pero sobre distintos tipos de datos.

La reutilización de software en la programación orientada a objetos es una de las razones principales para la utilización de funciones ya existentes como bloques de construcción de nuevos programas.

La repetición de código dentro de un programa se evita con la utilización de funciones, ya que el código es empacado como función, y es posible ejecutarlo desde varios puntos del programa, simplemente llamando a la función.

Una de las características más importantes de C++ es el prototipo de la función. El prototipo de la función le dice al compilador el nombre de la función, el tipo de datos que ésta devuelve, el número, tipo y orden de los parámetros que dicha función espera recibir. El compilador utiliza sus prototipos de función para validar las llamadas de funciones.

En nuestro programa, se tiene la siguiente llamada a la función;

```
AjustaGenNumAleatorio( ) ;
```

Como se puede observar el prototipo de esta función carece del tipo de datos que espera recibir, así como del tipo de datos que se debe usar en el cuerpo de la función. Esto significa que el llamado a la función *AjustaGenNumAleatorio ( )*, no espera la devolución de ningún valor, solo la ejecución de una serie de cálculos y acciones que permitan inicializar tres variables que posteriormente se utilizaran con los algoritmos del RAN3. Esta función es llamada en el archivo enum2d.cpp, el cual es el archivo principal del proyecto enum2d.

La definición de la función se encuentra en el archivo fractal.cpp, y esta escrito como sigue;

```
void AjustaGenNumAleatorio( ) /* Ajusta el generador de números aleatorios */
{
 srand(time(0));  //Semilla para el generador de números aleatorios
 random(time(0)); //Genera un número aleatorio entre 0 y time(0)
 IntSeed=-random(time(0))-1;
 double kk;
 for (int k=1; k<=50000L;k++)
   {double kk=2.0*k*2.0;}   //Retrazo intencional
 random(time(0));
 IntColor=-random(time(0))-1,
 for (int k=1; k<=50000L;k++)
   {double kk=2.0*k*2.0;}   //Retrazo intencional
 random(time(0));
 IntTheta=-random(time(0))-1;
}
```

Una vez que es invocada la función *AjustaGenNumAleatorio ( )*, inicia la ejecución de las declaraciones e instrucciones que forman parte del cuerpo de la función. En el cuerpo de la función

anterior se tienen funciones anidadas, sin que por esto se tenga que definir una función dentro de otra.

Por ejemplo en primera línea del cuerpo de la función anterior, se tiene línea *srand(time(0));*, esta instrucción forma parte de dos funciones anidadas, que son srand y time. La función srand forma parte de la biblioteca estándar y genera números pseudoaleatorios, ya que llamadas repetidas de esta instrucción produciría una secuencia de números al parecer aleatorios. La función time forma parte de la librería time.h (ctime en el nuevo estándar de C++), esta función con el argumento 0, devuelve en segundos la hora calendario actual, como un entero sin signo y sirve como semilla para la función srand. Es importante mencionar nuevamente que aunque existan llamados a otras funciones anidadas dentro de la declaración de la función *AjustaGenNumAleatorio ( )*, las declaraciones de estás funciones anidadas se encuentran fuera de ésta.

4.7. Estructuras de control

El algoritmo de este proyecto representado en el diagrama de flujo, muestra la ejecución secuencial del programa, los símbolos de inicio y fin del programa se representan mediante un ovalo, los símbolos de conexión son representados por círculos pequeños con un número interno que indica la referencia de conexión.

Fácilmente podemos distinguir aquellos símbolos que involucran una decisión, estas estructuras son conocidas como estructuras de selección.

C++ ofrece tres tipos de estructuras de decisión; *if*, *if/else* y *switch*.

La estructura de selección *if* se llama estructura de selección única, pues selecciona o ignora una sola acción. La estructura de selección *if/else* se llama estructura de doble selección, pues selecciona entre dos acciones distintas. La estructura de selección *switch* se llama estructura de selección múltiple, dado que selecciona la acción a ejecutarse entre varias acciones.

Por otra parte C++ ofrece estructuras de repetición que son *while*, *do/while* y *for*.

4.7.1. Estructuras de selección ; if, if/else y switch.

El símbolo de la expresión *if* contiene una condición, que puede ser verdadera o falsa. Del símbolo de decisión emergen dos líneas de flujo. Una indica la dirección que se debe tomar cuando la expresión del símbolo es verdadera (true); la otra, la dirección que se debe tomar cuando la expresión es falsa (false).

Como ejemplo en nuestro proyecto, se tiene las siguientes instrucciones;

```
if (o.CheckCubContact(Color2) == Yes)
        {
        K_C2++, /*Incrementa número de Partículas de Color 1 que trataron de integrarse
        al fractal de Color 2 */
        cout<<"\nParticula de Color 1 MUERTA por integrarse a FRACTAL de
        COLOR2"<<endl,
        goto L1;
        }
if (o CheckCubContact(Color1) == Yes)
        {
```

```
cout<<"\nParticula Agregada a Fractal de color 1 en Las Coordenadas\t( "<<
o.GetXPos()<<", "<<o.GetYPos()<<" )"<<endl;
NumParticlesInFractal [Color1]++;
}
```

Estas instrucciones forman parte del diagrama a bloques de la figura *4.4.6*, en la cual se muestran dos estructuras *if* anidadas (romboides conectados), los cuales toman una decisión al evaluar si la partícula en cuestión ha hecho contacto con alguna estructura fractal.

El primer *if* evalúa si un objeto (partícula con propiedades de color 1, posición en x y posición en y), se encuentra en una posición adyacente a la estructura fractal de color 2, en caso afirmativo se ejecuta el siguiente grupo de instrucciones que se encuentra entre llaves, caso contrario no evaluaría este grupo de instrucciones e iniciaría la siguiente estructura de decisión *if*.

En segundo *if* del ejemplo anterior evalúa nuevamente el objeto, pero ahora para saber si se encuentra en una posición adyacente a la estructura fractal de color 1, de ser afirmativo se puede observar que indicara esto a través de la pantalla mediante el mensaje *Partícula Agregada a Fractal de color 1 en Las Coordenadas....* (coordenadas en las cuales se encuentre el objeto).

La estructura de selección *if/else* permite indicar que se realizaran acciones diferentes cuando la condición sea verdadera y cuando sea falsa.

Del diagrama a bloques del programa, en la figura *4.4.5* se tiene un bloque denominado selección de paso, veamos parte de las instrucciones que integran esta acción;

```
void Cub_Step::ChooseStep()
{
 double w=sqrt(sqr((double)Pos.x)+sqr((double)Pos.y)),
 if (w < MaxRadius+10) Step=1;
 else
   if (w < MaxRadius+20) Step=2;
     else
       if (w < MaxRadius+40) Step=4;
         else
           if (w < MaxRadius+80) Step=8;
           else Step=16,
}
```

La primera instrucción se denomina función miembro de la clase Cub_Step, posteriormente ahondaremos más en el tema de clases.

Cuando inicia la ejecución de esta función miembro, primeramente se evalúa *w* como sigue;

$$w = \sqrt{\left(x^2 + y^2\right)}$$

con este valor de *w* se inicia una serie de if/else anidados, supongamos que el valor de *w* es *18* y el valor de *MaxRadius* es *6*, se tiene

1er if/else	*(w < MaxRadius+10)*	18 < 16	No se cumple
2° if/else	*(w < MaxRadius+20)*	18 < 26	Sí se cumple

Como en el primer if la desigualdad es falsa, entonces se realiza una segunda acción que dá inicio a otra estructura *if/else*, en la cual vuelve a existir una desigualdad que en este caso sí se cumple y que da por finalizado este ciclo de if/else, devolviendo en este caso *Step = 2*. Esto prácticamente nos indica que la posición del objeto (partícula) se encuentra muy alejada, por lo tanto el paso con el que debe moverse debe ser más grande, para con esto aumentar la eficiencia del programa.

Las estructuras *if/else* anidadas en este ejemplo son mucho más rápidas que una serie de estructuras *if* de decisión, pues existe la posibilidad de salir con rapidez de la estructura una vez que se satisfacen las condiciones.

La estructura switch es de selección múltiple, con frecuencia algún algoritmo contiene una serie de decisiones en las que se prueba por separado una variable o expresión para cada uno de los valores enteros que pueda tomar, efectuando diferentes acciones según sea el caso. C++ ofrece esta estructura para manejar tal toma de decisiones.

La estructura *switch* contiene una serie de etiquetas *case* y un caso opcional *default*. Veamos el siguiente ejemplo;

```
switch (RandInt3(IntSeed,4)) //Valor aleatorio en decimal multiplicado por 4
 {
  case 0 : o.MoveXP();
  cout<<"Derecha  \t"; break;
  case 1 : o.MoveXM();
  cout<<"Izquierda \t"; break;
  case 2 : o.MoveYP();
  cout<<"Arriba   \t"; break;
  case 3 : o.MoveYM();
  cout<<"Abajo    \t"; break;
 }
```

Después de la palabra *switch* se encuentra entre paréntesis *(RandInt3(IntSeed,4))*,. A esto se le llama estructura de control. El valor de esta expresión se compara con cada una de las etiquetas *case*. Sí sucede una coincidencia con cada una de las estructuras *case*, se ejecutan las instrucciones correspondientes a dicho *case* y se sale de inmediato de la estructura *switch* por medio de la instrucción *break*. A diferencia de otras estructuras de control, no es necesario encerrar entre llaves los *case* de varias instrucciones.

La instrucción *break* provoca que el control del programa pase a la primera instrucción después de la estructura *switch*. La instrucción *break* se utiliza porque de otra forma los *case* de una instrucción *switch* se ejecutarían juntos. Si no se indica *break* en ninguna parte de la estructura *switch*, cada vez que suceda una coincidencia en la estructura, se ejecutarán las instrucciones de los *case* restantes. Si no sucede ninguna coincidencia se puede incluir un caso *default* (opcional), que a la vez puede ejecutar alguna instrucción que envié un mensaje de error.

En el caso de ejemplo anterior, la estructura de control *(RandInt3(IntSeed,4))*, es una función de nombre *RandInt3*, cuyos argumentos son *(IntSeed,4)*. La tarea de esta función es devolver un valor entero aleatorio entre 0 y 3.

En este caso la función *RandInt3*, proporciona la información necesaria para trabajar en forma de argumentos. Esta función se encuentra definida en el archivo *random.cpp*, como sigue;

```
int RandInt3(int &idum, int IMax)
{
 return (int)floor(ran3(idum)*IMax);
}
```

Esta función regresa un valor entero, que se obtiene de multiplicar un valor aleatorio (obtenido de aplicar la función *ran3 (idum)*) por *Imax* (en esta caso 4), al resultado de este producto se le aplica su piso del número con lo que se obtiene un valor entero completamente aleatorio entre 0 y 3, dicho valor es regresado a la estructura de control *switch* para ser comparado con cada uno de los *case*.

Como se puede observar los únicos valores posibles que regresa la función *RandInt3*, son 0, 1, 2 y 3. Dichos valores proporcionan a su vez las únicas 4 posibilidades de dirección de movimiento de la partícula caminante (derecha, izquierda, arriba y abajo respectivamente en el espacio de dos dimensiones[XXII]).

El valor entero aleatorio que se compara solo coincidirá en un *case*, en ese momento se puede observar simplemente del ejemplo que:

- Se ejecuta la primera instrucción del *case* coincidente sobre un miembro de la clase *Cub_Step*, por el momento no ahondaremos en una explicación detallada de esto, simplemente pensaremos en que se realiza una acción a un objeto (partícula caminante).
- Se emitirá un mensaje indicando la dirección de movimiento y después se aplicará la instrucción *break* con la que saldrá de la estructura de control *switch*, y continuará la siguiente instrucción del programa.
- Se rompe con la estructura *switch* mediante la palabra *break*

4.7.2. Estructuras de repetición while, do/while y for.

La estructura de repetición *while* permite especificar una acción que debe repetirse mientras cierta condición permanece verdadera

```
while (tmp != NULL)
  {
   if (memcmp((void*)tmp,(void*)&Position,sizeof(TPosition)) == 0)
   {
    return True;
   }
   tmp=tmp->Next,
  }
```

[XXII] Esto puede ser aplicado en un espacio de tres dimensiones, simplemente multiplicandolo por 6, las cuales serían sus posibilidades de movimiento (derecha, izquierda, arriba, abajo, al frente y atras)

```
{
cout<<"\nDESTRUCCION de la Particula por colocarse"<<
"\njunto a la estructura de diferente color"<<endl;
return False;
}
```

En el pseudocódigo anterior se establece que mientras se cumpla la condición de que la variable *tmp* sea diferente de *NULL* (nulo), se debe ejecutar el siguiente bloque de llaves (línea *2* a *9*). Cuando esta condición ya no se cumpla, entonces se ejecutaran las últimas *4* líneas, emitiendo con esto el mensaje "Destrucción de la partícula por colocarse junto a la estructura de diferente color" y regresa un *false*.

La instrucción que se ejecutan mientras la condición *while* se cumple, es una instrucción de selección *if*, cuya condición emerge de una comparación de dos bloques de memoria.

La función *memcmp* es una función de biblioteca para el manejo de cadenas y comparación de bloques de memoria.

```
Memcmp (*s1, *s2, sizeof n )
```

La función *memcmp* compara el número especificado de caracteres de su primer argumento con los caracteres correspondientes de su segundo argumento. La función devuelve un valor mayor que *0* si el primer argumento es mayor que el segundo, devuelve 0 si los argumentos son iguales, y devuelve un valor menor que *0* si el primer argumento es menor que el segundo argumento.

La estructura de repetición *do/while* es parecida a la instrucción *while*. En esta última, la condición de ciclo se prueba al inicio del ciclo, antes de que se ejecute el cuerpo. La estructura *do/while* prueba la condición de continuación del ciclo después de que se ejecuta el cuerpo del ciclo; por lo tanto, el cuerpo se ejecutará cuando menos una vez. Cuando termina un *do/while*, la ejecución prosigue en la instrucción que esta después del *while*.

```
do
{
instrucciones
}
while (condición),
```

Veamos nuevamente parte del código utilizado en la sección de generación fractales

```
do
        {
        // Genera la dirección de un movimiento
        switch (RandInt3(IntSeed,4)) //Valor aleatorio en decimal multiplicado por 4
          {
            case 0 : o MoveXP();
            cout<<"Derecha  \t"; break.
            case 1   o.MoveXM().
```

```
        cout<<"Izquierda \t"; break;
        case 2 : o.MoveYP();
        cout<<"Arriba   \t"; break;
        case 3 : o.MoveYM();
        cout<<"Abajo    \t"; break;
      }
      if (o.CubInSphere() == No) goto L1; //Cuando se rebasa el radio de muerte
    }
    while (CheckMaxDistance(o.GetXPos(),o.GetYPos()) ==TRUE);
```

En este ejemplo, se esta indicando que las instrucciones del *do*, encerradas en llaves y que se refieren a dos estructuras de control de selección *if*, y *switch*, se estarán ejecutando mientras la condición del *while* no sea verdadera.

Básicamente en este ejemplo se genera un número aleatorio entre *0* y *1.0*, dicho numero es multiplicado por *4*, el resultado obtenido se le aplica el piso, para obtener el número entero inferior próximo. Dicho número nos dará la dirección de movimiento de la partícula que previamente estaba en una localización aleatoria dentro de una área de operación. Después de cada movimiento se encuentra el *if*, el cual estará verificando que con los movimientos de la partícula, está siempre se encuentre dentro de un radio calculado en función del tamaño de la estructura fractal formada. De tal forma que cuando dicha partícula sale de este radio, entonces se ejecuta el *goto* enviando la siguiente instrucción a la posición *L1*. *L1* inicia con la generación de una nueva partícula caminante y se registra que la partícula anterior ha muerto por salirse el radio de muerte.

La estructura de repetición *for* se encarga de todos los detalles de la repetición controlada por un contador.

El formato general de la estructura *for* es

for (expresión1, expresión2, expresión3)

Instrucción

Donde la *expresión1* inicializa la variable de control de flujo, *expresión2* es la condición de continuación del ciclo y *expresión3* incrementa la variable de control.

Si *expresión1* del encabezado de la instrucción *for* define la variable de control (es decir, si se especifica el tipo de la variable de control antes del nombre de la variable), la variable de control sólo puede utilizarse dentro del cuerpo de la estructura *for*, es decir, el valor de la variable de control será desconocido fuera de la estructura *for*. Este uso restringido del nombre de la variable de control se conoce como alcance de la variable. El alcance de la variable define dónde puede utilizarse en un programa.

A veces *expresión1* y *expresión3* son listas de expresiones separadas por comas. Las comas sirven aquí como operadores de coma que garantizan que las listas de expresiones se evalúen de izquierda a derecha. El operador de coma tiene la menor precedencia de todos los operadores de C++. El valor y tipo de una lista de expresiones separadas por comas es el valor y tipo de la expresión de la derecha de la lista. El uso más frecuente del operador de coma es en las estructuras *for*. Su principal aplicación es permitirle al programador emplear varias expresiones de inicialización y/o varias expresiones de incremento. Por ejemplo, puede haber varias variables de control en una misma estructura *for* que habrá que inicializar e incrementar.

Las tres expresiones de la estructura *for* son opcionales. Si se omite la expresión2, C++ supone que la condición de continuación de ciclo es verdadera, creando así un ciclo infinito. La expresión1 podría omitirse si la variable de control se inicializa en otra parte del programa. La expresión3 podría omitirse si el incremento se calcula por medio de instrucciones en el cuerpo del *for* o si no se necesita un incremento,

A veces se pone un punto y coma después de un encabezado *for* para crear un ciclo de retardo. Tal ciclo *for* con un cuerpo vacío efectúa la cantidad indicada de ciclos sin hacer nada más que contar. En ocasiones esto sirve para efectuar ciclos de retardo para disminuir la velocidad de un programa.

```
for (int n=1; n <= NumMaxExp; n++)
{
  NumParticulasEnFractal [Color1]=0;
  ..... /* Sección de inicialización de arreglos y variables */
L1:
  if (ran3(IntColor) < Probab)
    CubColor=Color1;
  else
    CubColor=Color2;
  ..... /* Sección de calculo de posición aleatoria de la partícula caminante */
  o.Init(X0,Y0,CubColor); //Se crea objeto con propiedades de pos.x, pos.y y Color
  NumTotalParticulas++;
    do   //Hasta Finalizar
    {
      do
      {
      switch (RandInt3(IntSeed,4)) //Valor aleatorio en decimal multiplicado por 4
        {
          case 0 : o.MueveXP(),
          case 1 : o.MueveXM();
          case 2 : o.MueveYP(),
          case 3 : o MueveYM();
        }
        if (o.CubInSphere() == No) goto L1; //Cuando se rebasa el radio de muerte
      }
      while          (VerificaDistanciaMaxima(o VerificaPosicionX(),o VerificaPosicionY())
       ==TRUE);
      switch (o GetColor())
```

```
{
  case Color1 :
  {
    if (o.VerificaContacto(Color2) == Yes) goto L1;
    if (o.VerificaContacto(Color1) == Yes)
    {
      NumParticulasEnFractal [Color1]++;
      .... /* sección de agregación a la estructura de color 1 y cálculos de las siguientes
    condiciones del experimento */
      if (Termina() == False) goto L1;
    }
    break;
  }
  case Color2 :
  {
    if (o.VerificaContacto(Color1) == Yes goto L1;
    if (o.VerificaContacto(Color2) == Yes)
    {
      (NumParticulasEnFractal [Color2])++;
      ... /* sección de agregación a la estructura de color 2 y cálculos de las siguientes
    condiciones del experimento */
      if (Termina() == False) goto L1;
    }
    break;
  }
 }

}
while (Termina()==False);
BorraFractal(Fractal[Color1]),
BorraFractal(Fractal[Color2]),
BorraTablaHash(Color1),
BorraTablaHash(Color2),
}
```

El código anterior representa la parte sustancial del algoritmo para la creación de las estructuras fractales, todo la sección esta delimitado por un ciclo *for* cuya variable de control *n* dependerá del número máximo de experimentos indicados por el usuario, de tal forma que si la cantidad de experimentos a realizar son 5, entonces se efectuaran todas las instrucciones encerradas en la llave de *for* 5 veces, y en cada una de ellas por ejemplo se crearan las tablas hash de cada experimento y posteriormente al finalizar la prueba, estas serán borradas como se muestra en las últimas líneas.

4.7.3. Arreglos

Un arreglo es un grupo consecutivo de localidades de memoria que tienen el mismo nombre y el mismo tipo. Para referirnos a una localidad o elemento particular de un arreglo, especificamos el nombre del arreglo y el número de posición del elemento particular dentro de éste.

El primer elemento de cualquier arreglo es el elemento cero. Por lo tanto nos referimos al primer elemento del arreglo como *array[0]*, al segundo elemento como *array[1]*, y así sucesivamente hasta el *i-esimo* elemento del arreglo como *array[i+1]*.

El número de posición entre corchetes se conoce como subíndice. Los subíndices deben ser enteros o expresiones enteras. Sí se emplea como subíndice una expresión, entonces ésta se evalúa para determinar el subíndice.

Los arreglos ocupan espacio en memoria. Se debe especificar el tipo de cada elemento y el número de elementos requerido para cada arreglo, de modo que el compilador reserve la memoria necesaria.

Los arreglos pueden declararse para que contengan otros tipos de datos. Por ejemplo, es posible utilizar un arreglo del tipo char para que almacene una cadena de caracteres.

Es conveniente cuando se usan arreglos, inicializar los valores de sus elementos a 0, muy frecuentemente mediante la estructura de control for seguido con una declaración de asignación a 0.

```
for (int k=0; k<=NumeroMaximoParticulas; k++)
{
    (*Lugar)[k].x=0,   // Inicializa a cero
    (*Lugar)[k].y=0, // Inicializa a cero
}
```

En el código anterior se utiliza la variable constante *k*, previamente definida como una constante de tipo entero que permite especificar el tamaño del arreglo *NumeroMaximoParticulas*, que hace que este programa sea más escalable en función del dato proporcionado por el usuario para determinar el número máximo de partículas que deben integrar alguno de los dos fractales en el problema de este modelo.

Para pasar un arreglo como argumentos de una función, se debe especificar el nombre del arreglo sin corchetes, por ejemplo la llamada de función

CreaFractales (Array, 20),

Le pasa el arreglo *Array* de tamaño *20* a la función *CreaFractales*. Al pasar un arreglo a una función, generalmente también se pasa su tamaño, con el fin de que la función pueda procesar la cantidad especifica de elementos que contiene.

4.7.4. Apuntadores

Dos maneras de invocar funciones en muchos lenguajes de programación son mediante llamada por valor y llamada por referencia. Cuando se pasa un argumento utilizando una llamada por valor, se hace una copia del argumento, la cual se pasa a la función llamada. Los cambios a la copia no afectan el valor de la variable original en el invocador. Con esto se evitan efectos secundarios.

Una de las ventajas de la llamada por valor es que, si se está pasando un elemento de datos grande, su copia puede tardar un tiempo de ejecución considerable.

Un parámetro de referencia es un alias de su argumento correspondiente. Para indicar que el parámetro de una función se pasa por referencia, simplemente se pone un ampersand (**&**) después del tipo del parámetro en el prototipo de la función. Se utiliza esta misma convención para listar el tipo de parámetros en el encabezado de la función.

La llamada por referencia es buena por cuestiones de desempeño, pues elimina la sobrecarga generada por copiar grandes cantidades de datos.

Una llamada por referencia puede afectar la seguridad, pues la función llamada puede alterar los datos del invocador.

Las variables de apuntador contienen direcciones de memoria como sus valores. Normalmente las variables contienen valores específicos. Por otra parte, los apuntadores contienen direcciones de variables que contienen valores específicos. En este sentido, los nombres de variables hacen referencia directa a un valor y los apuntadores hacen referencia indirecta a un valor. La referencia a un valor a través de un apuntador se llama indirección.

Los apuntadores, como cualquier otra variable, se deben declarar antes de utilizarlos, ya sea al declararlos o mediante una instrucción de asignación. Un apuntador puede inicializarse a *0*, a *NULL* o a una dirección. Cada variable que se declara como apuntador debe ir precedida por un asterisco (*).

El & u operador de dirección, es un operador unario que devuelve la dirección de su operando.

El * u operador de indirección devuelve un sinónimo, alias o apodo hacia el que apunta su operando (es decir, su apuntador).

Los apuntadores, como las referencias, también pueden servir para modificar una o más variables del invocador o para pasar apuntadores a objetos de datos grandes, evitando la sobrecarga de pasar los objetos mediante llamada por valor.

En C++, los apuntadores y el operador de indirección pueden utilizarse para simular llamadas por referencia. Al llamar una función con argumentos que deben ser modificados, se pasa la dirección de los argumentos. Por lo general, esto se logra aplicándole el operador de dirección (&) al nombre de la variable que se habrá de modificar. Cuando se pasa la dirección de una variable a una función, se puede emplear el operador de indirección (*) en la función, creando un sinónimo o alías del nombre de la variable, con esto a su vez es posible modificar el valor de la localidad de memoria del invocador (si la variable no esta declarada como const).

Una función que recibe como argumento una dirección debe definir un parámetro de apuntador para recibir dicha dirección.

```
void MakeFractalArray(TPtrArreglo &Lugar)
{
  Lugar=(TPtrArreglo) new Array.
```

```
if (Lugar == NULL)
{
        cout << "Error de asignación dinámica de memoria";
        exit(EXIT_FAILURE);
}
for (int k=0; k<=MaxNumParticles; k++)
{
        (*Lugar)[k].x=0;   // Inicializa
        (*Lugar)[k].y=0;
}
}
```

TPtrArreglo es un alías del tipo de datos de arreglo, el cual fué definido previamente como sigue

typedef Array TPtrArreglo;*

En el ejemplo anterior la variable la tipo *Lugar* es una referencia del tipo *TptrArreglo* (array), la cual es utiliza para la asignación dinámica de memoria, de tal forma que primeramente se establece un espacio de memoria y en caso de que este espacio no exista, se emitirá una indicación de falla en la asignación de memoria. En caso contrario se inicializaran a 0 los espacios correspondientes a la cantidad máxima de partículas disponibles en el arreglo.

Algunas variables del tipo *TPtrArreglo* utilizadas en diferentes archivos fuentes del programa, se identificaron como variables globales de alcance externo, como se muestra en la siguiente línea de código.

extern TPtrArreglo Fractal [2],

Este tipo de variables está disponible para cualquier función definidas en el proyecto, de tal forma que el especificador de la clase de almacenamiento *extern*, le indica al compilador que la variable *Fractal[2]* del tipo *Tptrarreglo*, está definida posteriormente en el mismo archivo o en uno diferente.[xxiii]

Como sabemos los arreglos son tipos de datos agrupados que almacenan elementos de datos relacionados del mismo tipo bajo el mismo nombre. Una estructura es capaz de almacenar elementos de datos relacionados de distintos tipos bajo un mismo nombre. Al invocar una función con un arreglo como argumento, éste se pasa de manera automática a la función simulando una llamada por referencia. Sin embargo, las estructuras siempre se pasan mediante llamada por valor (se pasa una copia de toda la estructura). Esto requiere la sobrecarga en tiempo de ejecución que es causada por copiar todos los elementos de datos de la estructura y almacenarlos en las pilas de llamadas de función de la computadora (que es el lugar donde se almacenan las variables locales empleadas en la función mientras ésta se ejecuta). Cuando hay que pasar una estructura a una función, puede utilizarse un apuntador hacia datos constantes (o una referencia hacia datos constantes), logrando el desempeño de una llamada por referencia y la protección de una llamada

[xxiii] Las variables globales incrementan su desempeño debido a que cualquier función las puede acceder directamente, eliminando con esto la sobrecarga del paso de datos a funciones

por valor. Cuando el apuntador se pasa hacia una estructura, sólo hay que copiar la dirección en la que se almacena dicha estructura.

4.8. Programación Orientada a Objetos (POO)

El programa utilizado en este trabajo para la realización de las simulaciones computacionales del modelo de enumeración exacta, esta escrito en C++, lo cual permite utilizar la tecnología orientada a objetos para la creación de las estructuras fractales.

La programación orientada a objetos se basa en:

- Clases
- Objetos
- Instancia de variables
- Métodos

La naturaleza del problema de agregación limitada por difusión, se adapta para aplicar la programación orientada a objetos, debido a que cada partícula puede ser tratada como un objeto y se pueden presentar diversas características.

En términos simples, la POO es una programación con objetos, los objetos son una extensión del concepto de registro. Los objetos nos permiten combinar datos y código dentro de paquetes. Un objeto es un lenguaje constructor que enlaza datos con funciones y estas funciones realizan alguna función sobre el dato. Los objetos pueden ser extendidos para incorporar nuevos elementos de datos y funciones, usando adecuadamente la propiedad de *herencia*.

Los objetos contienen código y datos, por lo cual éstos se parecen a programas miniaturas que permiten la construcción de objetos más complejos.

Una clase es un modelo usado para definir un objeto. La clase se puede reutilizar muchas veces para crear varios objetos de la misma clase. Las clases tienen la propiedad de ocultamiento de la información. Esto significa que, aunque los objetos de clase podrían saber como comunicarse entre ellos por medio de interfaces bien definidas, a las clases normalmente no se les permite conocer cómo se implementan otras clases. Los detalles de implementación están ocultos dentro de las clases mismas.

La programación de C++ se concentra en la creación de tipos definidos por el usuario, llamados clases. Cada clase contiene datos, así como el conjunto de funciones que los manipulan. Los componentes de datos de una clase se llaman datos miembro (o métodos en otros lenguajes orientados a objetos). Así como a una instancia de un tipo integrado, como int, se le llama variable, a una instancia de un tipo definido por el usuario (es decir, una clase) se le llama objeto. En C++ el enfoque es sobre clases, no sobre funciones.

Una clase contiene dos tipos de componentes: variables de clase y métodos. Una variable de clase sirve como un elemento de datos, y un método es una función. En un sentido, las variables de clases definen el estado interno del dato de un objeto. Los métodos definen al comportamiento del objeto, esto es, las acciones que un objeto puede realizar.

En C++ el termino dato miembro es usado para referirse a la variable de clase y el término funciones miembro será usado para referirse a métodos.

La POO muestra tres propiedades recurrentes

- Encapsulamiento

- Herencia
- Polimorfismo

El Encapsulamiento es un mecanismo que agrupa código y datos y los mantienen protegidos contra alguna interferencia o mal uso. El Encapsulamiento proporciona dos características importantes:

Pone datos y funciones sobre el mismo techo

Proporciona capacidades para ocultar datos.

El Encapsulamiento protege los datos de objetos, pues usualmente se accede a los datos a través de funciones que están definidas en el objeto, hace más fácil usar los datos de objetos y se puede utilizar para ocultar los detalles de la manera como los datos son almacenados o implementados.

La herencia es el proceso mediante el cual un objeto puede adquirir las propiedades de otro. La herencia nos permite derivar una nueva clase de una clase ya existente, lo cual nos permite:

- Agregar datos y código a una clase, sin tener que cambiar la clase original
- Reutilizar código
- Cambiar el comportamiento de una clase

Se puede usar la herencia para crear múltiples objetos que desarrollen nuevas acciones, aunque los objetos sean derivados del mismo bloque.

El polimorfismo es la cualidad que permite que un nombre se utilice para dos o más propósitos relacionados, pero técnicamente diferentes. El propósito del polimorfismo aplicado a la POO es permitir usar un nombre para especificar una clase general de acciones.

Un aspecto importante del polimorfismo, es que cada objeto en la familia puede tener métodos con los mismos nombres, pero el código para cada método del objeto puede ser enteramente diferente.

Las clases permiten que el programador modele objetos que tienen atributos (representados como datos miembro) y comportamiento u operaciones (representados como funciones miembro). En C++, los tipos que contienen datos miembro y funciones miembro se definen mediante la palabra clave class.

En algunos lenguajes de programación orientados a objetos las funciones miembro se conocen como métodos y se invocan en respuesta a mensajes enviados a un objeto. Un mensaje corresponde a la invocación de una función miembro enviada de un objeto a otro o de una función a un objeto.

Una vez que se ha definido una clase, es posible utilizar su nombre para declarar objetos de la misma.

Bibliografía

- Paul Meakin. *Fractals, scaling and growth far from equilibrium*, Cambridge University Press 1998
- Benoit B. Mandelbrot. *The fractal geometry of nature*, W. H. Freeman 1983
- Donald E. Knuth, *The Art of Computer Programming vol. 1: Fundamental Algorithms*, Addison Wesley 1979
- McCauley, Joseph L. *Chaos, Dynamics, and Fractals*, Cambridge Nonlinear Science Series 1994
- Donald E. Knuth, *Mathematics for the Analysis of Algorithms*, Stanford University, Stanford CA, 1990
- Donald E. Knuth, *The Art of Computer Programming vol. 3: Sorting and Searching*, Addison Wesley 1979
- William H. Press, Saul A. Teukolsky, William T. Vetering, Brian P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, 1988
- Meakin , P., *The diffusion-limited aggregation model and geological pattern formation In: Growth and dissolution in Geo-systems*. (eds. Jamtveit, B., and Meakin, P.) Kluwer Academic, 1999
- Herbert Schildt, *C++ para Programadores*, Mc Graw Hill 1996
- A. V. Hopcroft, J. E. Ullman, *Data Structures and Algorithms*, Addison Wesley, Reading mass

5. Análisis de los Datos Obtenidos

5.1. Modelo Bicolor DLA

5.1.1. Descripción del modelo

El proyecto de enumeración exacta parte inicialmente del modelo bicolor DLA, el cual consiste básicamente en colocar dos semillas de diferente color (*color1* y *color2*) a una distancia *d*, dentro de un enrejado. Posteriormente se tienen que generar partículas en lugares aleatorios dentro del enrejado, dichas partículas deben ser del color de alguna de las dos semillas, con una probabilidad complementaria ($pcolor1 + pcolor2 = 1.0$).

Una vez que ha sido creada una partícula, inicia una caminata aleatoria. Dentro de esta caminata puede suceder alguna de las diferentes situaciones:

- Llega el momento que la partícula en su caminata toca algún límite del enrejado, en ese momento dicha partícula muere y da inicio a la generación de una nueva partícula.
- Llega el momento en que partícula se encuentra adyacente a la semilla cuyo color es diferente, en este caso la partícula también muere y da origen a una nueva partícula en un lugar aleatorio dentro del enrejado
- El último caso es que la partícula en movimiento llegue a estar en un lugar adyacente a la semilla de su color, con lo cual se une a la semilla y empieza a formar una estructura

Este proceso se repita nuevamente y las partículas generadas solo se incorporaran cuando se encuentren adyacentes a las estructuras en formación de su mismo color y de igual forma se destruirán cuando se encuentren en lugares adyacentes a estructuras de diferente color o bien toquen el límite del enrejado.

Es conveniente mencionar que las distancia *d* de separación entre semillas es demasiado pequeña comparada con la distancia máxima de los límites del enrejado a cualquiera de las semillas.

Lo anteriormente expuesto en forma breve forma una idea respecto al funcionamiento del programa computacional, el cual se basa principalmente en generadores de números aleatorios de gran periodo y en comparadores de posición (debido a que se necesita saber en cada movimiento de la partícula su posición y esta compararla con el área o superficie de las estructuras fractales en formación, o limites del enrejado en cuestión). En el caso de dos dimensiones la comparación de posiciones se realiza con áreas de estructuras fractales, en el caso de 3 dimensiones estas comparaciones se tendrían que hacer contra volúmenes de superficies, y esto se complicarían conforme la dimensión se vaya incrementando, aun en el caso de que esto no proporcione un significado físico real.

El nuestro programa de enumeración exacta aplicado al modelo Bicolor DLA, utilizamos una dimensión de trabajo de 2. Una de las razones para trabajar en planos de 2 dimensiones es que los resultados que nos proporcionan la razón de simetría entre las estructuras fractales obtenidas y el cual es un número que proviene de una relación logarítmica es fácilmente representable en la determinación geométrica de las estructuras fractales obtenidas. Dicho de otra manera un simple número obtenido y que nos representa la asimetría de las estructuras nos lleva a imaginarnos rápidamente el comportamiento global de las estructuras fractales al finalizar del experimento.

Por ejemplo cuando al finalizar un experimento, se observa que una estructura alcanzó una cantidad de 5000 partículas, mientras que la otra solo alcanzo 4879 partículas, su razón de asimetría estaría dado por el $log(5000/4879) = 0.0106$. Este valor es muy cercano a cero y nos representa que ambas

estructuras resultan ser muy similares (forma y cantidad de agregados), una siguiente etapa a este trabajo de tesis es precisamente la graficación de estos resultados para confirmar lo que intuitivamente nos ha representado los resultados calculados anteriormente.

En el caso contrario en que una estructura alcanzo las 5000 partículas y la otra solamente fue integrada por 4 partículas, la razón de asimetría sería la siguiente $log(5000/4) = 3.0969$. En este otro caso este número nos representa que una estructura prácticamente absorbió a la otra.

En la figura *5.1.2.1* se observa una gráfica con resultados de asimetría relevantes en 100 experimentos.

5.1.2. Obtención de datos

La obtención de datos del Modelo de bicolor Agregación limitada se realizo mediante la ejecución de un programa computacional, y constituyó las siguientes etapas experimentales:

1ª Etapa

Realizar *100* ejecuciones de programa, cada ejecución a su vez proporcionó el resultado de *100* experimentos y cada uno de estos finalizó cuando alguna de las dos estructuras fractales en formación estuvo integrada con *5000* partículas.

Cada una de las ejecuciones del programa se realizó a diferentes distancias (*2, 4, 6, 8 y 10*) entre semillas y a diferentes probabilidades de que la partícula generada fuera del color 1 (*pcolor1* = diferentes valores entre *0.4* y *0.5*)

La siguiente tabla muestra el nombre del archivo obtenido en cada ejecución del programa, así como la relación entre las diferentes probabilidades y de las diferentes distancias entre semillas en cuestión.

	d = Distancia entre semillas				
Pcolor1	2	4	6	8	10
0.4000	exp_00	exp_20	exp_40	exp_60	exp_80
0.4100	exp_01	exp_21	exp_41	exp_61	exp_81
0.4200	exp_02	exp_22	exp_42	exp_62	exp_82
0.4300	exp_03	exp_23	exp_43	exp_63	exp_83
0.4400	exp_04	exp_24	exp_44	exp_64	exp_84
0.4500	exp_05	exp_25	exp_45	exp_65	exp_85
0.4550	exp_06	exp_26	exp_46	exp_66	exp_86
0.4600	exp_07	exp_27	exp_47	exp_67	exp_87
0.4625	exp_08	exp_28	exp_48	exp_68	exp_88
0.4650	exp_09	exp_29	exp_49	exp_69	exp_89
0.4675	exp_10	exp_30	exp_50	exp_70	exp_90
0.4700	exp_11	exp_31	exp_51	exp_71	exp_91
0.4725	exp_12	exp_32	exp_52	exp_72	exp_92
0.4750	exp_13	exp_33	exp_53	exp_73	exp_93
0.4775	exp_14	exp_34	exp_54	exp_74	exp_94
0.4800	exp_15	exp_35	exp_55	exp_75	exp_95
0.4850	exp_16	exp_36	exp_56	exp_76	exp_96
0.4900	exp_17	exp_37	exp_57	exp_77	exp_97
0.4950	exp_18	exp_38	exp_58	exp_78	exp_98
0.5000	exp_19	exp_39	exp_59	exp_79	exp_99

2ª Etapa

En cada archivo de datos, se obtuvo como ya se mencionó, el resultado de 100 experimentos. En cada experimento se obtuvo:

- El número total de partículas que se crearon
- El número total de partículas que se integraron a la estructura fractal 1
- El número total de partículas que se integraron a la estructura fractal 2
- La relación existente entre el logaritmo natural del número máximo de partículas integradas en una estructura fractal y el número de partículas integradas en la otra estructura fractal, lo cual es un indicador de simetría entre las estructuras fractales

$$\log\left(N_{\max}/N_{\min}\right)$$

Con los datos obtenidos en cada ejecución del programa (100 experimentos), se realizaron los siguientes cálculos:

- El número máximo correspondiente a la cantidad de partículas generadas en un experimento por ejecución del programa

Las coordenadas de las semillas son *(-1, 0)* y *(1, 0)* para el *color 1* y *2* respectivamente. Sí la probabilidad de que las partículas generadas sean de *color 1* es de *0.46* y solo nos interesa conocer la posición en que se agrega la primer partícula en *5000* experimentos, se pueden obtener los siguientes agregados iniciales:

Para la semilla 1, localizada en las coordenadas (-1, 0), las tres posiciones en las cuales se puede agregar la primer partícula caminante son las indicadas mediante el subíndice 1 en la figura 6.1.2

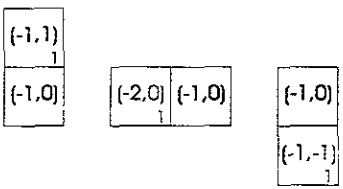


Figura 5 2.1.2

Para la semilla 2, localizada en las coordenadas (1, 0), las tres posiciones en las cuales se puede agregar la primer partícula caminante son las indicadas mediante el subíndice 1 en la figura 6.1.3

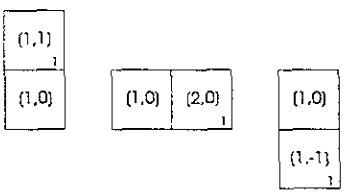


Figura 5 2 1 3

Para el experimento de este ejemplo, se requirieron los siguientes datos

DATOS	
Distancia entre semillas	2
Probabilidad de que las partículas generadas sean de color 1	0 46
Numero de experimentos en la prueba	5000
Numero máximo de partículas por fractal[XXVI]	2

[XXVI] Como las semillas de cada fractal son también partículas, entonces se le deben tomar en cuenta para establecer el número máximo de partículas por fractal

Para estos datos se obtuvieron los siguientes resultados:

RESULTADOS		
POSICIÓN	**CANTIDAD DE PARTICULAS AGREGADAS**	**% DEL TOTAL**
(-1, 1)	691	13.82
(-2, 0)	914	18.28
(-1, -1)	717	14.34
(1, -1)	820	16.4
(2, 0)	1066	21.32
(1, 1)	792	15.84
TOTAL	5000	100

Otros resultados interesantes que engloban toda la prueba (los *5000* experimentos), son:

	GENERADAS	MUERTAS	FRACTAL DE COLOR 1	FRACTAL DE COLOR 2
TOTAL DE PARTICULAS	30285	15285	7322	7678
PROMEDIO DE PARTICULAS	6.057	3.057	1.4644	1.5356

Una de las cualidades más importantes en este programa computacional es la generación masiva de resultados en tiempos de computadora muy pequeños, en el ejemplo anterior el tiempo para ejecutar esta prueba tomo aproximadamente 30 minutos en una computadora con procesador Celeron 466 Mhz y 256 Mb de RAM.

Datos del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0.3	(-1 , 1)	458	9 16	Generadas	29640	5 928
Núm de Exp	5000	(-2 , 0)	608	***12.16***	Integradas a Fractal 1	6521	1 3042
Dist Semillas	2	(-1 , -1)	455	9.1	Integradas a Fractal 2	8479	1 6958
P Fractal 1	1521	(1 , -1)	1005	20 1	Muertas	14640	2 928
P Fractal 2	3479	(2 , 0)	1306	***26.12***			
		(1 , 1)	1168	23 36			
		TOTAL	5000	100			

Datos del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0.35	(-1 , 1)	572	11 44	Generadas	30521	6 1042
Núm de Exp	5000	(-2 , 0)	723	***14.46***	Integradas a Fractal 1	6804	1 3608
Dist Semillas	2	(-1 , -1)	509	10 18	Integradas a Fractal 2	8196	1 6392
P Fractal 1	1804	(1 , -1)	933	18 66	Muertas	15521	3 1042
P Fractal 2	3196	(2 , 0)	1342	***26.84***			
		(1 , 1)	921	18 42			
		TOTAL	5000	100			

Datos del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 4	(-1 , 1)	619	12 38	Generadas	30012	6 0024
Núm de Exp	5000	(-2 , 0)	810	***16.2***	Integradas a Fractal 1	7012	1 4024
Dist Semillas	2	(-1 , -1)	583	11 66	Integradas a Fractal 2	7988	1 5976
P Fractal 1	2012	(1 , -1)	894	17 88	Muertas	15012	3 0024
P Fractal 2	2988	(2 , 0)	1187	***23.74***			
		(1 , 1)	907	18 14			
		TOTAL	5000	100			

Datos del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 41	(-1 , 1)	626	12 52	Generadas	29760	5 952
Núm de Exp	5000	(-2 , 0)	802	***16.04***	Integradas a Fractal 1	7000	1 4
Dist Semillas	2	(-1 , -1)	572	11 44	Integradas a Fractal 2	8000	1 6
P Fractal 1	2000	(1 , -1)	950	19	Muertas	14760	2 952
P Fractal 2	3000	(2 , 0)	1194	***23.88***			
		(1 , 1)	856	17 12			
		TOTAL	5000	100			

Datos del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 42	(-1 , 1)	632	12 64	Generadas	30273	6 0546
Num de Exp	5000	(-2 , 0)	880	***17.6***	Integradas a Fractal 1	7120	1 424
Dist Semillas	2	(-1 , -1)	608	12 16	Integradas a Fractal 2	7880	1 576
P Fractal 1	2120	(1 , -1)	884	17 68	Muertas	15273	3 0546
P Fractal 2	2880	(2 , 0)	1170	***23.4***			
		(1 , 1)	826	16 52			
		TOTAL	5000	100			

Datos del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 43	(-1 , 1)	638	12 76	Generadas	30071	6 0142
Num de Exp	5000	(-2 0)	865	***17.3***	Integradas a Fractal 1	7126	1 4252
Dist Semillas	2	(-1 , -1)	623	12 46	Integradas a Fractal 2	7874	1 5748
P Fractal 1	2126	(1 , -1)	867	17 34	Muertas	15071	3 0142
P Fractal 2	2874	(2 , 0)	1122	***22.44***			
		(1 1)	885	17 7			
		TOTAL	5000	100			

Datos del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 44	(-1 , 1)	614	12 28	Generadas	30371	6 0742
Num de Exp	5000	(-2 0)	932	***18.64***	Integradas a Fractal 1	7204	1 4408
Dist Semillas	2	(-1 , -1)	658	13 16	Integradas a Fractal 2	7796	1 5592
P Fractal 1	2204	(1 -1)	868	17 36	Muertas	15371	3 0742
P Fractal 2	2796	(2 , 0)	1107	***22 14***			
		(1 1)	821	16 42			
		TOTAL	5000	100			

Datos del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 45	(-1 , 1)	662	13 24	Generadas	30500	6 1
Num de Exp	5000	(-2 0)	958	***19 16***	Integradas a Fractal 1	7275	1 455
Dist Semillas	2	(-1 -1)	655	13 1	Integradas a Fractal 2	7725	1 545
P Fractal 1	2275	(1 -1)	808	16 16	Muertas	15500	3 1
P Fractal 2	2725	(2 , 0)	1073	***21.46***			
		(1 , 1)	844	16 88			
		TOTAL	5000	100			

Datos del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 46	(-1 , 1)	691	13 82	Generadas	30246	6 0 5 7

completó al llegar a *5000* partículas[XXIV], y la estructura de *color 2* solo fué integrada por su semilla. Con lo anterior podemos imaginar un fenómeno bastante raro en el cual la estructura que tiene menos probabilidad de formarse, se crea formando su estructura de *5000* partículas, mientras que la otra estructura fractal nunca aceptó ninguna partícula agregada, podemos decir que la estructura fractal de *color 1* anuló a la estructura fractal de *color 2*.

4ª Etapa

Graficación de varianzas del $\log_{10}$ de N_{max}/N_{min}, para una serie de *100* experimentos manteniendo constantes las probabilidades y distancias entre semillas.

En esta etapa se pretendió visualizar en una serie de *100* experimentos el comportamiento de las simetrías entre las estructuras fractales, cada experimento se ejecutó para una probabilidad y distancia entre semillas constantes.

De esto se obtiene la siguiente gráfica, que incorpora 5 funciones cada una de ellas a una distancia d igual a *2, 4, 6, 8 y 10* respectivamente.

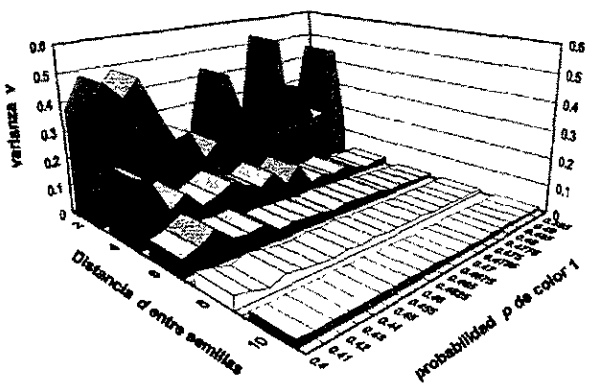


Figura 5.1.2.2

De la figura 5.1.2.2 de varianzas del $\log_{10}$ de N_{max}/N_{min}, para la serie de experimentos realizados, se obtuvo el calculo de las siguientes aproximaciones a polinomios:

Distancia d	Aproximación a la varianza experimental mediante un polinomio de 4° orden
2	$F(x) = -270635.6677\,x^4 + 486346\,6549\,x^3 - 327064.0449\,x^2 + 97548.29485\,x - 10886.60330$
4	$F(x) = 7991.212904\,x^4 - 14375.46571\,x^3 + 9692.54185\,x^2 - 2904\,926162\,x + 326.9090315$
6	$F(x) = 13954.53897\,x^4 - 25045.73658\,x^3 + 16808.95847\,x^2 - 4999.907907\,x + 556.2847488$

[XXIV] Recuérdese que la relación $\log_{10}$ de N_{max}/N_{min} nos ofrece un indicador de simetría, de tal forma que cuando esta se aproxima a cero se dice que las dos estructuras fractales son simétricas, o bien en el caso contrario, en el cual el valor de esta formula es muy grande, se dice que las estructuras fractales son asimétricas.

8	$F(x) = -8537.825334\,x^4 + 15441.31283\,x^3 - 10449.05008\,x^2 + 3135.046022\,x - 351.7987452$
10	$f(x) = 7334.851116\,x^4 - 13185.73354\,x^3 + 8866.987749\,x^2 - 2643.742284\,x + 294.9224881$

A continuación se muestra una gráfica obtenida a partir de los valores reales y la aproximación al polinomio de 4° orden[XXV] para encontrar la varianza del $log_{10}(Nmax/Nmin)$ a una distancia $d = 2$ (véase figura 5.1.2.3)

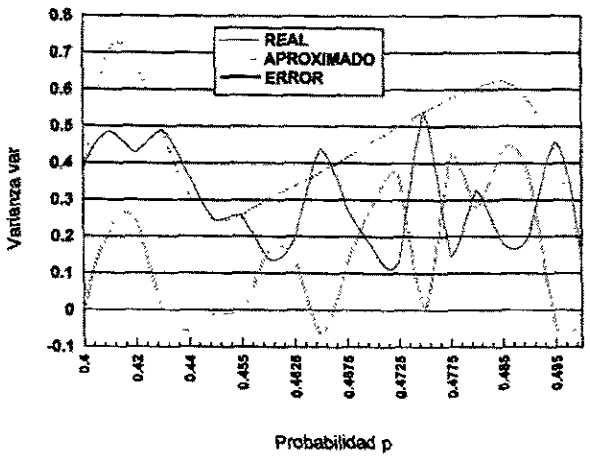


Figura 5.1.2.3

5.1.3. Equipo y material utilizado

El programa computacional es un desarrollo en C++ de linux (g++). Dicho programa fué ejecutado en dos computadoras personales, con las siguientes características cada una:

Procesador Pentium III a 450 Mhz

128 Mb Memoria RAM

Sistema Operativo Linux (Red Hat 6.1)

Cada ejecución de un programa se realizaba en un promedio de 8 horas en una computadora personal con las características mencionadas anteriormente y dedicando el 95% de los recursos del procesador para estas tareas. Recuérdese que se realizaron 100 ejecuciones de programas (800 horas)

Toda la información recopilada en las diferentes etapas experimentales se encuentra contenidas en tres volúmenes (aproximadamente 500 hojas)

[XXV] El polinomio al que se refiere en esta grafica es el indicado en la tabla anterior para una distancia $d = 2$

5.2. Enumeración exacta del modelo bicolor DLA

5.2.1. Descripción del Modelo

La enumeración exacta en el modelo tiene su origen en el análisis de los resultados obtenidos para grandes estructuras fractales del modelo Bicolor DLA, y cuya información relevante fué analizada en el capitulo anterior.

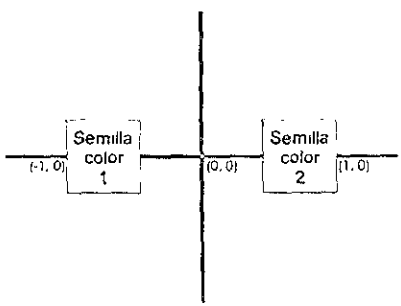
Los resultados de asimetría obtenidos a partir de muchos experimentos dejaron entrever aquellos casos en los cuales se rompía un esquema lógico, que partía de las condiciones de dichos experimentos. En el capitulo anterior como ya se observó en la figura *5.1.2.1*, casos en donde una estructura fractal invadía a otra cuando las probabilidades de que el color aleatorio de las partículas generadas son casi iguales fueron parte importante en la motivación para la realización del programa de enumeración exacta.

Este modelo pretende cuantificar con exactitud los orígenes de las estructuras fractales en el modelo Bicolor DLA. Esto significa que los resultados obtenidos a través de este modelo computacional nos deben generar datos del crecimiento inicial de las estructuras fractales, como son posiciones en las cuales se adhiere el primer agregado (para una semilla o la otra), cantidad de partículas que se mueren por haber rebasado el radio de muerte antes de que se adhiera la primer partícula, cantidad de partículas que mueren por intentar agregarse a una semilla de diferente color antes de que se haya adherido la primer partícula, determinar las posiciones de hasta las primeras 10 partículas que se integran a cualquier estructura fractal.

Todos estos resultados obtenidos para un crecimiento inicial de estructuras fractales en el modelo bicolor DLA, se obtienen para diferentes probabilidades de que las partículas generadas sean de un color u otro (como en el caso del capitulo anterior), para distancia entre semillas que pueden variar en *2, 4, 6, 8 y 10* unidades, para el número máximo de partículas que el usuario desee (se recomienda no mayor a *10* partículas).

Los resultados obtenidos se proporcionan a través de un archivos tipo texto, en el cual se indican las posiciones de agregación y el orden en que se van presentando conforme van creciendo las estructuras fractales.

Por ejemplo, consideremos que la distancia entre semillas es de *2* unidades como se muestra en la figura *6.1.1*



SEMILLAS CON UNA DISTANCIA d ENTRE ELLAS DE 2

Figura 5.2.1.1

Núm de Exp	5000	(-2 0)	914	***18.28***	Integradas a Fractal 1	7322	1 4644
Dist Semillas	2	(-1 , -1)	717	14 34	Integradas a Fractal 2	7678	1 5356
P Fractal 1	2322	(1 , -1)	820	16 4	Muertas	15285	3 057
P Fractal 2	2678	(2 , 0)	1066	***21.32***			
		(1 , 1)	792	15 84			
		TOTAL	**5000**	**100**			

Datos del Experimento		**Posiciones de agregación**	**Agregados**	**%**	**PARTICULAS**	**TOTALES**	**PROMEDIOS**
Prob Color 1	0 47	(-1 , 1)	687	13 74	Generadas	30126	6 0256
Núm de Exp	5000	(-2 , 0)	960	***19.2***	Integradas a Fractal 1	7377	1 4754
Dist Semillas	2	(-1 , -1)	730	14 6	Integradas a Fractal 2	7623	1 5246
P Fractal 1	2377	(1 , -1)	770	15 4	Muertas	15126	3 0256
P Fractal 2	2623	(2 , 0)	1084	***21.68***			
		(1 , 1)	769	15 38			
		TOTAL	**5000**	**100**			

Datos del Experimento		**Posiciones de agregación**	**Agregados**	**%**	**PARTICULAS**	**TOTALES**	**PROMEDIOS**
Prob Color 1	0 48	(-1 , 1)	673	13 46	Generadas	29613	5 9226
Núm de Exp	5000	(-2 , 0)	980	***19.6***	Integradas a Fractal 1	7412	1 4824
Dist Semillas	2	(-1 , -1)	759	15 18	Integradas a Fractal 2	7588	1 5176
P Fractal 1	2412	(1 , -1)	762	15 24	Muertas	14613	2 9226
P Fractal 2	2588	(2 , 0)	1060	***21.2***			
		(1 , 1)	766	15 32			
		TOTAL	**5000**	**100**			

Datos del Experimento		**Posiciones de agregación**	**Agregados**	**%**	**PARTICULAS**	**TOTALES**	**PROMEDIOS**
Prob Color 1	0 49	(-1 1)	765	15 3	Generadas	29354	5 8708
Núm. de Exp	5000	(-2 , 0)	982	***19.64***	Integradas a Fractal 1	7535	1 507
Dist Semillas	2	(-1 -1)	788	15 76	Integradas a Fractal 2	7465	1 493
P Fractal 1	2535	(1 -1)	747	14 94	Muertas	14354	2 8708
P Fractal 2	2465	(2 , 0)	1001	***20.02***			
		(1 , 1)	717	14 34			
		TOTAL	**5000**	**100**			

Datos del Experimento		**Posiciones de agregación**	**Agregados**	**%**	**PARTICULAS**	**TOTALES**	**PROMEDIOS**
Prob Color 1	0 5	(-1 , 1)	753	15 06	Generadas	30195	6 039
Num de Exp	5000	(-2 0)	1010	**20.2**	Integradas a Fractal 1	7530	1 506
Dist Semillas	2	(-1 , -1)	767	15 34	Integradas a Fractal 2	7470	1 494
P Fractal 1	2530	(1 -1)	720	14 4	Muertas	15195	3 039
P Fractal 2	2470	(2 , 0)	991	***19.82***			
		(1 1)	759	15 18			
		TOTAL	**5000**	**100**			

Tabla 5 2 2 1 Para una distancia entre semillas de 2 unidades

- *Para una distancia entre semillas de 4 unidades*

Datos del Experimento del Experimento		**Posiciones de agregación**	**Agregados**	**%**	**PARTICULAS**	**TOTALES**	**PROMEDIOS**
Prob Color 1	0 1	(-2 , 1)	136	2 72	Generadas	28295	5 659
Num de Exp	5000	(-3 , 0)	159	***3.18***	Integradas a Fractal 1	5475	1 095
Dist Semillas	4	(-2 , -1)	129	2 58	Integradas a Fractal 2	9525	1 905
P Fractal 1	475	(-1 , 0)	51	1 02	Muertas	13295	2 659
P Fractal 2	4525	(2 , -1)	1187	23 74			
		(3 0)	1529	***30 58***			
		(2 , 1)	1227	24 54			
		(1 0)	582	11 64			
		TOTAL	**5000**	**100**			

Datos del Experimento del Experimento		**Posiciones de agregación**	**Agregados**	**%**	**PARTICULAS**	**TOTALES**	**PROMEDIOS**
Prob Color 1	0 15	(-2 1)	195	3 9	Generadas	28048	5 6096
Num de Exp	5000	(-3 0)	286	***5 72***	Integradas a Fractal 1	5787	1 1574
Dist Semillas	4	(-2 , -1)	191	3 82	Integradas a Fractal 2	9213	1 8426
P Fractal 1	787	(-1 0)	115	2 3	Muertas	13048	2 6096
P Fractal 2	4213	(2 -1)	1145	22 9			
		(3 , 0)	1403	***28 06***			
		(2 , 1)	1134	22 68			
		(1 0)	531	10 62			
		TOTAL	**5000**	**100**			

Datos del Experimento del Experimento		**Posiciones de agregación**	**Agregados**	**%**	**PARTICULAS**	**TOTALES**	**PROMEDIOS**
Prob Color 1	0 2	(2 1)	277	5 54	Generadas	29162	5 8324

Núm de Exp	5000	(-3 , 0)	363	7.26	Integradas a Fractal 1	6075	1 215
Dist Semillas	4	(-2 , -1)	287	5 74	Integradas a Fractal 2	8925	1 785
P Fractal 1	1075	(-1 , 0)	148	2 96	Muertas	13163	2 6326
P Fractal 2	3925	(2 , -1)	1117	22 34			
		(3 , 0)	1253	25.06			
		(2 , 1)	1027	20 54			
		(1 , 0)	528	10 56			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 25	(-2 , 1)	331	6 62	Generadas	28140	5 628
Num de Exp	5000	(-3 , 0)	450	9	Integradas a Fractal 1	6292	1 2584
Dist Semillas	4	(-2 , -1)	347	6 94	Integradas a Fractal 2	8708	1 7416
P Fractal 1	1292	(-1 , 0)	164	3 28	Muertas	13140	2 628
P Fractal 2	3708	(2 , -1)	994	19 88			
		(3 , 0)	1234	24.68			
		(2 , 1)	1004	20 08			
		(1 , 0)	476	9 52			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 3	(-2 , 1)	419	8 38	Generadas	28172	5 6344
Núm de Exp	5000	(-3 , 0)	512	10.24	Integradas a Fractal 1	6534	1 3068
Dist Semillas	4	(-2 , -1)	424	8.48	Integradas a Fractal 2	8466	1 6932
P Fractal 1	1534	(-1 , 0)	179	3 58	Muertas	13172	2 6344
P Fractal 2	3466	(2 , -1)	926	18 52			
		(3 , 0)	1161	23.22			
		(2 , 1)	913	18 26			
		(1 , 0)	466	9 32			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 35	(-2 , 1)	447	8 94	Generadas	28600	5 72
Num de Exp	5000	(-3 , 0)	599	11.98	Integradas a Fractal 1	6785	1 357
Dist Semillas	4	(-2 , -1)	521	10 42	Integradas a Fractal 2	8215	1 643
P Fractal 1	1785	(-1 , 0)	218	4 36	Muertas	13600	2 72
P Fractal 2	3215	(2 , -1)	887	17 74			
		(3 , 0)	1082	21.64			
		(2 , 1)	864	17 28			
		(1 , 0)	382	7 64			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 4	(-2 , 1)	486	9 72	Generadas	28242	5 6484
Num de Exp	5000	(-3 , 0)	683	13.66	Integradas a Fractal 1	7001	1 4002
Dist Semillas	4	(-2 , -1)	573	11 46	Integradas a Fractal 2	7999	1 5998
P Fractal 1	2001	(-1 , 0)	259	5 18	Muertas	13242	2 6484
P Fractal 2	2999	(2 , -1)	804	16 08			
		(3 , 0)	1052	21 04			
		(2 , 1)	785	15 7			
		(1 , 0)	358	7 16			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 41	(-2 , 1)	539	10 78	Generadas	28833	5 7666
Núm de Exp	5000	(-3 , 0)	703	14 06	Integradas a Fractal 1	7078	1 4156
Dist Semillas	4	(-2 , -1)	578	11 56	Integradas a Fractal 2	7922	1 5844
P Fractal 1	2078	(-1 , 0)	258	5 16	Muertas	13833	2 7666
P Fractal 2	2922	(2 , -1)	778	15 56			
		(3 , 0)	1004	20 08			
		(2 , 1)	766	15 32			
		(1 , 0)	374	7 48			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 42	(-2 , 1)	518	10 36	Generadas	28815	5 763
Num de Exp	5000	(-3 , 0)	716	14.32	Integradas a Fractal 1	7042	1 4084
Dist Semillas	4	(-2 , -1)	534	10 68	Integradas a Fractal 2	7958	1 5916
P Fractal 1	2042	(-1 , 0)	274	5 48	Muertas	13815	2 763
P Fractal 2	2958	(2 , -1)	802	16 04			
		(3 , 0)	986	19 72			
		(2 , 1)	804	16 08			

		(1,0)	366	7 32
		TOTAL	5000	100

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 43	(-2,1)	629	12 58	Generadas	28141	5 6282
Num de Exp	5000	(-3,0)	748	***14.96***	Integradas a Fractal 1	7227	1 4454
Dist Semillas	4	(-2,-1)	597	11 94	Integradas a Fractal 2	7773	1 5546
P Fractal 1	2227	(-1,0)	253	5 06	Muertas	13141	2 6282
P Fractal 2	2773	(2,-1)	746	14 92			
		(3,0)	957	***19.14***			
		(2,1)	716	14 32			
		(1,0)	354	7 08			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 44	(-2,1)	592	11 84	Generadas	28340	5 668
Núm de Exp	5000	(-3,0)	752	***15.04***	Integradas a Fractal 1	7249	1 4498
Dist Semillas	4	(-2,-1)	609	12 18	Integradas a Fractal 2	7751	1 5502
P Fractal 1	2249	(-1,0)	296	5 92	Muertas	13340	2 668
P Fractal 2	2751	(2,-1)	710	14 2			
		(3,0)	976	***19.52***			
		(2,1)	743	14 86			
		(1,0)	322	6 44			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 45	(-2,1)	594	11 88	Generadas	28453	5 6906
Num de Exp	5000	(-3,0)	765	***15.3***	Integradas a Fractal 1	7271	1 4542
Dist Semillas	4	(-2,-1)	626	12 52	Integradas a Fractal 2	7729	1 5458
P Fractal 1	2271	(-1,0)	286	5 72	Muertas	13453	2 6906
P Fractal 2	2729	(2,-1)	748	14 96			
		(3,0)	908	***18.16***			
		(2,1)	730	14 6			
		(1,0)	343	6 86			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 46	(-2,1)	620	12 4	Generadas	27977	5 5954
Num de Exp	5000	(-3,0)	806	***16.12***	Integradas a Fractal 1	7368	1 4736
Dist Semillas	4	(-2,-1)	649	12 98	Integradas a Fractal 2	7632	1 5264
P Fractal 1	2368	(-1,0)	293	5 86	Muertas	12977	2 5954
P Fractal 2	2632	(2,-1)	724	14 48			
		(3,0)	876	***17.52***			
		(2,1)	699	13 98			
		(1,0)	333	6 66			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 47	(-2,1)	627	12 54	Generadas	28260	5 652
Num de Exp	5000	(-3,0)	824	***16.48***	Integradas a Fractal 1	7367	1 4734
Dist Semillas	4	(-2,-1)	582	11 64	Integradas a Fractal 2	7633	1 5266
P Fractal 1	2367	(-1,0)	334	6 68	Muertas	13260	2 652
P Fractal 2	2633	(2,-1)	673	13 46			
		(3,0)	910	***18 2***			
		(2,1)	705	14 1			
		(1,0)	345	6 9			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 48	(-2,1)	623	12 46	Generadas	28319	5 6638
Num de Exp	5000	(-3,0)	838	***16.76***	Integradas a Fractal 1	7411	1 4822
Dist Semillas	4	(-2,-1)	640	12 8	Integradas a Fractal 2	7589	1 5178
P Fractal 1	2411	(-1,0)	310	6 2	Muertas	13319	2 6638
P Fractal 2	2589	(2,-1)	711	14 22			
		(3,0)	917	***18 34***			
		(2,1)	640	12 8			
		(1,0)	321	6 42			
		TOTAL	5000	100			

Datos del Experimento del Experimento	Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS

Prob Color 1	0 49	(-2 ,1)	645	12 9
Num de Exp	5000	(-3 ,0)	874	***17.48***
Dist Semillas	4	(-2 ,-1)	665	13 3
P Fractal 1	2503	(-1 ,0)	319	6 38
P Fractal 2	2497	(2 ,-1)	667	13 34
		(3 ,0)	872	***17.44***
		(2 ,1)	647	12 94
		(1 ,0)	311	6 22
		TOTAL	**5000**	**100**

Generadas	28353	5 6706
Integradas a Fractal 1	7503	1 5006
Integradas a Fractal 2	7497	1 4994
Muertas	13353	2 6706

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob Color 1	0 5	(-2 ,1)	644	12 88
Num de Exp	5000	(-3 ,0)	875	***17.5***
Dist Semillas	4	(-2 ,-1)	725	14 5
P Fractal 1	2584	(-1 ,0)	340	6 8
P Fractal 2	2416	(2 ,-1)	644	12 88
		(3 ,0)	836	***16.72***
		(2 ,1)	622	12 44
		(1 ,0)	314	6 28
		TOTAL	**5000**	**100**

PARTICULAS	TOTALES	PROMEDIOS
Generadas	28564	5 7128
Integradas a Fractal 1	7584	1 5168
Integradas a Fractal 2	7416	1 4832
Muertas	13564	2 7128

Tabla 5.2.2 2 Para una distancia entre semillas de 4 unidades

- *Para una distancia entre semillas de 6 unidades*

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob Color 1	0 1	(-3 ,1)	110	2 2
Num de Exp	5000	(-4 ,0)	160	***3.2***
Dist Semillas	6	(-3 ,-1)	123	2 46
P Fractal 1	479	(-2 ,0)	86	1 72
P Fractal 2	4521	(3 ,-1)	1177	23 54
		(4 ,0)	1436	***28.72***
		(3 ,1)	1109	22 18
		(2 ,0)	799	15 98
		TOTAL	**5000**	**100**

PARTICULAS	TOTALES	PROMEDIOS
Generadas	29209	5 8418
Integradas a Fractal 1	5479	1 0958
Integradas a Fractal 2	9521	1 9042
Muertas	14209	2 8418

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob Color 1	0 15	(-3 ,1)	214	4 28
Num de Exp	5000	(-4 ,0)	250	***5***
Dist Semillas	6	(-3 ,-1)	210	4 2
P Fractal 1	815	(-2 ,0)	141	2 82
P Fractal 2	4185	(3 ,-1)	1108	22 16
		(4 ,0)	1332	***26.64***
		(3 ,1)	1050	21
		(2 ,0)	695	13 9
		TOTAL	**5000**	**100**

PARTICULAS	TOTALES	PROMEDIOS
Generadas	29224	5 8448
Integradas a Fractal 1	5815	1 163
Integradas a Fractal 2	9185	1 837
Muertas	14224	2 8448

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob Color 1	0 2	(-3 ,1)	271	5 42
Num de Exp	5000	(-4 ,0)	335	***6.7***
Dist Semillas	6	(-3 ,-1)	295	5 9
P Fractal 1	1102	(-2 ,0)	201	4 02
P Fractal 2	3898	(3 ,-1)	989	19 78
		(4 ,0)	1297	***25.94***
		(3 ,1)	962	19 24
		(2 ,0)	550	13
		TOTAL	**5000**	**100**

PARTICULAS	TOTALES	PROMEDIOS
Generadas	28639	5 7278
Integradas a Fractal 1	6102	1 2204
Integradas a Fractal 2	8898	1 7796
Muertas	13639	2 7278

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob Color 1	0 25	(-3 ,1)	324	6 48
Num de Exp	5000	(-4 ,0)	397	***7 94***
Dist Semillas	6	(-3 ,-1)	336	6 72
P Fractal 1	1267	(-2 ,0)	210	4 2
P Fractal 2	3733	(3 ,1)	979	19 58
		(4 ,0)	1190	***23 8***
		(3 ,1)	920	18 4
		(2 ,0)	644	12 88
		TOTAL	5000	100

PARTICULAS	TOTALES	PROMEDIOS
Generadas	29359	5 8718
Integradas a Fractal 1	6267	1 2534
Integradas a Fractal 2	8733	1 7466
Muertas	14359	2 8718

Datos del Experimento del Experimento	Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS

Prob_Color 1	0 3	(-3 ,1)	407	8 14	Generadas	28812	5 7624
Núm_de Exp	5000	(-4 ,0)	478	*9 56*	Integradas a Fractal 1	6506	1 3012
Dist_Semillas	6	(-3 ,-1)	379	7 58	Integradas a Fractal 2	8494	1 6988
P_Fractal 1	1506	(-2 ,0)	242	4 84	Muertas	13812	2 7624
P_Fractal 2	3494	(3 ,-1)	907	18 14			
		(4 ,0)	1097	*21.94*			
		(3 ,1)	925	18 5			
		(2 ,0)	565	11 3			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 35	(-3 ,1)	441	8 82	Generadas	28280	5 656
Núm_de Exp	5000	(-4 ,0)	612	*12.24*	Integradas a Fractal 1	6822	1 3644
Dist_Semillas	6	(-3 ,-1)	471	9 42	Integradas a Fractal 2	8178	1 6356
P_Fractal 1	1822	(-2 ,0)	298	5 96	Muertas	13280	2 656
P_Fractal 2	3178	(3 ,-1)	800	16			
		(4 ,0)	1027	*20.54*			
		(3 ,1)	806	16 12			
		(2 ,0)	545	10 9			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 4	(-3 ,1)	520	*10.4*	Generadas	29409	5 8818
Núm_de Exp	5000	(-4 ,0)	678	13 56	Integradas a Fractal 1	7353	1 4706
Dist_Semillas	6	(-3 ,-1)	509	10 18	Integradas a Fractal 2	7947	1 5894
P_Fractal 1	2053	(-2 ,0)	346	6 92	Muertas	14109	2 8218
P_Fractal 2	2947	(3 ,-1)	777	15 54			
		(4 ,0)	951	*19.02*			
		(3 ,1)	725	14 5			
		(2 ,0)	494	9 88			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 41	(-3 ,1)	504	10 08	Generadas	26976	5 7952
Num_de Exp	5000	(-4 ,0)	647	*12.94*	Integradas a Fractal 1	7074	1 4148
Dist_Semillas	6	(-3 ,-1)	558	11 16	Integradas a Fractal 2	7926	1 5852
P_Fractal 1	2074	(-2 ,0)	365	7 3	Muertas	13976	2 7952
P_Fractal 2	2926	(3 ,-1)	762	15 24			
		(4 ,0)	926	*18.52*			
		(3 ,1)	754	15 08			
		(2 ,0)	484	9 68			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 42	(-3 ,1)	561	11 22	Generadas	28784	5 7568
Num_de Exp	5000	(-4 ,0)	656	*13.12*	Integradas a Fractal 1	7124	1 4248
Dist_Semillas	6	(-3 ,-1)	534	10 68	Integradas a Fractal 2	7876	1 5752
P_Fractal 1	2124	(-2 ,0)	373	7 46	Muertas	13784	2 7568
P_Fractal 2	2876	(3 ,-1)	768	15 36			
		(4 ,0)	898	*17 96*			
		(3 ,1)	736	14 72			
		(2 ,0)	474	9 48			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 43	(-3 ,1)	568	11 36	Generadas	28864	5 7728
Num_de Exp	5000	(-4 ,0)	687	*13 74*	Integradas a Fractal 1	7145	1 429
Dist_Semillas	6	(-3 ,-1)	563	11 26	Integradas a Fractal 2	7855	1 571
P_Fractal 1	2145	(-2 ,0)	327	6 54	Muertas	13864	2 7728
P_Fractal 2	2855	(3 ,-1)	780	15 6			
		(4 ,0)	889	*17 78*			
		(3 ,1)	735	14 7			
		(2 ,0)	451	9 02			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 44	(-3 ,1)	569	11 38	Generadas	28960	5 7936
Num_de Exp	5000	(-4 ,0)	773	*15 46*	Integradas a Fractal 1	7338	1 4676
Dist_Semillas	6	(-3 ,-1)	583	11 66	Integradas a Fractal 2	7662	1 5324
P_Fractal 1	2338	(-2 ,0)	413	8 26	Muertas	13960	2 7936
P_Fractal 2	2662	(3 ,-1)	717	14 34			
		(4 ,0)	922	*18 44*			

		(3 ,1)	683	13 66
		(2 ,0)	440	8 8
		TOTAL	5000	100

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 45	(-3 ,1)	600	12	Generadas	28448	5 6896
Núm de Exp	5000	(-4 ,0)	673	***13.46***	Integradas a Fractal 1	7218	1 4436
Dist Semillas	6	(-3 ,-1)	583	11 66	Integradas a Fractal 2	7782	1 5564
P Fractal 1	2218	(-2 ,0)	362	7 24	Muertas	13448	2 6896
P Fractal 2	2782	(3 ,-1)	738	14 76			
		(4 ,0)	843	***16.86***			
		(3 ,1)	722	14 44			
		(2 ,0)	479	9 58			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 46	(-3 ,1)	621	12 42	Generadas	29359	5 8718
Num de Exp	5000	(-4 ,0)	757	***15.14***	Integradas a Fractal 1	7384	1 4768
Dist Semillas	6	(-3 ,-1)	622	12 44	Integradas a Fractal 2	7616	1 5232
P Fractal 1	2384	(-2 ,0)	384	7 68	Muertas	14359	2 8718
P Fractal 2	2616	(3 ,-1)	671	13 42			
		(4 ,0)	805	***16.1***			
		(3 ,1)	654	13 08			
		(2 ,0)	486	9 72			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 47	(-3 ,1)	580	11 6	Generadas	28313	5 6626
Num de Exp	5000	(-4 ,0)	714	***14.28***	Integradas a Fractal 1	7302	1 4604
Dist Semillas	6	(-3 ,-1)	606	12 12	Integradas a Fractal 2	7698	1 5396
P Fractal 1	2302	(-2 ,0)	402	8 04	Muertas	13313	2 6626
P Fractal 2	2698	(3 ,-1)	706	14 12			
		(4 ,0)	869	***17.38***			
		(3 ,1)	712	14 24			
		(2 ,0)	411	8 22			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 48	(-3 ,1)	625	12 5	Generadas	29016	5 8032
Num de Exp	5000	(-4 ,0)	791	***15.82***	Integradas a Fractal 1	7461	1 4922
Dist Semillas	6	(-3 ,-1)	630	12 6	Integradas a Fractal 2	7539	1 5078
P Fractal 1	2461	(-2 ,0)	415	8 3	Muertas	14016	2 8032
P Fractal 2	2539	(3 ,-1)	621	12 42			
		(4 ,0)	787	***15.74***			
		(3 ,1)	698	13 96			
		(2 ,0)	433	8 66			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 49	(-3 ,1)	616	12 32	Generadas	28812	5 7624
Num de Exp	5000	(-4 ,0)	805	***16 1***	Integradas a Fractal 1	7532	1 5064
Dist Semillas	6	(-3 ,-1)	698	13 96	Integradas a Fractal 2	7468	1 4936
P Fractal 1	2532	(-2 ,0)	413	8 26	Muertas	13812	2 7624
P Fractal 2	2468	(3 ,-1)	631	12 62			
		(4 ,0)	770	***15 4***			
		(3 ,1)	645	12 9			
		(2 ,0)	422	8 44			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 5	(-3 ,1)	635	12 7	Generadas	28684	5 7368
Num de Exp	5000	(-4 ,0)	850	***17***	Integradas a Fractal 1	7560	1 512
Dist Semillas	6	(-3 ,-1)	689	13 78	Integradas a Fractal 2	7440	1 488
P Fractal 1	2560	(-2 ,0)	386	7 72	Muertas	13684	2 7368
P Fractal 2	2440	(3 ,-1)	637	12 74			
		(4 ,0)	780	***15 6***			
		(3 ,1)	623	12 46			
		(2 ,0)	400	8			
		TOTAL	5000	100			

Tabla 5 2 2 3 Para una distancia entre semillas de 6 unidades

- *Para una distancia entre semillas de 8 unidades*

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 1	(-4 , 1)	111	2 22	Generadas	29979	5 9958
Núm de Exp	5000	(-5 , 0)	176	*3.52*	Integradas a Fractal 1	5486	1 097
Dist Semillas	8	(-4 , -1)	116	2 32	Integradas a Fractal 2	9515	1 903
P Fractal 1	485	(-3 , 0)	82	1 64	Muertas	14979	2 9958
P Fractal 2	4515	(4 , -1)	1179	23 46			
		(5 , 0)	1349	*26.98*			
		(4 , 1)	1129	22 58			
		(3 , 0)	864	17 28			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 15	(-4 , 1)	206	4 12	Generadas	29140	5 828
Núm de Exp	5000	(-5 , 0)	221	*4.42*	Integradas a Fractal 1	5810	1 162
Dist Semillas	8	(-4 , -1)	218	4 36	Integradas a Fractal 2	9190	1 838
P Fractal 1	810	(-3 , 0)	165	3 3	Muertas	14140	2 828
P Fractal 2	4190	(4 , -1)	1112	22 24			
		(5 , 0)	1263	*25.26*			
		(4 , 1)	992	19 84			
		(3 , 0)	823	16 46			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 2	(-4 , 1)	264	5 28	Generadas	29962	5 9924
Num de Exp	5000	(-5 , 0)	330	*6 6*	Integradas a Fractal 1	6049	1 2098
Dist Semillas	8	(-4 , -1)	244	4 88	Integradas a Fractal 2	8951	1 7902
P Fractal 1	1049	(-3 , 0)	211	4 22	Muertas	14962	2 9924
P Fractal 2	3951	(4 , -1)	992	19 84			
		(5 , 0)	1213	*24.26*			
		(4 , 1)	971	19 42			
		(3 , 0)	775	15 5			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 25	(-4 , 1)	322	6 44	Generadas	29722	5 9444
Num de Exp	5000	(-5 , 0)	411	*8 22*	Integradas a Fractal 1	6311	1 2622
Dist Semillas	8	(-4 , -1)	324	6 48	Integradas a Fractal 2	8689	1 7378
P Fractal 1	1311	(-3 , 0)	254	5 08	Muertas	14722	2 9444
P Fractal 2	3689	(4 , -1)	906	18 12			
		(5 , 0)	1126	*22.52*			
		(4 , 1)	932	18 64			
		(3 , 0)	725	14 5			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 3	(-4 , 1)	370	7 4	Generadas	29661	5 9322
Num de Exp	5000	(-5 , 0)	476	*9 52*	Integradas a Fractal 1	6555	1 311
Dist Semillas	8	(-4 , -1)	406	8 12	Integradas a Fractal 2	8445	1 689
P Fractal 1	1555	(-3 , 0)	303	6 06	Muertas	14661	2 9322
P Fractal 2	3445	(4 , -1)	901	18 02			
		(5 , 0)	1018	*20.36*			
		(4 , 1)	848	16 96			
		(3 , 0)	678	13 56			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 35	(-4 , 1)	461	9 22	Generadas	29739	5 9478
Num de Exp	5000	(-5 , 0)	516	*10 32*	Integradas a Fractal 1	6784	1 3568
Dist Semillas	8	(-4 , -1)	438	8 76	Integradas a Fractal 2	8216	1 6432
P Fractal 1	1784	(-3 , 0)	369	7 38	Muertas	14739	2 9478
P Fractal 2	3216	(4 , -1)	805	16 1			
		(5 , 0)	981	*19.62*			
		(4 , 1)	809	16 18			
		(3 , 0)	621	12 42			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 4	(-4 , 1)	583	*11 66*	Generadas	29457	5 8914

Num de Exp	5000	(-5 , 0)	551	11 02	Integradas a Fractal 1	7040	1 408
Dist Semillas	8	(-4 , -1)	498	9 96	Integradas a Fractal 2	7960	1 592
P Fractal 1	2040	(-3 , 0)	408	8 16	Muertas	14457	2 8914
P Fractal 2	2960	(4 , -1)	790	15 8			
		(5 , 0)	882	*17.64*			
		(4 , 1)	698	13 96			
		(3 , 0)	590	11 8			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 41	(-4 , 1)	570	11 4	Generadas	29328	5 8656
Núm de Exp	5000	(-5 , 0)	612	*12.24*	Integradas a Fractal 1	7119	1 4238
Dist Semillas	8	(-4 , -1)	539	10 78	Integradas a Fractal 2	7881	1 5762
P Fractal 1	2119	(-3 , 0)	398	7 96	Muertas	14328	2 8656
P Fractal 2	2881	(4 , -1)	730	14 6			
		(5 , 0)	851	*17.02*			
		(4 1)	725	14 5			
		(3 0)	575	11 5			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 42	(-4 , 1)	529	10 58	Generadas	29435	5 887
Núm de Exp	5000	(-5 , 0)	644	*12.88*	Integradas a Fractal 1	7161	1.4322
Dist Semillas	8	(-4 , -1)	555	11 1	Integradas a Fractal 2	7839	1 5678
P Fractal 1	2161	(-3 , 0)	433	8 66	Muertas	14435	2 887
P Fractal 2	2839	(4 , -1)	728	14 56			
		(5 , 0)	841	*16.82*			
		(4 1)	701	14 02			
		(3 , 0)	569	11 38			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 43	(-4 1)	559	11 18	Generadas	29207	5 8414
Núm de Exp	5000	(-5 , 0)	635	*12.7*	Integradas a Fractal 1	7159	1 4318
Dist Semillas	8	(-4 -1)	556	11 12	Integradas a Fractal 2	7841	1 5682
P Fractal 1	2159	(-3 , 0)	409	8 18	Muertas	14207	2 8414
P Fractal 2	2841	(4 , -1)	718	14 36			
		(5 , 0)	878	*17.56*			
		(4 1)	732	14 64			
		(3 , 0)	513	10 26			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 44	(-4 , 1)	548	10 96	Generadas	29619	5 9238
Num de Exp	5000	(-5 0)	710	*14.2*	Integradas a Fractal 1	7224	1 4448
Dist Semillas	8	(-4 -1)	536	10 72	Integradas a Fractal 2	7776	1 5552
P Fractal 1	2224	(-3 0)	430	8 6	Muertas	14619	2 9238
P Fractal 2	2776	(4 , -1)	702	14 04			
		(5 0)	828	*16.56*			
		(4 1)	688	13 76			
		(3 0)	558	11 16			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 45	(-4 1)	633	12 66	Generadas	29672	5 9344
Num de Exp	5000	(-5 , 0)	689	*13.78*	Integradas a Fractal 1	7322	1 4644
Dist Semillas	8	(-4 , -1)	585	11 7	Integradas a Fractal 2	7678	1 5356
P Fractal 1	2322	(-3 , 0)	415	8 3	Muertas	14672	2 9344
P Fractal 2	2678	(4 -1)	720	14 4			
		(5 , 0)	772	*15.44*			
		(4 1)	638	12 76			
		(3 0)	548	10 96			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 46	(-4 1)	592	11 84	Generadas	29878	5 9756
Num de Exp	5000	(-5 , 0)	684	*13 68*	Integradas a Fractal 1	7321	1 4642
Dist Semillas	8	(-4 , -1)	602	12 04	Integradas a Fractal 2	7679	1 5358
P Fractal 1	2321	(-3 0)	443	8 86	Muertas	14878	2 9756
P Fractal 2	2679	(4 -1)	706	14 12			
		(5 0)	797	*15 94*			
		(4 1)	657	13 14			

		(3 , 0)	519	10 38
		TOTAL	5000	100

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 47	(-4 , 1)	584	11 68	Generadas	29323	5 8646
Num_de Exp	5000	(-5 , 0)	752	***15.04***	Integradas a Fractal 1	7401	1 4802
Dist_Semillas	8	(-4 , -1)	613	12 26	Integradas a Fractal 2	7599	1 5198
P_Fractal 1	2401	(-3 , 0)	452	9 04	Muertas	14323	2 8646
P_Fractal 2	2599	(4 , -1)	678	13 56			
		(5 , 0)	794	***15.88***			
		(4 , 1)	605	12 1			
		(3 , 0)	522	10 44			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 48	(-4 , 1)	626	12 52	Generadas	29596	5 9192
Núm_de Exp	5000	(-5 , 0)	733	***14.66***	Integradas a Fractal 1	7386	1 4772
Dist_Semillas	8	(-4 , -1)	604	12 08	Integradas a Fractal 2	7614	1 5228
P_Fractal 1	2386	(-3 , 0)	423	8 46	Muertas	14596	2 9192
P_Fractal 2	2614	(4 , -1)	664	13 28			
		(5 , 0)	763	***15.26***			
		(4 , 1)	689	13 78			
		(3 , 0)	498	9 96			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 49	(-4 , 1)	634	12 68	Generadas	29523	5 9046
Num_de Exp	5000	(-5 , 0)	740	***14 8***	Integradas a Fractal 1	7511	1 5022
Dist_Semillas	8	(-4 , -1)	565	13 3	Integradas a Fractal 2	7489	1 4978
P_Fractal 1	2511	(-3 , 0)	472	9 44	Muertas	14523	2 9046
P_Fractal 2	2489	(4 , -1)	609	12 18			
		(5 , 0)	746	***14.92***			
		(4 , 1)	641	12 82			
		(3 , 0)	493	9 86			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 5	(-4 , 1)	589	11 78	Generadas	29560	5 912
Num_de Exp	5000	(-5 , 0)	771	***15 42***	Integradas a Fractal 1	7536	1 5072
Dist_Semillas	8	(-4 , -1)	694	13 88	Integradas a Fractal 2	7464	1 4928
P_Fractal 1	2536	(-3 , 0)	482	9 64	Muertas	14560	2 912
P_Fractal 2	2464	(4 , -1)	613	12 26			
		(5 , 0)	727	***14.54***			
		(4 , 1)	629	12 58			
		(3 , 0)	495	9 9			
		TOTAL	5000	100			

Tabla 5.2 2.4 Para una distancia entre semillas de 8 unidades

- *Para una distancia entre semillas de 10 unidades*

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 1	(-5 , 1)	131	2 62	Generadas	31029	6 2058
Num_de Exp	5000	(-5 , 0)	153	3.06	Integradas a Fractal 1	5537	1 1074
Dist_Semillas	10	(-5 , -1)	148	2 96	Integradas a Fractal 2	9463	1 8926
P_Fractal 1	537	(-4 , 0)	105	2 1	Muertas	16029	3 2058
P_Fractal 2	4463	(5 , -1)	1181	23 62			
		(6 , 0)	1277	***25 54***			
		(5 , 1)	1090	21 8			
		(4 , 0)	915	18 3			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob_Color 1	0 15	(-5 , 1)	185	3 7	Generadas	30937	6 1874
Num_de Exp	5000	(-6 , 0)	236	***4 72***	Integradas a Fractal 1	5784	1 1568
Dist_Semillas	10	(-5 , -1)	205	4 1	Integradas a Fractal 2	9216	1 8432
P_Fractal 1	784	(-4 , 0)	158	3 16	Muertas	15937	3 1874
P_Fractal 2	4216	(5 , -1)	1054	21 08			
		(6 , 0)	1258	***25 12***			
		(5 , 1)	1053	21 06			

		(4 , 0)	853	17 06
		TOTAL	5000	100

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 2	(-5 , 1)	270	5 4	Generadas	30306	6 0612
Núm de Exp	5000	(-6 , 0)	309	*6.18*	Integradas a Fractal 1	6044	1 2088
Dist Semillas	10	(-5 , -1)	263	5 26	Integradas a Fractal 2	8956	1 7912
P Fractal 1	1044	(-4 , 0)	202	4 04	Muertas	15306	3 0612
P Fractal 2	3956	(5 , -1)	971	19 42			
		(6 , 0)	1137	***22.74***			
		(5 , 1)	983	19 66			
		(4 , 0)	865	17.3			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 25	(-5 , 1)	308	6 16	Generadas	31086	6 2172
Núm de Exp	5000	(-6 , 0)	386	*7.72*	Integradas a Fractal 1	6296	1 2592
Dist Semillas	10	(-5 , -1)	348	6 96	Integradas a Fractal 2	8704	1 7408
P Fractal 1	1296	(-4 , 0)	254	5.08	Muertas	16086	3 2172
P Fractal 2	3704	(5 , -1)	933	18 66			
		(6 , 0)	1098	***21.96***			
		(5 , 1)	922	18 44			
		(4 , 0)	751	15 02			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 3	(-5 , 1)	378	7 56	Generadas	30722	6 1444
Num de Exp	5000	(-6 , 0)	468	*9.36*	Integradas a Fractal 1	6569	1 3138
Dist Semillas	10	(-5 , -1)	405	8 1	Integradas a Fractal 2	8431	1 6862
P Fractal 1	1569	(-4 , 0)	318	6 36	Muertas	15722	3 1444
P Fractal 2	3431	(5 , -1)	904	18 08			
		(6 , 0)	955	***19.1***			
		(5 , 1)	825	16 5			
		(4 , 0)	747	14 94			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 35	(-5 , 1)	469	9 38	Generadas	30373	6 0746
Num de Exp	5000	(-6 , 0)	498	*9.96*	Integradas a Fractal 1	6784	1 3568
Dist Semillas	10	(-5 , -1)	479	9 58	Integradas a Fractal 2	8216	1 6432
P Fractal 1	1784	(-4 , 0)	338	6 76	Muertas	15373	3 0746
P Fractal 2	3216	(5 , -1)	803	16 06			
		(6 , 0)	929	***18.58***			
		(5 , 1)	821	16 42			
		(4 , 0)	663	13 26			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 4	(-5 , 1)	488	9 76	Generadas	30935	6 187
Num de Exp	5000	(-6 , 0)	609	***12 18***	Integradas a Fractal 1	7019	1 4038
Dist Semillas	10	(-5 , -1)	528	10 56	Integradas a Fractal 2	7981	1 5962
P Fractal 1	2019	(-4 , 0)	394	7 88	Muertas	15935	3 187
P Fractal 2	2981	(5 , -1)	762	15 24			
		(6 , 0)	873	***17.46***			
		(5 , 1)	737	14 74			
		(4 , 0)	609	12 18			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 41	(-5 , 1)	515	10 3	Generadas	30085	6 017
Num de Exp	5000	(-6 , 0)	634	***12 68***	Integradas a Fractal 1	7135	1 427
Dist Semillas	10	(-5 , -1)	545	10 9	Integradas a Fractal 2	7865	1 573
P Fractal 1	2135	(-4 , 0)	441	8 82	Muertas	15085	3 017
P Fractal 2	2865	(5 , -1)	753	15 06			
		(6 , 0)	869	***17 38***			
		(5 , 1)	684	13 68			
		(4 , 0)	559	11 18			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS

Prob_Color 1	0 42	(-5 , 1)	503	10 06
Num_de Exp	5000	(-6 , 0)	619	***12.38***
Dist_Semillas	10	(-5 , -1)	503	10 06
P_Fractal 1	2050	(-4 , 0)	425	8 5
P_Fractal 2	2950	(5 , -1)	742	14 84
		(6 , 0)	865	***17.3***
		(5 , 1)	745	14 9
		(4 , 0)	598	11 96
		TOTAL	**5000**	**100**

Generadas	31152
Integradas a Fractal 1	7050
Integradas a Fractal 2	7950
Muertas	16152

	6 2304
	1 41
	1 59
	3 2304

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob_Color 1	0 43	(-5 , 1)	538	10 76
Núm_de Exp	5000	(-6 , 0)	624	***12.48***
Dist_Semillas	10	(-5 , -1)	568	11 36
P_Fractal 1	2176	(-4 , 0)	446	8 92
P_Fractal 2	2824	(5 , -1)	719	14 38
		(6 , 0)	802	***16.04***
		(5 , 1)	701	14 02
		(4 , 0)	602	12 04
		TOTAL	**5000**	**100**

PARTICULAS	TOTALES	PROMEDIOS
Generadas	30082	6 0164
Integradas a Fractal 1	7176	1 4352
Integradas a Fractal 2	7824	1 5648
Muertas	15082	3 0164

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob_Color 1	0 44	(-5 , 1)	547	10 94
Núm_de Exp	5000	(-6 , 0)	665	***13.3***
Dist_Semillas	10	(-5 , -1)	544	10 88
P_Fractal 1	2231	(-4 , 0)	475	9 5
P_Fractal 2	2769	(5 , -1)	725	14 5
		(6 , 0)	794	***15.88***
		(5 , 1)	671	13 42
		(4 , 0)	579	11 58
		TOTAL	**5000**	**100**

PARTICULAS	TOTALES	PROMEDIOS
Generadas	30304	6 0608
Integradas a Fractal 1	7231	1 4462
Integradas a Fractal 2	7769	1 5538
Muertas	15304	3 0608

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob_Color 1	0 45	(-5 , 1)	556	11 12
Num_de Exp	5000	(-6 , 0)	637	***12.74***
Dist_Semillas	10	(-5 , -1)	594	11 88
P_Fractal 1	2260	(-4 , 0)	473	9 46
P_Fractal 2	2740	(5 -1)	717	14 34
		(6 , 0)	778	***15.56***
		(5 , 1)	659	13 18
		(4 , 0)	586	11 72
		TOTAL	**5000**	**100**

PARTICULAS	TOTALES	PROMEDIOS
Generadas	30797	6 1594
Integradas a Fractal 1	7260	1 452
Integradas a Fractal 2	7740	1 548
Muertas	15797	3 1594

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob_Color 1	0 46	(-5 , 1)	609	12 18
Num_de Exp	5000	(-6 , 0)	690	***13.8***
Dist_Semillas	10	(-5 , -1)	617	12 34
P_Fractal 1	2418	(-4 0)	502	10 04
P_Fractal 2	2582	(5 -1)	666	13 32
		(6 , 0)	728	***14.56***
		(5 , 1)	660	13 2
		(4 , 0)	528	10 56
		TOTAL	**5000**	**100**

PARTICULAS	TOTALES	PROMEDIOS
Generadas	30841	6 1682
Integradas a Fractal 1	7418	1 4836
Integradas a Fractal 2	7582	1 5164
Muertas	15841	3 1682

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob_Color 1	0 47	(-5 , 1)	562	11 24
Num_de Exp	5000	(-6 , 0)	710	***14.2***
Dist_Semillas	10	(-5 , -1)	604	12 08
P_Fractal 1	2337	(-4 0)	461	9 22
P_Fractal 2	2663	(5 -1)	699	13 98
		(6 , 0)	780	*15.6*
		(5 , 1)	633	12 66
		(4 , 0)	551	11 02
		TOTAL	**5000**	**100**

PARTICULAS	TOTALES	PROMEDIOS
Generadas	30796	6 1592
Integradas a Fractal 1	7337	1 4674
Integradas a Fractal 2	7663	1 5326
Muertas	15796	3 1592

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%
Prob_Color 1	0 48	(-5 , 1)	600	12
Num_de Exp	5000	(-6 , 0)	726	***14.52***
Dist_Semillas	10	(-5 , -1)	620	12 4
P_Fractal 1	2408	(-4 0)	462	9 24
P_Fractal 2	2592	(5 -1)	667	13 34
		(6 , 0)	763	*15 26*

PARTICULAS	TOTALES	PROMEDIOS
Generadas	30366	6 0732
Integradas a Fractal 1	7408	1 4816
Integradas a Fractal 2	7592	1 5184
Muertas	15366	3 0732

		(5 , 1)	636	12 72
		(4 , 0)	526	10 52
		TOTAL	5000	100

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob. Color 1	0 49	(-5 , 1)	663	13 26	Generadas	30669	6 1338
Núm. de Exp	5000	(-6 , 0)	704	*14.08*	Integradas a Fractal 1	7482	1 4964
Dist. Semillas	10	(-5 , -1)	663	13.06	Integradas a Fractal 2	7518	1 5036
P. Fractal 1	2482	(-4 , 0)	462	9 24	Muertas	15669	3 1338
P. Fractal 2	2518	(5 , -1)	605	12.1			
		(6 , 0)	749	*14.98*			
		(5 , 1)	646	12 92			
		(4 , 0)	518	10.36			
		TOTAL	5000	100			

Datos del Experimento del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob. Color 1	0.5	(-5 , 1)	656	13.12	Generadas	30893	6.1786
Núm. de Exp	5000	(-6 , 0)	755	*15.1*	Integradas a Fractal 1	7592	1.5184
Dist. Semillas	10	(-5 , -1)	694	13.88	Integradas a Fractal 2	7408	1 4816
P. Fractal 1	2592	(-4 , 0)	487	9.74	Muertas	15893	3 1786
P. Fractal 2	2408	(5 , -1)	616	12 32			
		(6 , 0)	674	*13.48*			
		(5 , 1)	636	12.72			
		(4 , 0)	482	9 64			
		TOTAL	5000	100			

Tabla 5.2.2 5 Para una distancia entre semillas de 10 unidades

3ª Etapa

Como se vio en la etapa anterior, la generación masiva de resultados muchas veces no proporciona una visión rápida del comportamiento de dichos resultados. A continuación se presentan diversas gráficas que representan los resultados de la segunda etapa de obtención de datos.

- *P(d, p)*, es la función representada en la siguiente gráfica para cuando el primer agregado en la estructura fractal de la semilla de *color 1* se integra en la parte superior de esta (figura *6.2.1*).

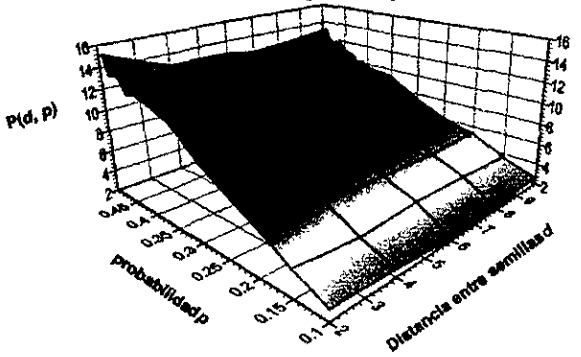


Figura 5 2 2.1 Primer agregado en el fractal de color 1 se integra por la parte superior

P(d, p), es una función que depende de *2* variables *d* y *p* (distancia entre semillas y probabilidad de que las partículas generadas sean de color 1 respectivamente). En la tabla 5.2.2 se muestra el resumen de resultados obtenidos a partir de que la probabilidad de que las partículas generadas sean de color 1 sea igual a 0.5 y que la distancia entre semillas sea de 2.

Datos del Experimento		Posiciones de agregación	Agregados	%	PARTICULAS	TOTALES	PROMEDIOS
Prob Color 1	0 5	(-1 , 1)	753	15 06	Generadas	30195	6 039
Num de Exp	5000	(-2 , 0)	1010	***20.2***	Integradas a Fractal 1	7530	1 506
Dist Semillas	2	(-1 , -1)	767	15 34	Integradas a Fractal 2	7470	1 494
P Fractal 1	2530	(1 , -1)	720	14 4	Muertas	15195	3 039
P Fractal 2	2470	(2 , 0)	991	***19.82***			
		(1 1)	759	15 18			
			5000	100			

La función *P(d, p)* evaluada en esta serie de experimentos, se representa como *P(2, 0.5)*, en donde los resultados obtenidos están representados con un valor de porcentaje obtenido a partir de la posición de agregación inicial.

Por ejemplo en la gráfica anterior la condición nos dice que dicha gráfica se obtuvo a partir de que el primer agregado en la estructura fractal de la semilla de color 1 se integró por la parte superior de esta, lo cual indicaría que para una distancia entre semillas de 2 en la estructura fractal de color 1, la posición de agregación debió ser *(-1, 1)*, observando el valor del porcentaje correspondiente a estas condiciones en la tabla se obtiene el valor de *15.06*, dicho valor en la figura *5.2 2.1* no es sino un punto de la función *P(d, p)*, localizado en la sección de color rojo.

En las siguiente gráficas la función *P(d, p)* proporciona esta misma visión de porcentajes, pretendiendo con esto analizar en todas las gráficas los valores obtenidos, así como los cambios en las pendientes y secciones en las cuales parecen tener comportamientos lineales. Es muy importante seguir experimentando sobre estos resultados a fin de ver el impacto que tienen estos cambios en las pendientes en la formación de un fractal de gran tamaño.

P(d, p), cuando el primer agregado en la estructura fractal de la semilla de *color 1* se integra por el lado izquierdo.

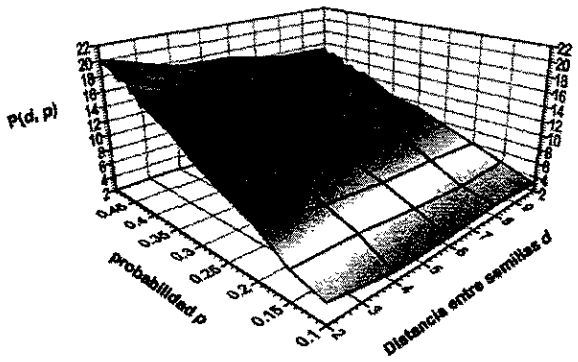


Figura 5.2.2.2 Primer agregado en el fractal de color 1 se integra por la parte izquierda

- *P(d, p)*, cuando el primer agregado en la estructura fractal de la semilla de *color 1* se integra por abajo.

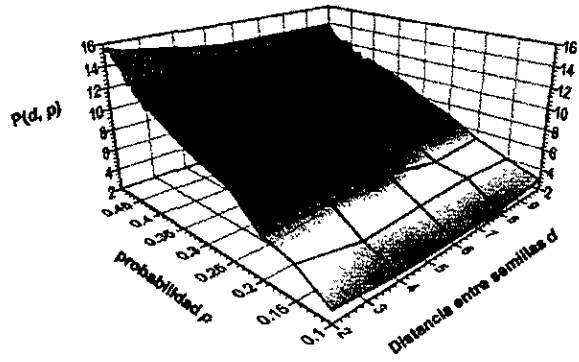


Figura 5.2 2.3 Primer agregado en el fractal de color 1 se integra por la parte inferior

- *P(d, p)*, cuando el primer agregado en la estructura fractal de la semilla de *color 1* se integra por el lado derecho.

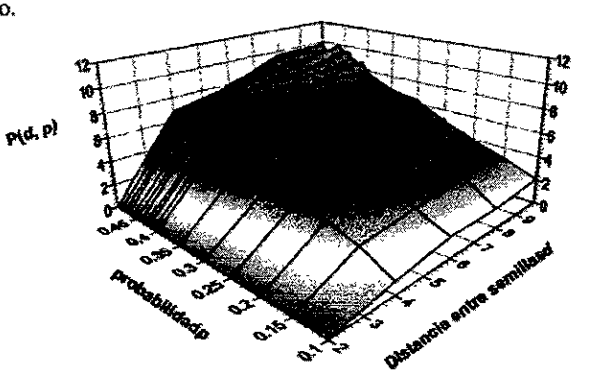


Figura 5.2.2.4 Primer agregado en el fractal de color 1 se integra por la parte derecha

- *P(d, p)*, cuando el primer agregado en la estructura fractal de la semilla de color 2 se integra por arriba.

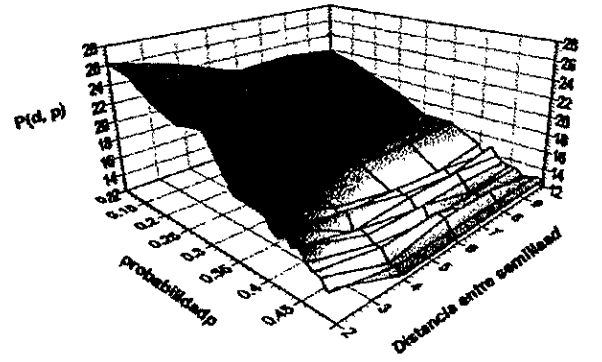


Figura 5.2.2.5 Primer agregado en el fractal de color 2 se integra por la parte superior

- *P(d, p)*, cuando el primer agregado en la estructura fractal de la semilla de color 2 se integra por el lado izquierdo.

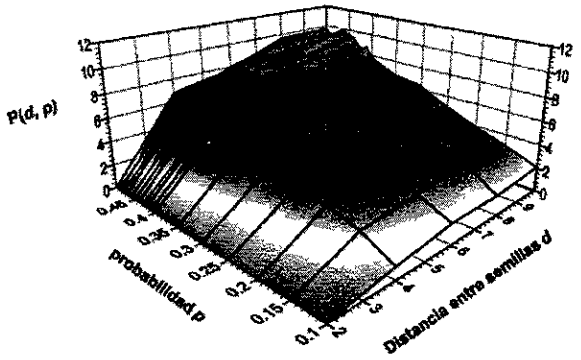


Figura 5 2 2.6 Primer agregado en el fractal de color 2 se integra por la parte izquierda

- *P(d, p)*, cuando el primer agregado en la estructura fractal de la semilla de color 2 se integra por abajo.

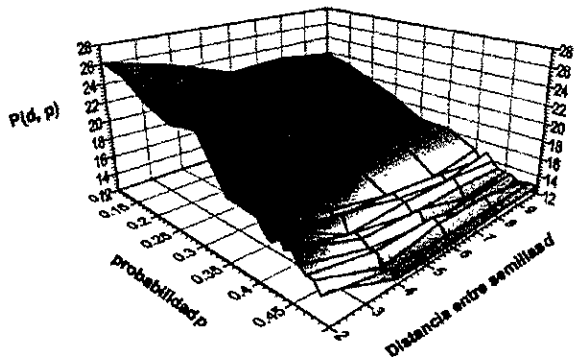


Figura 5 2.2.7 Primer agregado en el fractal de color 2 se integra por la parte inferior

- *P(d, p)*, cuando el primer agregado en la estructura fractal de la semilla de color 2 se integra por el lado derecho

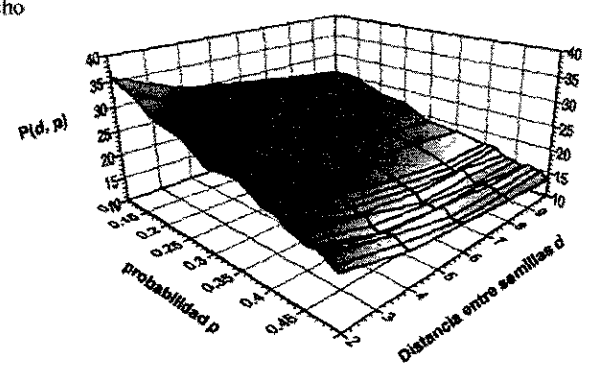


Figura 5.2.2.8 Primer agregado en el fractal de color 2 se integra por la parte derecha

5.2.3. Equipo y material utilizado

Este programa de Enumeración exacta también se desarrolló en C++. Dicho programa fué ejecutado en una computadoras personal, con las siguientes características cada una:

Procesador Pentium III a 450 Mhz

256 Mb Memoria RAM

Sistema Operativo Windows 2000

Cada ejecución de un programa se realizaba en un promedio de 30 minutos, dedicando el 95% de los recursos del procesador para estas tareas.

Los datos numéricos fueron importados desde los archivos *.txt a Excel 2000, para posteriormente ser analizados y graficados

Para la graficación se utilizo el software Harvard ChartXL, versión 2.0

Conclusiones Finales

El modelo Bicolor DLA al igual que el modelo original de Witten y Sander pretenden contribuir en la simulación de procesos naturales con una base estocástica. Desde hace algunos años se utilizan para simular procesos como el crecimiento del virus HIV, el crecimiento de colonias bacterianas, crecimientos cancerigenos, etc.

Muy particularmente el DLA en ciencias como la biología y la medicina exploran las geometrías fractales para entender y describir organismos biológicos, su desarrollo y crecimiento, así como su diseño estructural y propiedades funcionales. Extender estos conocimientos para evaluar cambios asociados con una enfermedad proporciona una esperanza en la comprensión de procesos patogénicos en la medicina.

Ahora bien, el desarrollar un programa que simule el modelo bicolor DLA, es otra perspectiva que de igual forma intenta contribuir en la comprensión de procesos aleatorios que bien pueden ser *aplicados a diversas ciencias.*

El programa de enumeración exacta surge de la necesidad de describir los orígenes de fractales que se comportan como en el modelo bicolor DLA, así como su impacto en la forma, tamaño y orientación final de las estructuras fractales de este modelo.

De las principales cualidades en el desarrollo de este programa se tienen:

- La utilización del algoritmo para la generación de números aleatorios de muy amplio periodo, sin correlaciones y rápido.
- Por otro lado la utilización de las tablas hash, las cuales proporcionan la ventaja de poderse aplicar a cualquier dimensión de trabajo y que el tiempo en el que el algoritmo verifica si la partícula esta o no unida a la estructura no depende de la cantidad de partículas que ya estén formando la estructura. Este tiempo se mantiene casi constante
- Una ventaja también muy importante es la utilización del algoritmo que aumentan terriblemente el desempeño del programa, alguno de ellos se refiere por ejemplo a la determinación del tamaño del paso dependiendo de la distancia a la que se encuentre la partícula caminante con respecto a las estructuras fractales. Esto es, cuando esta distancia es muy grande los pasos de desplazamiento de la partícula son grandes, lo cual origina que se acerque o aleje la mayor de las veces esta partícula se aleje o acerque a la estructura fractal más rápidamente, provocando con esto la muerte o bien la agregación de esta partícula a la estructura fractal.

Este trabajo ha intentado describir alguna de estas técnicas que hacen más eficiente el programa, otra parte de este trabajo fué el presentar y analizar brevemente algunos resultados interesantes como lo son las formas geométricas que se forman en las estructuras fractales con una distancia entre semillas $d=2$, así como con el valor de probabilidad cercano a $p=0.462564$ de que la partícula caminante sea de color 1.

Existieron otros resultados interesantes, sin embargo el más interesante siempre se obtuvo con las condiciones anteriores.

Los resultados experimentales de la ejecución del programa también arrojaros datos estadísticos, los cuales deben formar parte de la bitácora para un análisis de los siguientes resultados que se generen con nuevas condiciones, y con esto generar un análisis profundo que involucre todas las posibilidades de crecimiento de fractales aplicados al modelo bicolor DLA

Un siguiente trabajo que forma parte de la continuidad de estos desarrollos seria el implementar estos algoritmos para una dimensión de trabajo de 3, así como la graficación de los resultados, todo esto teniendo en cuenta no desviar mucho la atención de procesador, ya que lo primordial en este tipo de trabajos son los resultados.

Una gran ventaja en la actualidad es la creciente evolución de computadoras, lo cual nos permite realizar miles de operaciones y cálculos en tiempos muy pequeños con una cantidad mínima de recursos al alcance de cualquier persona.

Un reto ahora se dirige a desarrollar algoritmos complejos y eficientes que puedan ser utilizados en la programación de modelación de procesos.

Otro reto se dirige a otras áreas de investigación, alas cuales deben utilizar estas herramientas para la solución de sus objetivos particulares, y en caso de que no se acople a sus necesidades establecer un equipo con el área de desarrollo.

Por último, se debe tener en cuenta siempre la posibilidad de encontrar alternativas en el desarrollo de procesos aleatorios.

ANEXO 1

Rutina de Generación de Números Aleatorios ran3.

En este proyecto, la rutina de generación de números aleatorios se usa para la simulación de caminatas aleatorias, así como a la generación aleatoria del color que se asigna a cada partícula al momento de su formación. Para generar las estructuras fractales necesarias en esté propósito, las rutinas deben generar secuencias aleatorias de muy amplio periodo.

En este apartado se describirá la rutina de un generador de números aleatorios entre 0 y 1 utilizado en los experimentos de enumeración exacta del modelo DLA.

La rutina consta principalmente de dos procedimientos:

El primero se le conoce como inicialización del generador de números aleatorios, y el segundo como generador de números aleatorios (ran3).

1. Inicialización del generador de números aleatorios

El código básico de este procedimiento es el siguiente:

```
#include<time.h>
#include<stdlib.h>

void main ( ) {
 AjustaGenNumAleatorio();
 Datos( );
 Salida( );
}

void AjustaGenNumAleatorio( ){
 srand(time(0));
 random(time(0));
 IntTheta=-random(time(0))-1;
}
```

AjustaGenNumAleatorio, inicializa el valor de *IntTheta*.

La expresión *srand(time(0))*, se utiliza para generar una secuencia diferente de números aleatorios en cada ejecución (Aleatorización), debido a que la función time con argumento 0 devuelve en segundos la "hora calendario actual", este valor se convierte a un entero sin signo y se utiliza como la semilla para el generador de números aleatorios. Los prototipos de las funciones *srand* y *time* se encuentran en *<stdlib.h>* y *<time h>* respectivamente.

La expresión *random(time(0))*, es la encargada de devolver un valor del generador de números aleatorios, debido a que la función *srand* no devuelve un valor.

La última expresión *IntTheta=-random(time(0))-1*, asigna el valor aleatorio obtenido a IntTheta.

2. Generador de números aleatorios (ran3)

El código básico de este procedimiento es el siguiente:

```
#include<iostream>
#include<math>
#include<stdlib>

#define MBIG 1000000000
#define MSEED 161803398
#define MZ 0
#define FAC (1.0/MBIG)

int IntSeed=-3;
int IntColor=-3465;
int IntTheta=-13459;
int TemIntTheta=1;

double ran3(int &idum)
{
static int inext,inextp,
static long ma[56 ],
static int iff=0;
long mj,mk;
int i,ii,k,

if (idum < 0 || iff == 0){
   iff=1;
   mj =MSEED-(idum < 0 ? -idum : idum);
   mj %=MBIG; // Obtiene el Mod mj de MBIG
   ma[55]=mj;
   mk=1;
   for (i=1; 1<=54;i++)
   {
                ii=21*i) % 55,
                ma[ii]  mk,
```

```
            mk=mj-mk;
            if (mk < MZ) mk += MBIG;  //mk= MBIG+mk cuando mk < MZ
            mj=ma[ii];
    }
    for(k=1; k<=4, k++)
            for (i=1; i<=55;i++)
    {
              ma[i]-= ma[1+(i+30) %55];
              if (ma[i] < MZ) ma[i] += MBIG;
            }
    inext=0;
    inextp=31;
    idum=1;
  }
  if (++inext == 56) inext=1,
  if (++inextp == 56) inextp=1;
  mj=ma[inext]-ma[inextp];
  if (mj < Mz) mj += MBIG,
  ma[inext]=mj;
  return mj*FAC;
  }
```

En algún lugar del proyecto del programa se puede incluir esta instrucción *ran3(IntTheta)*, la cual invocaría el procedimiento *ran3*

(int &idum). En este caso *IntTheta* corresponde al valor aleatorio obtenido en el procedimiento 1 (inicialización del generador de números aleatorios), logrando con esto que cada vez que se ejecute el programa el valor de *IntTheta* será diferente.

Como *idum* es el valor de IntTheta, en el cual se puede observar que es negativo desde la asignación del procedimiento 1, por lo cual *mj=MSEED-(idum < 0? –idum : idum)* será la suma de *161803398 + idum*.

Básicamente el algoritmo que realiza la generación de números aleatorios *ran3*, consiste en formar arreglos de 56 elementos, numerados del 0 al 55, pero el trabajo significativo solo lo realiza con los subindices del 1 al 55.

El primer arreglo consiste en:

1° Asignar el valor de *mj* obtenido anteriormente al arreglo *ma[55]* e inicializar *mk* con 1

ma[55]=mj,

mk= 1,

```
for (i=1; i<=54; i++)
{
        ii=(21*I) %55;
          ma[ii]=mk;
        mk=mj-mk;
          if (mk <MZ) mk += MBIG;   // mk=MBIG+mk cuando mk < MZ
          mj=ma[ii];                // Corresponde a ma[34] con i=54 al finalizar
}
```

2° Posteriormente se inicia un ciclo con 54 iteraciones, las cuales van a ir formando el primer conjunto de arreglos, como se muestra en el código anterior

Los siguientes cuatro arreglos se obtienen con la ejecución del siguiente código

```
For (k=1; k<=4; k++)
        For (i=1; i<55; I++)
  {
          ms[i] -= ma[1+(i+30) % 55];
          if (ma[i] < MZ) ma[i] += MBIG;
  }
```

Cada arreglo formado en forma subsecuente, sustituye el arreglo anterior, de tal forma que cuando se ha realizado la última iteración con k=4, se obtiene el arreglo final *ma[i]*, con *i=1 a 55*.

El último paso consiste en la ejecución del siguiente pseudocódigo:

```
If (++inext == 56) inext=1,
If (++ inextp == 56) inextp=1,
Mj=ma[inext]-ma[inextp];
If (mj < Mz) mj += MBIG;
Ma[inext]=mj;
Return mj*FAC;
```

Estas últimas instrucciones proporcionan solo el *ma[1]*, el cual corresponde a *mj* y se multiplica por *FAC*. Es importante mencionar que esta multiplicación solo se realiza para que el número aleatorio generado se encuentre entre 0 y 1, con una precisión de hasta 10 decimales.

A continuación se anexan los archivos del proyecto random.bpr diseñado en Borland C++ Builder, así como un archivo de datos obtenido en una ejecución.

random.cpp

```
#pragma hdrstop
#include <condefs.h>
```

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio>
#include "rand3.h"
#include "varios.h"
#include "salida.h"
//---------------------------------------------------------------------------
USEUNIT("rand3.cpp");
USEUNIT("varios.cpp");
USEUNIT("salida.cpp");
//---------------------------------------------------------------------------
#pragma argsused

void main()
{

  AjustaGenNumAleatorio();  /* Inicializa los valores del generador de
                               números aleatorios */
  Datos();
  Salida();
  getch();
}
```

varios.cpp

```
#include <vcl.h>
#include <time>
#include <stdlib>
#pragma hdrstop
#include "varios.h"
#include "rand3.h"
#pragma package(smart_init)

void AjustaGenNumAleatorio() /* Ajusta el generador de números aleatorios */
```

```
{
 srand(time(0)); // Semilla del Generador de números aleatorios
 random(time(0)); //Genera un numero aleatorio entre 0 y time(0)
 IntSeed=-random(time(0))-1;
 double kk;
 for (int k=1; k<=50000L;k++)
   {double kk=2.0*k*2.0;}   //Retrazo intencional
 random(time(0));
 IntColor=-random(time(0))-1;
 for (int k=1; k<=50000L;k++)
   {double kk=2.0*k*2.0;}   //Retrazo intencional
 random(time(0));
 IntTheta=-random(time(0))-1;
 TemIntTheta = IntTheta;
}
```

salida.cpp

```
#include <conio>
#include <iostream>
#include <fstream>
#include "rand3.h"

void Datos ()
{
 clrscr();
 cout << "\n\n\tPROGRAMA DE GENERACION DE 3 NUMEROS ALEATORIOS" << endl;
 getch();
 cout << "\n\n\t\tValores de Inicialization\n" << endl;
 int Int_Seed=IntSeed;
 cout <<"\tAleatorio inicial de IntSeed  = " << Int_Seed << endl,
 int Int_Theta=IntTheta,
 cout <<"\tAleatorio inicial de IntTheta = " << Int_Theta << endl,
 int Int_Color=IntColor,
 cout <<"\tAleatorio inicial de IntColor = " << Int_Color << endl,
```

```
getch();
cout << "\n\t\tValores Aleatorios con periodos\n\t\tAleatorios de inicializacion\n" << endl;
double RandomSeed = ran3(IntSeed);
cout<<"\tPara IntSeed  =\t" <<RandomSeed<<endl;
double RandomTheta = ran3(IntTheta);
cout<<"\tPara IntTheta =\t" <<RandomTheta<<endl;
double RandomColor = ran3(IntColor);
cout<<"\tPara IntColor =\t" <<RandomColor<<endl;
getch();
cout<<"\n\n\t\t\tOBSERVE LOS DATOS OBTENIDOS EN EL ARCHIVO
      random.dat"<<endl;
ofstream Fout("random.dat");
  if (!Fout)
  {
  cout << "ERROR - al abrir archivo de datos para escritura" << endl;
  exit(1);
  }
Fout << "\n\nValores de Inicializacion\n" << endl;
Fout <<"Aleatorio inicial de IntSeed  = " << Int_Seed << endl;
Fout <<"Aleatorio inicial de IntTheta = " << Int_Theta << endl;
Fout <<"Aleatorio inicial de IntColor = " << Int_Color << endl;
Fout << "\n\nValores Aleatorios con periodos Aleatorios de inicializacion\n" << endl;
Fout<<"IntSeed  = " <<RandomSeed<<endl;
Fout<<"IntTheta = " <<RandomTheta<<endl;
Fout<<"IntColor = " <<RandomColor<<endl;
Fout.close();
}

void Salida ()
{
 getch(),
 clrscr(),
 cout << endl << "\n\n\n\tPROGRAMA DE GENERACION DE NUMEROS ALEATORIOS"
 <<"\n\tPARA PRUEBA DE TESIS DE MAESTRIA"<<
```

```
  "\n\n\n\tElaborado por:" <<  "\n\t\t\t\tIng. Sergio Ramírez García" <<
  "\n\n\tCon la colaboración de:" << "\n\n\t\t\t\tDra. Suemi Rodríguez Romo"
  << "\n\t\t\t\tDr. Vladimir E. Tchijov "<<endl ;
}
```

rand3.cpp

```
#include <vcl.h>
#include <iostream>
#include <math>
#include <stdlib>

#define MBIG 1000000000
#define MSEED 161803398
#define MZ 0
#define FAC (1.0/MBIG)
#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28441
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349

int IntSeed=-3;
int IntColor=-3465;
int IntTheta=-13459,
int TemIntTheta=1;

double ran3(int &idum)
{
```

```
static int inext,inextp;
static long ma[56];
static int iff=0;
long mj,mk;
int i,ii,k;

if (idum < 0 || iff == 0){
  iff=1;
  mj=MSEED-(idum < 0 ? -idum : idum); /* Obtiene el valor de 161803398 +
                          idum cuando idum < 0 (Siempre)*/
  mj %= MBIG;  // Obtiene el Mod mj de MBIG
  ma[55]=mj;
  mk=1;
  for (i=1; i<=54; i++)
  {
          ii=(21*i) % 55;
          ma[ii]=mk;
          mk=mj-mk,
          if (mk < MZ) mk += MBIG;   // mk= MBIG+mk cuando mk < MZ
          mj=ma[ii];                 // Corresponde a ma[34] con i=54 al finalizar
  }
  for (k=1; k<=4; k++)
          for (i=1; i<=55; i++)
    {
            ma[i] -= ma[1+(i+30) % 55];
            if (ma[i] < MZ) ma[i] += MBIG;
          }
  inext=0;
  inextp=31;
  idum=1;
}
if (++inext == 56) inext=1,
if (++inextp == 56) inextp=1;
mj=ma[inext]-ma[inextp],
```

```
  if (mj < MZ) mj += MBIG;
  ma[inext]=mj;
  return mj*FAC;
 }

 void nrerror(char *error_text)
 {
  cout << error_text << endl;
  exit(1);
 }

 int RandInt3(int &idum, int IMax)
 {
  return (int)floor(ran3(idum)*IMax);
 }
```

varios.h

```
//---------------------------------------------------------------------------
#ifndef __VARIOS__
#define __VARIOS__
//---------------------------------------------------------------------------
void AjustaGenNumAleatorio();
#endif
```

salida.h

```
//---------------------------------------------------------------------------
#ifndef _SALIDA_
#define _SALIDA_
//---------------------------------------------------------------------------
void Datos();
void Salida();
void Archivo();
#endif
```

rand3.h

```
//---------------------------------------------------------------------------
#ifndef _RANDFUNC_
#define _RANDFUNC_
//---------------------------------------------------------------------------
extern int IntSeed;
extern int IntColor;
extern int IntTheta;
extern int TemIntTheta;

double ran3(int &);
int RandInt3(int &, int);
#endif
```

Resultados Obtenidos en una ejecución de random

Valores de inicialización	
Aleatorio inicial de *IntSeed*	-567949429
Aleatorio inicial de *IntTheta*	-75140693
Aleatorio inicial de *IntColor*	-23662312
Valores Aleatorios con Periodos Aleatorios de Inicialización	
IntSeed	0.805726
IntTheta	0.336308
IntColor	0.448291

Para el valor de inicialización de la variable *IntSeed*, que se muestra en la tabla anterior y los siguientes datos.

MBIG	1000000000
MSEED	161803398
MZ	0
FAC	0.000000001
Inicializador de *IntSeed*	**-567949429**
Resultado Esperado ¿??	**0.805726492**

A continuación se muestran los arreglos formados mediante el algoritmo ran3 para los datos de este ejemplo.

ARREGLO OBTENIDO EN EL PRIMER CICLO *ma[i]*							
	ma[ ]		ma[ ]		ma[ ]		ma[ ]
0	299508917	14	900021165	28	863931814	42	161803398
1	400016711	15	405572259	29	63654101	43	300028968
2	352784963	16	809016987	30	500070193	44	652782630
3	103444166	17	200181611	31	605562927	45	501315511
4	100474640	18	163906151	32	400502367	46	101242309
5	886155526	19	700191590	33	203252287	47	494560427
6	700072403	20	508514552	34	597525755	48	400025619
7	322291369	21	1	35	500004454	49	458359555
8	838196603	22	99987743	36	52787296	50	514589808
9	799958775	23	700002333	37	705572821	51	299888582
10	47219703	24	602128655	38	99706971	52	441656776
11	100813144	25	999232331	39	277750625	53	700310777
12	897990022	26	391595099	40	119187	54	694737735
13	897034672	27	300046784	41	186223183	55	-838196601

VALORES DE CALCULOS REALIZADOS PARA LA OBTENCION DEL PRIMER ARREGLO *ma[ ]*							
i	ii	ma[ii]	mk	mj	mk	mk+	mj
1	21	1	1	-406146031	-406146032	593853968	1
2	42	593853968	593853968	1	-593853967	406146033	593853968
3	8	406146033	406146033	593853968	187707935	187707935	406146033
4	29	187707935	187707935	406146033	218438098	218438098	187707935
5	50	218438098	218438098	187707935	-30730163	969269837	218438098
6	16	969269837	969269837	218438098	-750831739	249168261	969269837
7	37	249168261	249168261	969269837	720101576	720101576	249168261
8	3	720101576	720101576	249168261	-470933315	529066685	720101576
9	24	529066685	529066685	720101576	191034891	191034891	529066685
10	45	191034891	191034891	529066685	338031794	338031794	191034891
11	11	338031794	338031794	191034891	-146996903	853003097	338031794
12	32	853003097	853003097	338031794	-514971303	485028697	853003097
13	53	485028697	485028697	853003097	367974400	367974400	485028697
14	19	367974400	367974400	485028697	117054297	117054297	367974400
15	40	117054297	117054297	367974400	250920103	250920103	117054297
16	6	250920103	250920103	117054297	-133865806	866134194	250920103
17	27	866134194	866134194	250920103	-615214091	384785909	866134194

18	48	384785909	384785909	866134194	481348285	481348285	384785909
19	14	481348285	481348285	384785909	-96562376	903437624	481348285
20	35	903437624	903437624	481348285	-422089339	577910661	903437624
21	1	577910661	577910661	903437624	325526963	325526963	577910661
22	22	325526963	325526963	577910661	252383698	252383698	325526963
23	43	252383698	252383698	325526963	73143265	73143265	252383698
24	9	73143265	73143265	252383698	179240433	179240433	73143265
25	30	179240433	179240433	73143265	-106097168	893902832	179240433
26	51	893902832	893902832	179240433	-714662399	285337601	893902832
27	17	285337601	285337601	893902832	608565231	608565231	285337601
28	38	608565231	608565231	285337601	-323227630	676772370	608565231
29	4	676772370	676772370	608565231	-68207139	931792861	676772370
30	25	931792861	931792861	676772370	-255020491	744979509	931792861
31	46	744979509	744979509	931792861	186813352	186813352	744979509
32	12	186813352	186813352	744979509	558166157	558166157	186813352
33	33	558166157	558166157	186813352	-371352805	628647195	558166157
34	54	628647195	628647195	558166157	-70481038	929518962	628647195
35	20	929518962	929518962	628647195	-300871767	699128233	929518962
36	41	699128233	699128233	929518962	230390729	230390729	699128233
37	7	230390729	230390729	699128233	468737504	468737504	230390729
38	28	468737504	468737504	230390729	-238346775	761653225	468737504
39	49	761653225	761653225	468737504	-292915721	707084279	761653225
40	15	707084279	707084279	761653225	54568946	54568946	707084279
41	36	54568946	54568946	707084279	652515333	652515333	54568946
42	2	652515333	652515333	54568946	-597946387	402053613	652515333
43	23	402053613	402053613	652515333	250461720	250461720	402053613
44	44	250461720	250461720	402053613	151591893	151591893	250461720
45	10	151591893	151591893	250461720	98869827	98869827	151591893
46	31	98869827	98869827	151591893	52722066	52722066	98869827
47	52	52722066	52722066	98869827	46147761	46147761	52722066
48	18	46147761	46147761	52722066	6574305	6574305	46147761
49	39	6574305	6574305	46147761	39573456	39573456	6574305
50	5	39573456	39573456	6574305	-32999151	967000849	39573456
51	26	967000849	967000849	39573456	-927427393	72572607	967000849
52	47	72572607	72572607	967000849	894428242	894428242	72572607
53	13	894428242	894428242	72572607	-821855635	178144365	894428242
54	34	178144365	178144365	894428242	716283877	716283877	178144365
	0	716283877	716283877	178144365	-538139512	461860488	716283877

i	m(i)	k=1		k=1			k=2			k=3			k=4	mj	
0	718283877														
1	577910661	-275092438	-275092438	724907564	271476185	271476185	271476185	173147019	173147019	173147019	-556925972	-556925972	443074028	-194273508	**805726492**
2	652515333	94349176	94349176	94349176	-507728013	-507728013	492271987	-374073301	-374073301	825926699	-127512714	-127512714	872487286	229740388	229740388
3	720101578	541957211	541957211	541957211	-183723494	-183723494	618276506	744482920	744482920	744482920	-21021589	-21021589	978978411	-6026681	991973319
4	676772370	-226685254	-226685254	773334746	614074948	614074948	614074948	-110368456	-110368456	889033544	20377710	20377710	20377710	-462957596	537042404
5	39573456	-14995490	-14995490	985004510	884885218	884885218	884885218	576308164	576308164	576308164	-179764111	-179764111	620235889	135893952	135893952
6	250920103	1751842	1751842	1751842	-603449897	-603449897	396550103	220469296	220469296	220469296	275445888	-275445888	724554112	563914151	563914151
7	230390729	-378174502	-378174502	621825498	303573681	303573681	303573681	31878262	31878262	31878262	-847059028	-847059028	152940972	-673069409	326930591
8	406146033	399571728	399571728	390571728	355102193	355102193	355102193	-631743815	-631743815	368258185	92725455	92725455	92725455	-204112317	795887683
9	73143265	-43911032	-43911032	956088955	735731901	735731901	735731901	112905875	112905875	112905875	110692515	110692515	110692515	-595798856	404201144
10	151591893	-547538340	-547538340	452483650	-348112881	-348112881	653887119	306289077	306289077	306289077	-221500583	221500583	778499417	314412126	314412126
11	338031794	-255822174	-255822174	744177825	434818394	434818394	434818394	-144814430	-144814430	855185570	385920528	385920528	385920528	169926525	169926525
12	185813352	-65570348	-65570348	934429654	831582258	831582258	831582258	532484739	532484739	532484739	71730358	71730358	71730358	761834637	238365363

26	25	24	23	22	21	20	19	18	17	16	15	14	13
957000849	931792861	529086885	402053613	325528963	1	928518962	367874400	46147761	285337601	969269837	707084279	481348285	894428242
872651873	1204885297	935212716	-228593582	159501734	-52722065	35816130	149538302	-715505464	-99446308	896697230	-37895230	290313394	643968522
872651873	206885297	935212716	-228593582	-159501734	-52722065	35816130	149538302	-715505464	-99446308	896697230	-37895230	290313394	643968522
872651873	206885297	935212718	773408418	840498265	947277935	35818130	149536302	284494538	800551692	896697230	962104770	290313394	643968522
1383379659	-64510898	818403076	322247814	558172994	589958827	-534109719	-198250123	-270273392	450978499	597531041	57623527	46556438	429120932
380379668	-64590898	818403076	322247814	558172994	589958827	534109719	-198250123	-270273392	450978499	597531041	57623527	46556438	429120932
1383379665	935409102	818403076	322247814	558172994	589958827	465390281	803749877	729726608	450978499	597531041	57623527	46556438	429120932
754452987	782282083	-124935898	-347972302	-388314071	313967129	404064909	-181658862	110367782	-180191618	-379387334	-288884722	-807241891	-319834377
754452987	782282083	-124935898	-347972302	-388314071	313967129	404064909	-181658862	110367782	-180191618	-379387334	-288884722	-807241891	-319834377
754452987	702262083	875064104	652027698	612685929	313967129	404064909	638343138	110367782	819808382	620012086	711315278	392758109	680165623
851965701	1319188055	73317538	542276774	430728887	229689897	105657094	627389386	-746728961	63702369	295721989	-21307042	52926909	335275223
851965701	319188055	73317538	542276774	430728887	229689897	105657094	627389388	-746728961	63702369	295721989	-21307042	52926909	335275223
851965701	319188055	73317538	542276774	430728887	229689897	105657094	627389388	253271039	63702369	295721989	976692958	52926909	335275223
9478415	-123885973	-739777594	268440930	189065979	-883310779	-408311575	298401335	284637649	-819088106	-488891914	676799525	-57234394	96041917
9478415	876114027	260222406	268440930	189065979	316689221	591688425	298401335	715362351	380913894	513108086	676799525	942765606	96041917

27	686134194	324178983	324178983	324178983	507900477	507900477	507900477	-236582443	-236582443	763417557	784439146	784439146	784439146	-194539285	805460735
28	488737504	695402758	695402758	695402758	81327810	81327810	81327810	191694266	191694266	191694266	171316556	171316556	171316556	150936848	150936848
29	187707935	202703425	202703425	202703425	-662161793	-662161793	337838207	-238469977	-238469977	761530023	941294134	941294134	941294134	121058245	121058245
30	179240433	177488591	177488591	177488591	760938488	760938488	760938466	560469192	560469192	560469192	835915080	835915080	835915080	111360968	111360968
31	98869827	477044329	477044329	477044329	173470668	173470668	173470668	141592406	141592406	141592406	988651434	988651434	988651434	835710462	835710462
32	653003097	453431369	453431369	453431369	98329176	98329176	98329176	730072991	730072991	730072991	637347536	637347536	637347536	544822081	544822081
33	556186157	602077169	602077169	602077169	-133654712	-133654712	866345288	753439413	753439413	753439413	642746898	642746898	642746898	532054363	532054363
34	178144365	725680705	725680705	725680705	1071793586	71793586	71793586	-234495491	-234495491	765504509	987005092	987005092	987005092	208505875	208505875
35	903437624	1159259798	159259798	159259798	-275558596	-275558596	724441404	869255834	869255834	869255834	483335306	483335306	483335306	97414778	97414778
36	54565946	120139292	120139292	120139292	-711442966	-711442966	288557034	-243927705	-243927705	756072295	684341937	684341937	684341937	612611579	612611579
37	249188261	-394798261	-394798261	605201739	176080807	176080807	176080807	495915184	495915184	495915184	160639961	160639961	160639961	-174635262	825364738
38	605565231	318251837	318251837	318251837	271695399	271695399	271695399	878937290	878937290	878937290	826010381	826010381	826010381	773083472	773083472
39	6574305	44469535	44469535	44469535	-13153992	-13153992	986846008	1275530730	275530730	275530730	296837772	296837772	296837772	-681855186	318144814
40	117054297	-779642933	-779642933	220357067	-377173974	-377173974	622826026	1002213380	2213380	2213380	293508629	293508629	706491371	410769382	410769382

41	699128233	798578541	798578541	798578541	347598042	347598042	347598042	527789660	527789660	527789060	484087281	484087281	484087281	400384922	400384922
42	593853968	1309359432	309359432	309359432	579632824	579632824	579632824	489285042	489285042	489285042	1215994003	215994003	215994003	-37277038	962722964
43	252383698	102847396	102847398	102847396	299097519	299097519	299097519	460754381	460754381	460754381	-166635005	-166635005	833364995	205975609	205975609
44	250461720	214845590	214845590	214845590	748955309	748955309	748955309	344890400	344890400	344890400	239233306	239233306	239233306	133576212	133576212
45	191034891	243756956	243756956	243756956	-346201671	-346201671	653798329	339831200	339831200	339831200	110161303	110161303	110161303	-119508594	880491406
46	744879509	904481243	904481243	904481243	346308249	346308249	346308249	732822320	732822320	732822320	301893433	301893433	301893433	-128835454	871164546
47	72572607	299166189	299166189	299166189	-23081825	23081825	976918375	1324890677	324890677	324890077	-217386097	-217386097	782613903	240337129	240337129
48	384785909	-550428807	-550428807	449573193	-366829883	-366829883	631170117	758106013	758106013	758106013	682788475	682788475	682788475	609470937	609470937
49	761653225	-445232072	-445232072	554767928	619358826	619358826	619358826	-142903257	-142903257	857090743	-482091312	-482091312	537908668	218720633	218720633
50	218438098	-654213575	-654213575	345786425	-1034593281	-34593281	965406739	210953752	210953752	210953752	-871011949	-871011949	328988051	-552977850	447022350
51	893902832	56972.,849	569725849	569725849	61825372	61825372	61825372	298407815	298407815	298407815	-486031331	-486031331	513968669	-270470477	729529523
52	52722068	-842680692	-842680692	357319308	275991498	275991498	275991498	84297232	84297232	84297232	-87019324	-87019324	912980676	741684120	741684120
53	485028697	282325272	282325272	282325272	944487065	944487065	944487065	1182957042	182957042	182957042	-758337092	-758337092	241682908	-698831226	300368774
54	628647195	451158804	451158804	451158804	-329779884	-329779884	670220116	109750924	109750924	109750924	-726184156	-726184156	273835844	-562079236	437920764
55	-408148031	-883190360	-883190350	116839640	56561028	56561028	943338972	831746565	831746568	831746566	168904858	168904858	613095132	-175556302	824443698

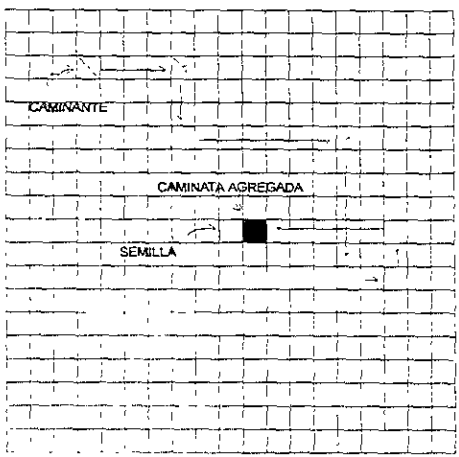
ANEXO 2

Fundamentos básicos de caminatas aleatorias

Desde los 80's se han realizado estudios acerca del comportamiento de procesos aleatorios, sin embargo los resultados han sido como una caja negra, ya que los algoritmos y técnicas de simulación se ocultan a lectores de estos estudios.

El reto es desarrollar algoritmos cada vez más eficientes y rápidos para la simulación de estos procesos. Todo esto con el fin de proponer cada vez nuevos modelos que motiven a una investigación más profunda de estos.

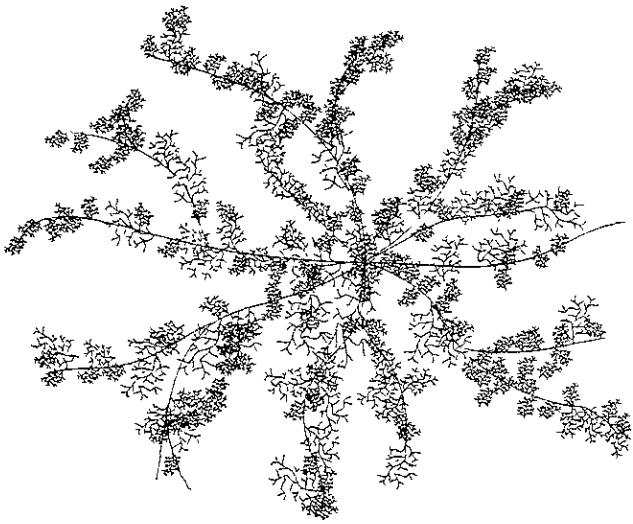
Las caminatas aleatorias en los procesos de agregación tienen su origen en el modelo de los investigadores Witten y Sander[27], los cuales, basándose en sus estudios proponen un modelo teórico y computacional (partícula o semilla y reticulado o enrejado), en el cual una partícula aparece en un lugar lejano a una semilla (la semilla reside en el centro del enrejado) y camina con movimientos aleatorios hasta llegar a un sitio adyacente a la semilla y pasa a formar de una estructura de grupo. Una nueva partícula se introduce, también en un punto distante y camina hasta unirse al grupo, si una partícula, durante su caminata aleatoria, toca los limites del enrejado, entonces es eliminada y otra partícula nueva se genera. Este proceso se repite hasta formar estructuras de tamaño suficientemente grande para obtener datos estadísticos. La siguiente figura muestra un modelo de caminatas al azar similar al proceso descrito anteriormente.



CAMINATAS AL AZAR

Debido a que una partícula recorre un camino aleatorio comenzando desde un punto a gran distancia de la semilla, y para cada punto que toca dentro de su recorrido, es necesario saber si ya esta colocada en un lugar adyacente a la semilla. De ser así, se adhiere al grupo; de no ser así, la partícula sigue su camino aleatorio hasta que ocurran una de dos cosas: o bien sale del área delimitada por el enrejado, o bien se coloca en un punto adyacente a la semilla y pasa a formar parte del grupo.

La siguiente figura muestra el resultado obtenido después de que un gran número de partículas se han adherido a la semilla en dos dimensiones.



Modelo de Witten y Sander en dos dimensiones

Es evidente que los cálculos deben ser muy numerosos antes de saber si alguno de los dos sucesos descritos antes ocurrieron. Esto significa mucho tiempo de procesamiento. Por esta razón, es necesario contar con algoritmos rápidos y eficientes que reduzcan el tiempo de procesamiento mediante herramientas computacionales.

Diversos modelos han surgido para investigar entre otras cosas la relación entre los diferentes parámetros de cada uno de ellos, por ejemplo: la utilización o no del enrejado, la relación existente entre la dimensión fractal y la dimensión euclidiana de las estructuras que se simularon. La relación entre el número de partículas que forman una estructura y la dimensión de ésta.

- Witten y Sander reportan que la correlación de la densidad del modelo que utilizan, cae dentro de un modelo cuyo comportamiento es exponencial, aclarando que limitan el estudio a un crecimiento dendrítico.
- P. Meakin[28], reporta que el radio de giro (R_g) de la estructura esta relacionada con el número de partículas *Ni* que forman dicha estructura, esta relación es:

$$R_g = N^{\beta}$$

Para un número de partículas N grande, en donde

$$\beta \approx 6 / 5 d$$

Donde d es la dimensión euclidiana del enrejado

El radio de giro de un número N de partículas en el agregado se define como

$$R_g^2 = \frac{1}{N} \sum_{i=1}^{N} \left(R_i - R_c(N)\right)^2$$

R_i es el vector de la i-esima partícula

$R_c(N)$ es el radio, vector del centro de la masa del fractal, considerando que cada partícula tiene una masa igual a 1.

En una función de recurrencia, R_g se calcula como:

$$R_c(N)^2 = \frac{N-1}{N} R_g{}^2(N-1) + \frac{N-1}{N^2}(R_c(N-1) - R_n)^2$$

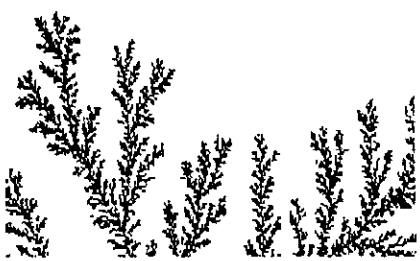
Donde:

$$R_c(N) = \frac{N-1}{N} R_c(N-1) + \frac{1}{N} R_n$$

Los resultados reportados por P. Meakin indican una relación entre la dimensión fractal o también llamada dimensión Hausdorff, y la dimensión clásica o euclidiana; dicha relación es expresada de la siguiente forma y es valida para 2 dimensiones y hasta 6 dimensiones:

$$D = 5d/6$$

Donde, D es la dimensión fractal o Hausdorff y d la dimensión clásica o euclidiana.
Los doctores V. Tchijov, Suemi Rodríguez Romo y S. Nachaev han propuesto un nuevo modelo desarrollado sobre la base del original de Witten y Sander. Este nuevo modelo se llama *Colored DLA mode*[29]*l*. Esta variación considera dos semillas de diferente color separadas por una cierta distancia d. Se considera una partícula viajera que tiene cierta probabilidad p de ser de un color, y una probabilidad 1-p de ser de otro color. Sí en su caminata aleatoria, la partícula toma un lugar adyacente a un color que no es el suyo, se elimina y una nueva empieza otra vez a caminar. Cuando toma un lugar adyacente a su color, la partícula se fija y el proceso se repite hasta formar estructuras fractales de un tamaño que permita obtener información estadística importante. En otro de sus trabajos, se realizan experimentos para encontrar la relación entre la distancia de separación de las semillas y la dimensión fractal que alcanzan las estructuras formadas.



Patrón similar al crecimiento de minerales como el oxido de magnesio en roca verde

REFERENCIAS DOCUMENTALES

[1] Benoit Mandelbrot, The fractal geometry of nature, W. H. Freeman, New York, 1983
[2] Benoit Mandelbrot, The fractal geometry of nature, W. H. Freeman, New York, 1983
[3] T.A. Witten, L.A. Sander, Physical Review Letter 41 (1981) 1400
[4] P. G. De Gennes, Scaling concepts in polymer physics. Cornell University Press, Ithaca, New York, 1979.
[5] T. A. Witten and L. M. Sander. Diffusion-limited aggregation, a kinetic critical phenomenon. Physical Review Letters, 47:1400-1403, 1981
[6] P. Gressberger, R. Procaccia, American Physical rev v.28 No. 4 Oct 1983
[7] Park, Stephen K. and Keith W. Miller. "Random Number Generators: Good Ones Are Hard to Find". Communications of the ACM, October 1988, p. 1192.
[8] Masaya Yamaguti, Masayoshi Hata, and Kigami Jun, Mathematics of Fractals, American Mathematical Society, 15 January 1998
[9] Kenneth Falconer, "Fractal Geometry: Mathematical Foundations and applications", John Wiley and Sons, 1990.
[10] Paul Meakin , Fractal, Scaling and growth far from equilibrium, The scaling structure of DLA, pp 198, Cambridge University Press 1998
[11] Murray Eden, A two dimensional hrowth process, In J. Neyman, editor, 4th Berkeley symposium on mathematics, statics and probably, pages 223-239, University of California Press, Berkeley, 1961, Volume IV: Biology and the problems of health
[12] T. A. Witten and L. M. Sander. Diffusion-limited aggregation, a kinetic critical phenomenon. Physical Review Letters, 47:1400-1403, 1981
[13] L. Pietronero, E. Tosatti (Eds) Fractal in Physics, Elseiver. Amsterdam, 1986
[14] T. Bishkek, Fractal grwth phenomena, 2ª Edition, World Scientific, Singapure, 1987
[15] V. Tchijov, A. Keller, S. Rodríguez Romo, Revista Mexicana de Física, Enero 1997
[16] AHO, A.V., HOPCROFT, J.E. Y ULLMAN, J.D.: "Estructuras de datos y algoritmos", Addison-Wesley 1988
[17] TENEMBAUM, A.M., LANGSAM, Y., AUGENSTEIN, M.A.: "Estructuras de datos en C", Prentice Hall 1993
[18] SPRUNGNOLI, R.: "Perfect Hashing Functions: A Single Probe Retrieving Method for Static Sets", Comm, ACM, 20 (11), 1977
[19] CICHELLI, R. J.: "Minimal Perfect Hash Functions Made Simple", Comm. ACM, 7 (1), 1980
[20] Carter, J. L., Wegman, M. N., "Universal Classes of Hash Functions", IBM Research Report RC 6395, Thomas J. Watson Research Center, Yorktown Heights, 1977
[21] Tenembaum, A.M., Langsam, Y., Augentein, M.A.: "Estructuras de datos en C", Prentice Hall 1993
[22] Weiss, M. A., Estructuras de datos y algoritmos, Addison-Wesley, 1995
[23] Tenembaum, A. M. Langsam, Augestein, Estructuras de datos en C, Prentice may, 1993
[24] Weiss, M. A., Estructuras de datos y algoritmos, Addison-Wesley, 1995
[25] J. L. Bentley and J. B. Saxe, Descomposable searching problems, Journal of Algorithms, 1980
[26] P. Flajolet, J. Francon, J. Vuillemin, Sequence of operations analysis for dynamic data structures, Journal of Algorithms, 1(2), 111-141
[27] Paul Meakin, Formation of fractals clusters and networks by irreversible diffusion-limited aggregation. Physical Review Letters, 47:1400-1403, 1981
[28] Paul Meakin , Fractal, Scaling and growth far from equilibrium, The scaling structure of DLA, pp 198, Cambridge University Press 1998.
[29] V. Tchijov, S. Nechaev, S. Rodríguez Romo, JETP Letter 64, (1996) 498