

Cálculo de la dimensión fractal de objetos 3D

Grado en Ingeniería Informática



Trabajo Fin de Grado

Autor:

Gacel Ivorra Rodríguez

Tutor/es:

Miguel Ángel Cazorla

Sergio Ramón



Universitat d'Alacant
Universidad de Alicante

Resumen

Este Trabajo de Fin de Grado consiste en la investigación y experimentación del cálculo de la dimensión fractal por medio del algoritmo de conteo de cajas, en inglés “Box Counting”, aplicado sobre nubes de puntos 3D mediante la implementación de una herramienta software que se encargue de hacer los cálculos necesarios.

Previamente se hará una introducción al mundo de los fractales, los objetos matemáticos que conforman la Geometría de la Teoría del Caos, la geometría fractal y las aplicaciones que está puede tener en diferentes contextos.

Se ha detectado que no existe documentación ni publicaciones donde podamos ver que se haya intentado realizar este tipo de cálculos sobre objetos representados mediante nubes de puntos, por lo que la mayor parte del trabajo se centra en, una vez desarrollado el algoritmo, realizar una extensa experimentación tratando de obtener la mejor solución posible variando los parámetros que modifican el comportamiento de éste, y realizando una comparativa de resultados sobre estas distintas variantes.

Para la experimentación se utilizan algunas formas cuya dimensión fractal es conocida teóricamente y se pretende en este trabajo ajustar el algoritmo de tal manera que sea capaz de dar un resultado igual o muy aproximado a la dimensión teórica del objeto en cuestión.

Una vez finalizada la fase de ajuste del algoritmo, se realizarán pruebas sobre modelos de objetos reales con los que poder verificar que el comportamiento y los resultados del algoritmo son igualmente válidos y que muestran coherencia con los fundamentos teóricos a los que apunta la geometría fractal.

Índice

Resumen	2
Índice	3
Capítulo 1: Motivación	6
Capítulo 2: Objetivos	8
Capítulo 3: Introducción	10
1.- ¿Qué es un fractal?	10
2.- Características de un fractal	10
3.- ¿Qué es la geometría fractal?	11
4.- Dimensión fractal	12
Capítulo 4: Estado del arte	13
1.- Análisis de fractales	14
2.- Aplicaciones del análisis fractal	16
2.1.- Distribución urbana	16
2.2.- Comunicaciones	16
2.3.- Medicina	17
2.4.- Economía	17
2.5.- Informática	17
2.6.- Geología	18
2.7.- Industria	18
2.8.- Ámbito militar	18
2.9.- Medio ambiente	18
2.10.- Música	18
Capítulo 5: Metodología	20
Capítulo 6: Herramientas utilizadas	21
1.- Recursos hardware	21
2.- Plataforma y librerías software	21
2.1.- Lenguaje de programación	21
2.2.- Point Cloud Library (PCL)	21
2.3.- MeshLab	22
2.4.- Librería QT4	22
2.5.- Gnuplot y Gnuplot-iostream	23
3.- Generador de fractales	24
3.1.- Tetraedro de Sierpinski	24
3.2.- Pirámide de Sierpinski	25
3.3.- Terrenos rugosos	26

Capítulo 7: Algoritmo Box Counting	27
Capítulo 8: Desarrollo del algoritmo	29
1.- Implementación del algoritmo	29
2.- Experimentación y ajuste del algoritmo	30
2.1.- Fase 1	30
2.1.1.- Resultados	31
2.1.2.- Conclusiones	33
2.2.- Fase 2	38
2.2.1.- Problemas a resolver	38
2.2.2.- Modificaciones	39
2.2.3.- Resultados	39
2.2.4.- Conclusiones	43
2.3.- Fase 3	44
2.3.1.- Problemas a resolver	44
2.3.2.- Modificaciones	44
2.3.3.- Resultados	44
2.3.4.- Conclusiones	48
2.4.- Fase 4	49
2.4.1.- Problemas a resolver	49
2.4.2.- Modificaciones	49
2.4.3.- Resultados	49
2.4.4.- Conclusiones	56
2.5.- Fase 5	57
2.5.1.- Problemas a resolver	57
2.5.2.- Modificaciones	58
2.5.3.- Resultados	59
2.5.4.- Conclusiones	62
2.6.- Fase 6	63
2.6.1.- Problemas a resolver	63
2.6.2.- Modificaciones	63
2.6.3.- Resultados	63
2.6.4.- Conclusiones	65
2.7.- Fase 7	66
2.7.1.- Problemas a resolver	66
2.7.2.- Modificaciones	66
2.7.3.- Resultados	67
2.7.4.- Conclusiones	69
2.8.- Fase 8	70
2.8.1.- Problemas a resolver	70
2.8.2.- Modificaciones	70
2.8.3.- Resultados	71

2.8.4.- Conclusiones	75
3.- Experimentación y validación de la solución obtenida	76
3.1.- Independencia de los resultados frente a el tamaño de la figura	76
3.2.- Experimentación con modelos reales	77
3.3.- Tiempos de ejecución	81
Capítulo 9: Conclusiones	83
Bibliografía	84

Capítulo 1: Motivación

Este proyecto viene motivado por el impulso de desarrollar y poner en práctica técnicas que hagan uso de lo que se conoce como geometría fractal.

La geometría fractal es un campo relativamente reciente de las matemáticas que aporta un enfoque muy interesante ya que es capaz de “decodificar” o reconocer patrones existentes en objetos aparentemente caóticos los cuales no se podían llegar a describir mediante matemáticas tradicionales.

Como más adelante se explicará en este documento y ya es conocido por muchas personas que han trabajado o estudiado acerca de este enfoque matemático, existe un enorme potencial en el aspecto científico sobre esta materia ya que a raíz de éste se han obtenido unas de las técnicas matemáticas que más se aproximan a lo que sería una descripción natural y simple de nuestra realidad, el universo y la naturaleza en su conjunto.

"Las nubes no son esferas, las montañas no son conos y los litorales no son circulares, lo mismo que los relámpagos no viajan en línea recta" (Benoît Mandelbrot).

Desde el punto de vista científico y objetivo, no hay ningún objeto o forma física existente que pueda representarse de manera estricta mediante líneas rectas, planos, esferas perfectas, etc. ¿Por qué es cierta esta afirmación? Se ha demostrado desde hace muchos años que, haciendo zoom sobre cualquier objeto aparentemente plano, a medida en que se van obteniendo mayores ampliaciones, se comienzan a observar más y más rugosidades sobre éste, y si pudiéramos llegar hasta el fondo veríamos que ese objeto al final es un conjunto de átomos agrupados formando el objeto original pero que jamás podríamos considerar de manera objetiva como una superficie lisa.

Es una manera simple de explicar algo extremadamente complejo, pero podemos concluir que se hace ver una carencia en las matemáticas tradicionales para describir formas y fenómenos existentes en la naturaleza como pueden ser un árbol, una nube, una montaña, etc. de una manera relativamente sencilla.

Teniendo en cuenta todo lo anterior, siendo conscientes de la sabiduría innata de la naturaleza, y sabiendo que gran parte de la tecnología y los grandes inventos desarrollados por los seres humanos se basan en imitar estos patrones naturales, se hace una obligación el explorar y seguir desarrollando ideas basadas en esta “matemática de lo natural”.

A continuación, se muestran una serie de imágenes de fractales naturales donde se observa que la naturaleza utiliza los mismos patrones repetitivos desde la escala macro a la micro, en todos los niveles (esto se aplica también para todo tipo de patrones existentes, no solo de formas):

Col Romanesca



Copo de nieve



Árbol



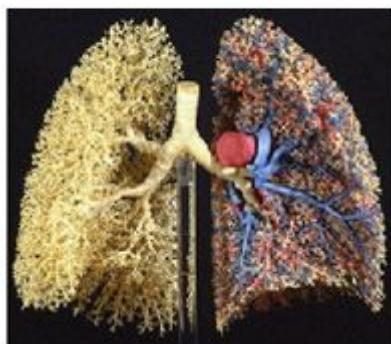
Rayo



Desembocadura del río Lena, Siberia



Molde del árbol bronquial humano



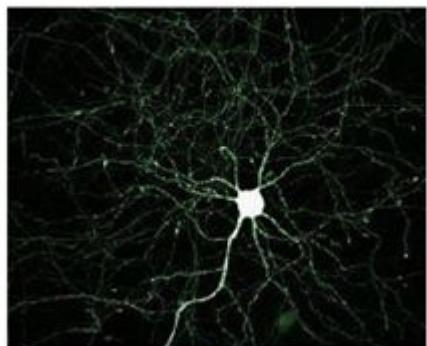
Sistemas nubosos



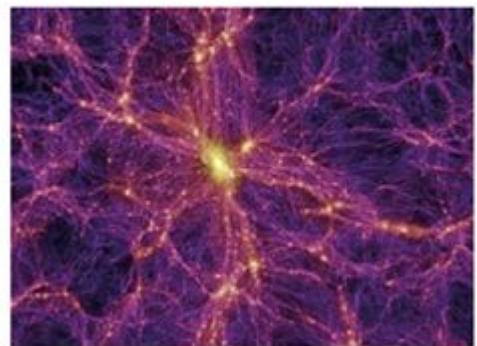
Vía Láctea



Red de neuronas



Cúmulo de galaxias y filamentos



Capítulo 2: Objetivos

Los objetivos principales de este trabajo son:

- Implementar el algoritmo Box Counting aplicado sobre figuras representadas por nubes de puntos 3D.
- Investigar sobre la factibilidad de aplicar técnicas de cálculo de dimensión fractal sobre objetos representados mediante nubes de puntos.
- Realizar una experimentación sobre qué solución del algoritmo proporciona los mejores resultados.
- Establecer las premisas a seguir para aplicar Box Counting sobre nubes de puntos 3D mediante las conclusiones obtenidas de la investigación y experimentación realizada.
- Obtener una herramienta software que permita calcular de manera fiable la dimensión fractal de objetos 3D.
- Profundizar y divulgar en la medida de lo posible sobre técnicas de geometría fractal.

Aunque existe una motivación de fondo por utilizar la dimensión fractal como una característica de peso para realizar clasificación de objetos 3D, queda fuera del ámbito de este proyecto la aplicación de dicha característica sobre este ni sobre ningún otro fin en concreto, ya que si se realiza el trabajo con éxito se conseguirá una utilidad que sirva para cualquier campo de aplicación en el que se desee utilizar.

Las diferentes aplicaciones de la medida conocida como dimensión fractal serán detalladas más adelante a lo largo de este documento.

Capítulo 3: Introducción

Para empezar con el desarrollo de este trabajo, es necesario explicar una serie de conceptos claves para su entendimiento.

1.- ¿Qué es un fractal?

Un **fractal** es un ente geométrico cuya estructura básica se repite a diferentes escalas. El término fue propuesto por el matemático Benoît Mandelbrot en 1975 y deriva del latín *fractus*, que significa quebrado o fracturado.

Un **fractal “ideal”** es una figura geométrica que los matemáticos crean por medio de un algoritmo iterativo o regla repetitiva que tiene una forma, bien sea sumamente irregular, bien sumamente interrumpida o fragmentada, y sigue siendo así a cualquier escala que se produzca el examen. Los fractales matemáticos cumplen con la propiedad de auto-similitud exacta.

Un **fractal “natural”** es un elemento de la naturaleza que puede ser descrito mediante la geometría fractal. Las nubes, las montañas, el sistema circulatorio, las líneas costeras o los copos de nieve son fractales naturales. Esta representación es aproximada, pues las propiedades atribuidas a los objetos fractales ideales, como el detalle infinito, tienen límites en el mundo físico.

2.- Características de un fractal

Un objeto tiene tendencia fractal cuando es demasiado irregular para ser descrito en términos geométricos tradicionales.

De forma general, podemos caracterizar los fractales mediante las siguientes propiedades:

- Tienen una estructura compleja a cualquier resolución.
- Tienen una dimensión no entera.
- Tienen un perímetro de longitud que tiende a infinito, pero un área limitada.
- Son auto-similares e independientes de la escala

La característica más destacada es la auto-similitud. Según Benoît Mandelbrot, un objeto es autosimilar o autosemejante si sus partes tienen la misma forma o estructura que el todo, aunque pueden presentarse a diferente escala y pueden estar ligeramente deformadas.

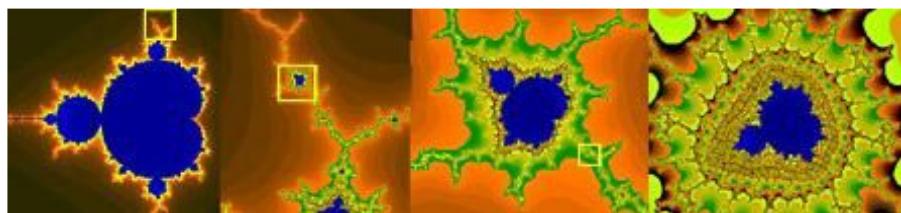
Los fractales pueden presentar tres tipos de auto-similitud:

- **Autosimilitud exacta:** Este es el tipo más restrictivo de auto-similitud: exige que el fractal parezca idéntico a diferentes escalas. A menudo la encontramos en fractales

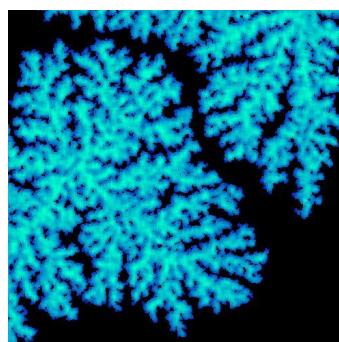
definidos por sistemas de funciones iteradas (IFS), un ejemplo es el triángulo de Sierpinski:



- **Cuasautosimilitud:** Exige que el fractal parezca aproximadamente idéntico a diferentes escalas. Los fractales de este tipo contienen copias menores y distorsionadas de sí mismos. Los fractales definidos por relaciones de recurrencia son normalmente de este tipo. En este grupo encontramos, por ejemplo, el famoso fractal de Mandelbrot:



- **Autosimilitud estadística:** Es el tipo más débil de auto-similitud: se exige que el fractal tenga medidas numéricas o estadísticas que se preserven con el cambio de escala. Los fractales aleatorios son ejemplos de fractales de este tipo. A continuación, un fractal con autosimilitud estadística generado por el proceso de agregación limitada por difusión:



3.- ¿Qué es la geometría fractal?

La geometría fractal ofrece un modelo alternativo que busca una regularidad en las relaciones entre un objeto y sus partes a diferentes escalas. Esta forma de regularidad no precisa el encorsetamiento del objeto en otras formas geométricas que, aunque elementales, no dejan de ser externas al mismo, sino que busca la lógica interna del propio objeto mediante relaciones intrínsecas entre sus elementos constitutivos cuando estos se examinan a diferentes escalas. De esta forma no se pierden ni la perspectiva del objeto global, ni del aspecto del mismo en cada

escala de observación. La geometría fractal busca y estudia los aspectos geométricos que son invariantes con el cambio de escala.

4.- Dimensión fractal

Para dar una idea de qué es la dimensión fractal y qué significado tiene veámoslo con un ejemplo:

Si consideramos que una hoja de papel es un objeto bidimensional podríamos coger la hoja y arrugarla hasta formar una bola. El objeto tendría volumen, pero no sería tridimensional porque la bola estaría llena de huecos y discontinuidades. Para convertirla en una esfera tendríamos que hacer un largo número de interpolaciones lineales. Todo esto explica porqué es tan difícil modelar la naturaleza con la geometría euclíadiana.

La **dimensión fractal** intenta medir en qué grado un objeto 2D llena el espacio 3D, o un objeto de dimensión 1 se asemeja a una superficie 2D.

Esta última definición es perfecta cuando estamos hablando de objetos geométricos, pero esta misma idea también se da con una infinidad de patrones de toda clase que se pueden presentar como una forma geométrica, por ejemplo una gráfica de datos. De esta manera podríamos obtener una estimación de la dimensión fractal de la curva del valor de las acciones en bolsa durante diferentes etapas y podríamos hacer una comparativa en el tiempo para luego realizar predicciones. Se puede asignar una dimensión fractal a una trayectoria (1D) en el plano (2D), o incluso en el espacio (3D), por ejemplo, para analizar y comparar los movimientos de un ratón en respuesta a diferentes medicamentos y poder tener un índice de la excitación o relajación producida en respuesta a cada medicamento. Puede extrapolarse esta medida a infinidad de patrones a los que se nos ocurra aplicarlo, muchos conocidos y muchos otros aún por descubrir e investigar, y se ha demostrado que funciona muy bien en diferentes campos de la ciencia.

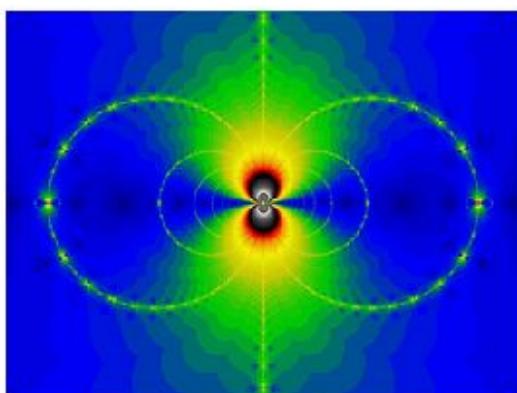
En resumen, la dimensión fractal es una medida capaz de captar la esencia de un patrón repetitivo, parecido a la manera en que la naturaleza se manifiesta en la realidad, tanto a nivel de formas (ríos, montañas, árboles, plantas, vasos sanguíneos, nubes, etc.), como de patrones de todo tipo que seamos capaz de extrapolar a esta matemática.

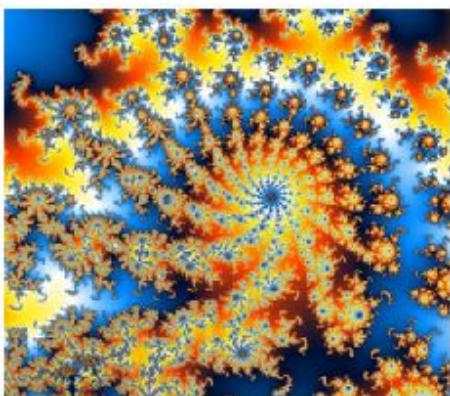
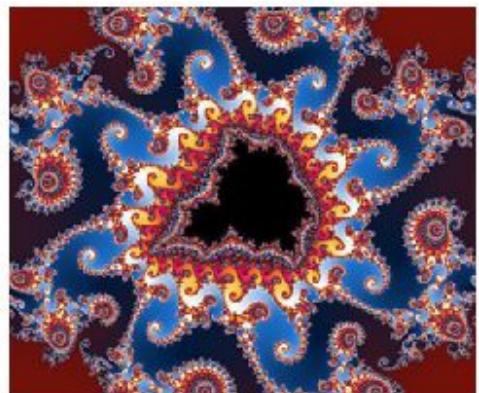
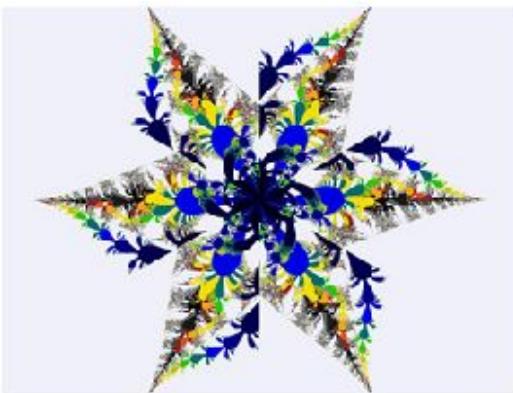
Capítulo 4: Estado del arte

En matemática clásica, la geometría se basaba en rectas, curvas, círculos o polígonos con los que sólo se podía medir objetos de formas simples. Pero muchos elementos de la naturaleza no se asemejan a esas formas simples, como por ejemplo, las redes nerviosas, los vasos sanguíneos, los árboles, los copos de nieve, las nubes y hasta la misma Vía Láctea. Estas formas naturales sirvieron de inspiración a estudiosos y científicos que, desde finales del siglo XIX, buscaban analizar sus grados de complejidad e irregularidad.

Fue el matemático Benoît Mandelbrot, considerado como el padre de la geometría fractal, el que introdujo el término "fractal" con sus experimentos de computación hacia 1974. Mandelbrot tomó esta palabra de raíz latina para nombrar este nuevo lenguaje matemático creado para medir la singularidad de las formas naturales. Los fractales pertenecen a un universo diferente al de la geometría clásica euclíadiana que regida por figuras regulares y simples como planos, círculos, triángulos o cuadrados.

La aparición y avance de la computación en las últimas décadas posibilitó la producción gráfica de fractales que requieren cálculos altamente complejos. De esta forma, matemáticos, científicos e incluso artistas encontraron un nuevo medio de expresión e investigación. Es un hecho curioso, ya que la teoría fractal ha sido un punto de encuentro entre matemáticas y arte debido a que las imágenes surgidas de diferentes conjuntos fractales resultan muy bellas y curiosas. A continuación se muestran algunos ejemplos de arte fractal generado mediante fórmulas recursivas:





Estas increíbles, bellas y complejas formas que vemos, son unos pocos ejemplos de la infinidad que existen (literalmente), son generadas mediante fórmulas recursivas muy sencillas, las cuales llegan a producir patrones muy complejos. Es curioso como programando estas fórmulas, obtenemos figuras que pueden parecer muy abstractas o caóticas, al mismo tiempo, contienen patrones matemáticos muy complejos de auto-similitud y orden. Es una representación muy parecida a la vida y la naturaleza, construyendo de lo simple a lo complejo.

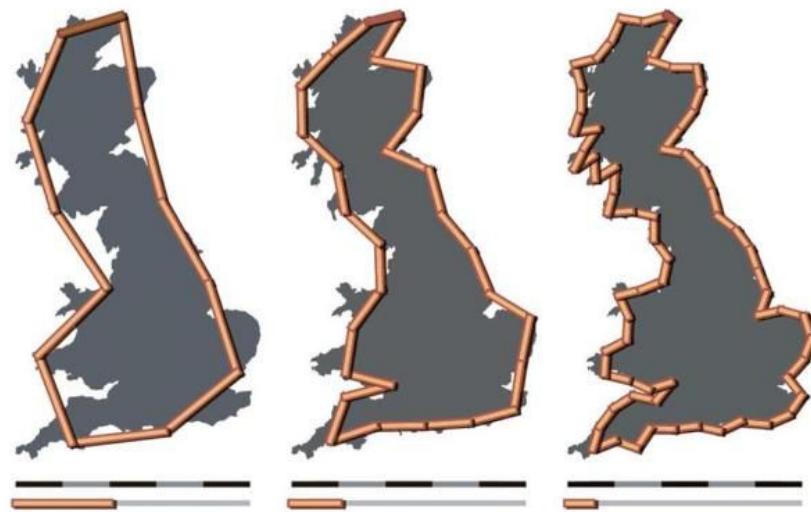
Para más información, buscar acerca de los conjuntos de Mandelbrot y de Julia.

Surgen pues, a raíz de esta nueva teoría matemática, dos grandes ramas de la ciencia: la **generación de fractales** y el **análisis**. Este trabajo se sitúa en el ámbito del análisis fractal ya que estamos tratando de calcular la dimensión fractal de un objeto, aunque también se han implementado algoritmos de generación de fractales para tener figuras de referencia con las que realizar experimentaciones, sabiendo el resultado teórico de estas.

1.- Análisis de fractales

Mandelbrot lanzó una pregunta de difícil respuesta: ¿Cuánto mide la costa de Gran Bretaña? Una forma de medirla consiste en ir trazando rectas de 200 km, por ejemplo, que vayan desde los diferentes vértices que nos muestra el accidentado relieve de la costa. La suma de todas esas rectas nos dará una medida aproximada (unos 2.400 km.). Si queremos que la medida sea más

precisa lo que tenemos que hacer es ir reduciendo el tamaño de las rectas que utilizamos. Si en vez de utilizar rectas de 200 km, utilizamos rectas de 50 km, la aproximación será más precisa y la media mayor (unos 3.400 km). Y así podemos seguir, utilizando rectas más y más pequeñas y consiguiendo resultados más precisos y mayores. Aparece un problema: ¿Hasta cuándo debemos seguir reduciendo el tamaño de las rectas para conseguir, no una aproximación, sino la medida real? Hasta el infinito, pues siempre podremos hacer una recta más corta que la anterior. Esto se basa en que cualquier número es infinitamente divisible en nuevos números. Por lo tanto, la respuesta a la pregunta de Mandelbrot es que la costa de Gran Bretaña mide infinitos kilómetros.



Según esta conclusión, sería entonces imposible hacer una comparación de la longitud de diferentes costas, ya que todas tienen una longitud infinita.

Lo que sí podríamos hacer es obtener un valor de cómo varía la longitud según el tamaño del elemento de medición utilizado. Esto aproxima lo que viene a ser la dimensión fractal de la línea de costa. De esta forma, podremos obtener la dimensión fractal de diferentes costas y tener una medida descriptiva de cada una, permitiendo compararlas.

A continuación, presentamos una tabla con la dimensión fractal de las costas europeas:

	Escala			Dimensión Fractal
	1/100,000	1/1,000,000	1/3,000,000	
Bélgica	98	72	65	1,49
Dinamarca	4488	3780	3614	1,24
Alemania	1864	1586	1551	1,2
Grecia	5333	4596	4387	1,22
España	6567	5306	5110	1,29
Francia	7205	4739	4682	1,54
Irlanda	5148	3842	3295	1,56
Italia	7409	6188	6088	1,22
Holanda	861	761	745	1,16
Portugal	924	843	836	1,11
Reino Unido	12911	11698	11577	1,37
Total Europa	55806	43410	41950	1,33

Esto es un ejemplo de aplicación de la dimensión fractal como característica descriptiva de una forma.

2.- Aplicaciones del análisis fractal

En este apartado se describen diferentes aplicaciones de la geometría fractal utilizadas hoy en día.

2.1.- Distribución urbana

Se puede calcular la dimensión fractal de una determinada zona urbana. Mediante la figura formada por sus calles y edificios, puede determinarse el nivel de aglomeración existente.

Con técnicas de geometría fractal podríamos obtener los niveles de crecimiento de cada ciudad, pudiendo hacer así comparativas entre ellas.

Se pueden relacionar datos de cada región en función a la dimensión fractal, como son índices de contaminación, índices de delincuencia, índices de tráfico y colapso, índices marginalidad, etc.

Mediante el análisis de toda esta información se pueden obtener cuales son los patrones de crecimiento de las ciudades que proporcionan mayor armonía.

2.2.- Comunicaciones

Se ha demostrado que el tráfico de paquetes a lo largo y ancho de Internet sigue un modelo que se comporta como un fractal, de hecho, analizando gráficas del tráfico de datos a través del tiempo, se puede apreciar que tiende a tener la propiedad de auto-similitud. Gracias a estos modelados, se pueden disminuir las pérdidas de paquetes producidas por diversos motivos, y mejorar así el rendimiento de la red.

Otra utilidad de los fractales en el mundo de las comunicaciones está referida a la fabricación de antenas. Y es que una antena con forma de algún objeto fractal (como el que hemos descrito antes) puede mejorar el rendimiento del equipo en mucho menos espacio.

2.3.- Medicina

Nuestros pulmones, el sistema sanguíneo, el cerebro, tienen estructuras fractales. Se ajustan a la propiedad de auto-semejanza en los cambios de escala. La superficie de los pulmones maximiza la capacidad de intercambio. Por ejemplo, los bronquios en sus siete primeras ramificaciones tienen una dimensión fractal diferente a la dimensión de las ramificaciones de mayor nivel. Existen relaciones, por ejemplo, de la dimensión fractal de los vasos sanguíneos de un cierto órgano del cuerpo con posibles enfermedades que pueda padecer, como el cáncer. En cardiología se estudia la variabilidad de la dimensión fractal del árbol coronario izquierdo en pacientes con enfermedad arterial oclusiva severa.

2.4.- Economía

Se ha comprobado que los cambios de los precios de los activos de las empresas tienen un comportamiento fractal, y por tanto, pueden ser estudiados utilizando el conocimiento que tenemos de ellos. Esto frena todo intento de encasillar en modelos estadísticos los mercados bursátiles, y basarse en modelos fractales para obtener mejores predicciones de cambio.

2.5.- Informática

En informática existen diversas aplicaciones como la compresión fractal. Este es un método de compresión con pérdida para imágenes digitales, basado en fractales. El método es el más apropiado para texturas e imágenes naturales, basándose en el hecho de que partes de una imagen, a menudo, se parecen a otras partes de la misma imagen. Los algoritmos fractales convierten estas partes en datos matemáticos llamados «códigos fractales» los cuales se usan para recrear la imagen codificada.

Mediante el análisis fractal de objetos naturales, se puede después tratar de reproducirlos computacionalmente. Por ejemplo, se han utilizado medidas y técnicas fractales ampliamente para la producción de efectos especiales y escenarios como montañas, vegetación, nubes, etc.



2.6.- Geología

Las técnicas de análisis fractal ayudan a entender las redes de fracturas de los macizos rocosos y las microestructuras de los minerales. Se hacen análisis de patrones sísmicos, fenómenos de erosión, modelos de formaciones geológicas, etc.

2.7.- Industria

Durante el zincado o galvanizado de superficies, se produce una distribución idéntica al crecimiento de los corales marinos, con una dimensión fractal del orden de 1'7.

2.8.- Ámbito militar

La Geometría fractal ha demostrado su utilidad en la detección de almacenamientos bajo el agua o en la trazabilidad del movimiento de submarinos, como en análisis medioambiental para la determinación del origen y ruta seguida por nubes de lluvia ácida.

2.9.- Medio ambiente

Se ha comprobado que la propagación de un incendio forestal en una plantación ordenada de árboles sigue una conducta fractal.

La propia distribución de los bosques sigue estructuras fractales.

2.10.- Música

Un equipo de investigación ha completado el análisis fractal de las partituras de cerca de 2.000 composiciones musicales escritas por más de 40 compositores durante los últimos 400 años, en una gran variedad de géneros musicales occidentales.

Este análisis ha permitido descubrir una fórmula matemática que gobierna los patrones rítmicos por los que se rige toda pieza musical mínimamente convencional.

Una de las cosas que en las últimas dos décadas han constatado musicólogos, psicólogos y otros estudiosos de la física y las matemáticas subyacentes en la música, es que la distribución tonal y la distribución de volumen siguen en la música patrones matemáticos predecibles.

Podríamos seguir enumerando aplicaciones de la geometría fractal pero nunca acabaríamos. Aquí se han expuesto algunos ejemplos, pero dadas las particularidades de este enfoque matemático y su capacidad de captar la información de forma similar a como la genera la propia naturaleza, en este momento el horizonte de aplicaciones posibles no tiene límites.

Capítulo 5: Metodología

Veamos los pasos seguidos para el desarrollo de este trabajo:

En una primera fase, se ha realizado un estudio sobre las técnicas disponibles para el cálculo de la dimensión fractal. De forma paralela, se ha buscado información sobre trabajos previos realizados sobre objetos representados como nube de puntos para saber cuál era el punto de partida y las ideas que se habían desarrollado previamente y de las que pudiéramos hacer uso.

Al no encontrar nada similar que pudiera servir de referencia, se ha optado por desarrollar el algoritmo Box Counting que, aunque no sea el mejor de todos a nivel de resultados, es el más extendido en todas las áreas debido a que es más sencillo de implementar. Proporciona buenos resultados y es rápido, lo que nos va a permitir desarrollar una experimentación más fluida y poder abarcar más pruebas.

Una vez implementado el software necesario, se ha realizado una experimentación bastante extensa, aplicando el algoritmo sobre objetos con dimensión fractal conocida, analizando los resultados mediante gráficas, y buscando qué solución proporciona los mejores resultados.

También se realizarán pruebas sobre familias de objetos cuya dimensión fractal exacta no es conocida pero sabiendo que siguen una escala de rugosidad que debería verse reflejada por el resultado obtenido para cada uno de ellos.

La solución final obtenida no proporcionará la dimensión fractal de forma exacta, pero seguiría siendo un algoritmo válido si es capaz de conservar la escala ya que permitiría hacer comparaciones entre objetos.

Capítulo 6: Herramientas utilizadas

1.- Recursos hardware

A continuación, se detallan las características del equipo utilizado:

- Procesador: Intel Core i5 M450 @ 2.40GHz 64 bits
- Memoria RAM: 4GB DDR2 800MHz
- Sistema operativo: Linux Mint 17.2 Rafaela

2.- Plataforma y librerías software

2.1.- Lenguaje de programación

Se ha decidido realizar el desarrollo mediante el lenguaje de programación C++ debido a que es un lenguaje muy potente, estable y que produce software muy rápido. También porque es el lenguaje nativo de la PCL.

2.2.- Point Cloud Library (PCL)

La herramienta conocida como PCL (Point Cloud Library) consiste en una librería desarrollada para el tratamiento completo de nubes de puntos 3D, liberada bajo licencia BSD, que cuenta con varios métodos de alta eficiencia computacional.

PCL surgió ante la necesidad de que los robots tengan la capacidad de percibir el mundo tal como lo hacemos los seres humanos, es decir, que puedan determinar las diferentes características y detalles que el ojo humano puede observar. Esta librería se ha potencializado debido al gran avance y al bajo costo de los sensores de adquisición de imágenes 3D, en el que cabe destacar en sensor Kinect® de la compañía Microsoft el cual se basa en la tecnológica PrimeSense.

Esta librería dispone de varios módulos que proporcionan métodos computacionales para manipulación de las nubes de puntos, entre ellos los que se han utilizado para este trabajo:

- **Filtros** (libpclfilters): PCL posee métodos de filtrado digital como por ejemplo eliminadores de datos atípicos, filtro tipo voxel, eliminador con condición, suavizado, índices de extracción y proyecciones. Debido al volumen de datos que genera una nube de puntos también existe la posibilidad de utilizar filtros que además de eliminar valores

atípicos, hacen una gran reducción de datos, permitiendo así una mayor rapidez en la computación.

- **KdTree** (libpclKdtree): Esta biblioteca consiste en una estructura tipo árbol que almacena un conjunto de puntos k-dimensional con el fin de realizar búsquedas eficientes del vecino más cercano.
- **Reconstrucción de Superficies** (libpclsurface): Esta biblioteca se utiliza para mejorar la visualización de un modelo tridimensional procesando los puntos de una nube para obtener una representación en malla o una superficie alisada.
- **Visualización** (libpclvisualization): Esta librería permite la rápida visualización de algoritmos que operan sobre nubes de puntos 3D. Cuenta con métodos de procesamiento configuración de propiedades visuales como colores, tamaños de punto y opacidad.

2.3.- MeshLab

MeshLab es un software de código abierto para procesar y editar mallas 3D. Proporciona un conjunto de herramientas para editar, limpiar, inspeccionar, renderizar, texturizar y convertir mallas. Ofrece características para procesar datos en bruto producidos por otras herramientas o dispositivos de digitalización 3D y para preparar modelos para la impresión 3D.

En este trabajo se ha utilizado para poder visualizar y renderizar figuras durante la generación de fractales y como herramienta de apoyo para algunas operaciones durante la experimentación.

2.4.- Librería QT4

Qt es una amplia plataforma de desarrollo gratuita y de código abierto que incluye clases, librerías y herramientas para la producción de aplicaciones de interfaz gráfica en C++ que pueden operar en varias plataformas.

Una de esas herramientas es Qt Creator, un IDE que facilita la creación de formularios, botones y ventanas de diálogo con el uso del ratón.

Por todo ello, es adecuada para poder crear una interfaz sencilla donde visualizar las nubes de puntos y realizar algunas operaciones sobre ellas.

2.5.- Gnuplot y Gnuplot-iostream

Gnuplot es una utilidad gráfica de línea de comandos para Linux, OS / 2, MS Windows, OSX, VMS y muchas otras plataformas. Es gratuita y fue creada para permitir a los científicos y estudiantes visualizar funciones matemáticas y datos de forma gráfica.

Gnuplot-iostream es una interfaz que permite que Gnuplot sea controlado desde un programa en C++.

Esta herramienta ha sido utilizada para representar los resultados del algoritmo implementado en una gráfica XY.

3.- Generador de fractales

Para poder realizar el estudio aplicando el algoritmo Box Counting en 3D, se ha implementado un software que genera dos fractales simples, cuya dimensión fractal es conocida. Servirán de modelos con los que ir ajustando el algoritmo cada vez más.

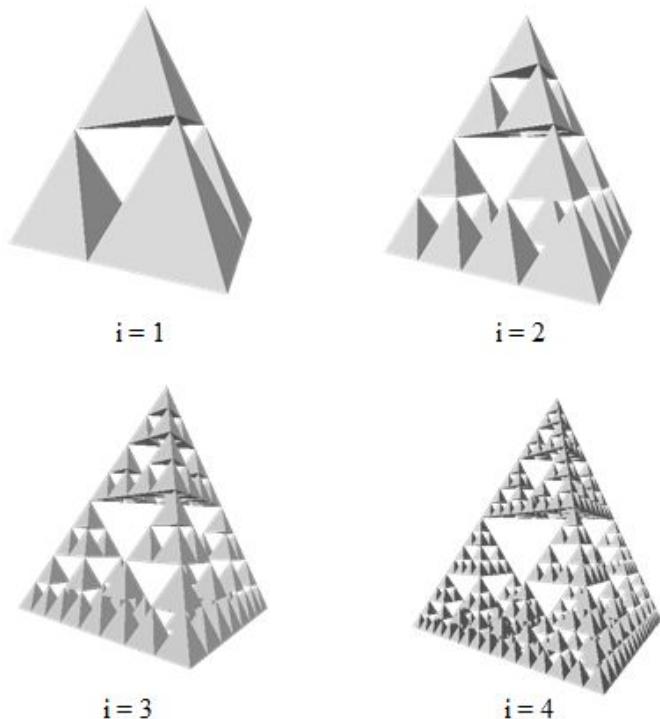
Las figuras generadas estarán representadas como nubes de puntos, los cuales representan cada uno de los vértices de la figura, en formato PLY.

3.1.- Tetraedro de Sierpinski

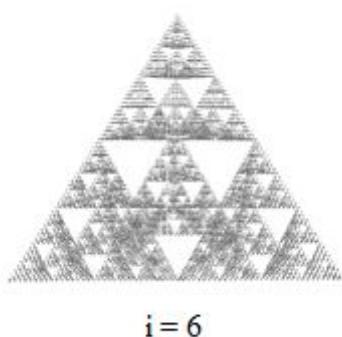
El tetraedro de Sierpinski es la versión 3D del triángulo de Sierpinski, una de las figuras fractales más conocidas. Se ha generado de la forma más sencilla y rápida computacionalmente mediante el siguiente algoritmo iterativo:

1. Partimos de un tetraedro equilátero de lado $l = 100$ representado por sus 4 vértices.
2. Se realizan tres copias del tetraedro inicial realizando un desplazamiento a los puntos de cada uno para que entre los 4 tetraedros formen un tetraedro que duplica las medidas del tetraedro inicial (no se duplican los puntos que tengan las mismas coordenadas).
3. Vamos a generar **dos versiones del tetraedro** que se diferencian en este punto según cuál de las dos decisiones posibles se tomen. Nos viene bien tener estas dos versiones para poder demostrar que el tamaño del objeto no es relevante para el cálculo de la dimensión fractal:
 - a. Se dividen entre 2 las coordenadas de todos los puntos para mantener la escala inicial (en todas las iteraciones el tetraedro tendrá $l = 100$).
 - b. No se realiza ninguna operación en este punto, por lo que el tetraedro resultante de cada iteración doblará las medidas del de la anterior.
4. Se comienza en el punto 2 hasta realizar todas las iteraciones que se deseen realizar.

En las imágenes se muestran los tetraedros representados por polígonos y no por los vértices para que sea más claro a la vista, aunque posteriormente trabajaremos solo con los vértices como una nube de puntos.



A medida que aumentan las iteraciones utilizadas, más fácilmente puede diferenciarse la forma solamente visualizando los vértices.

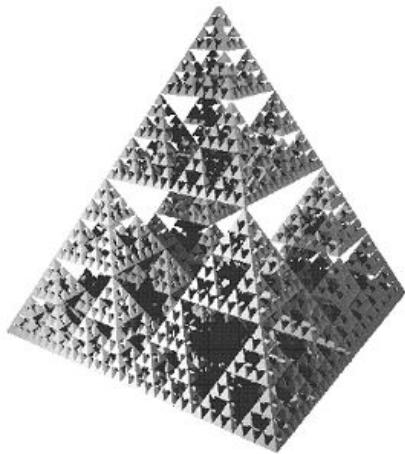


Sabemos que la dimensión fractal de este objeto se puede calcular como:

$$D = \frac{\log(4)}{\log(2)} = 2$$

3.2.- Pirámide de Sierpinski

La pirámide de Sierpinski es una variante del tetraedro, pero en lugar de partir de un tetraedro equilátero, se parte de una pirámide. La generación se realiza siguiendo los mismos pasos descritos para el tetraedro, obteniendo en cada iteración una pirámide mayor formada por 5 pirámides (cuatro en la base y una en la parte superior) correspondientes a la obtenida en la iteración anterior.



$i = 6$

Sabemos que la dimensión fractal de este objeto se puede calcular como:

$$D = \frac{\log(5)}{\log(2)} = 2.3219$$

3.3.- Terrenos rugosos

Mediante el algoritmo fBM (Fractal Brownian Motion) del software MeshLab se han generado 13 terrenos de los cuales no sabemos su dimensión fractal exacta pero sí que siguen una escala de rugosidad. A continuación se muestran varios de esos modelos generados, no todos ya que no es relevante, simplemente aquí se expone que tenemos una serie de modelos de los que sabemos que varía su rugosidad y los resultados del algoritmo deberán mantener esa escala reflejada:



Capítulo 7: Algoritmo Box Counting

El algoritmo conocido como Box Counting es el algoritmo más extendido en todas las ramas del conocimiento para el cálculo de la dimensión fractal debido a su relativamente sencilla implementación y la velocidad a la que realiza los cálculos.

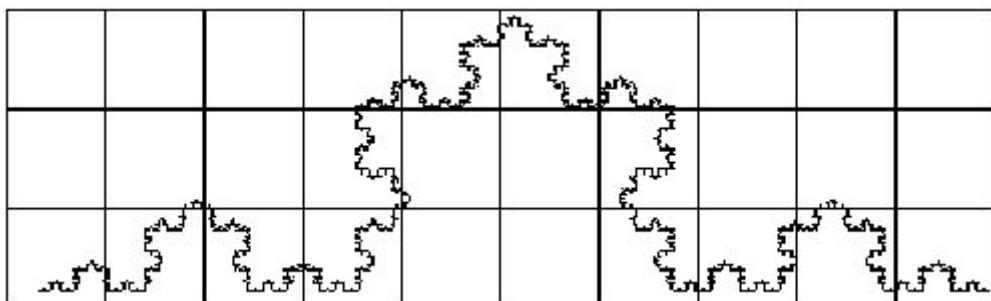
Se va a realizar una explicación del algoritmo en 2 dimensiones para que se pueda entender más fácilmente, y a continuación se describe la implementación realizada en este trabajo para 3 dimensiones.

Como su propio nombre indica, lo que realiza el algoritmo es un conteo de cajas. Es un algoritmo iterativo el cual realiza los siguientes pasos:

1. Calcula el rectángulo que recubre toda la curva o figura
2. Realiza una división del rectángulo en cajas o celdas del mismo tamaño t .
3. Se realiza un conteo de las celdas que contienen parte de la curva
4. Se almacena un punto (x,y) en el que la x se corresponde al tamaño de las cajas en esta iteración y la y al número de cajas que contienen parte de la curva
5. Se realiza un decremento sobre el tamaño de las cajas y se vuelve al punto 2 tantas veces como iteraciones se quieran realizar.
6. Una vez completadas todas las iteraciones y teniendo el resultado de cada iteración, se calculan los puntos $(\log(x), \log(y))$ para cada punto obtenido en cada iteración.
7. Se calcula la recta de regresión mediante el método de mínimos cuadrados $(\log(x), \log(y))$
8. La pendiente de la recta obtenida es una estimación de la dimensión fractal del objeto con signo negativo. Si se realiza la recta de regresión sobre el conjunto de puntos $(\log(1/x), \log(y))$ o $(\log(x), \log(1/y))$ se obtendrá la pendiente con signo positivo por lo que tendremos directamente la dimensión fractal.

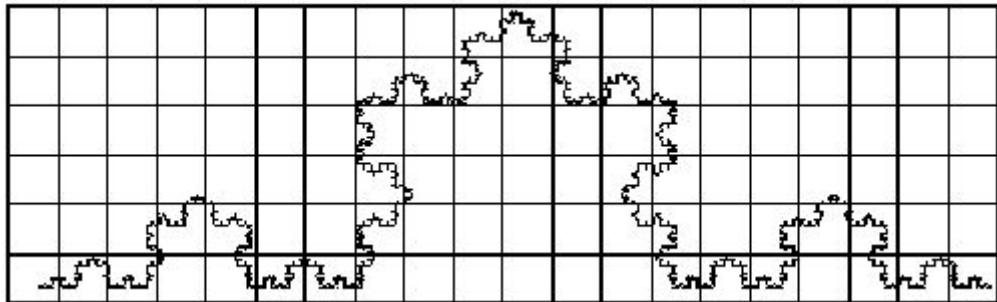
A continuación, se muestra un ejemplo de 3 iteraciones realizado sobre la curva de Koch, otro de los fractales matemáticos más conocidos y utilizados para demostraciones.

Iteración $i = 1$, tamaño $t = 10$



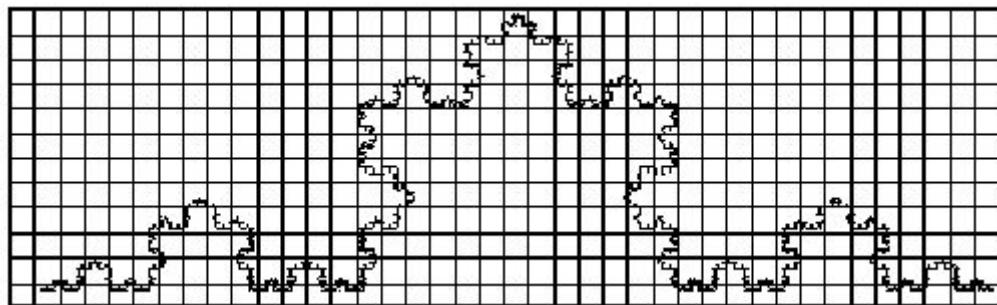
18 de 30 cajas contienen parte de la curva, obtenemos el punto $p1 = (10, 18)$

Iteración $i = 2$, tamaño $t = 5$



41 de 120 cajas contienen parte de la curva, obtenemos el punto $p2 = (5, 41)$

Iteración $i = 3$, tamaño $t = 2.5$



105 de 480 cajas contienen parte de la curva, obtenemos el punto $p3 = (2.5, 105)$

En este punto, si realizamos el cálculo $\log(x)$, $\log(1/y)$ para cada punto obtenemos los siguientes puntos:

$$p1' = (\log(10), \log(\frac{1}{18})) = (1, -1.255)$$

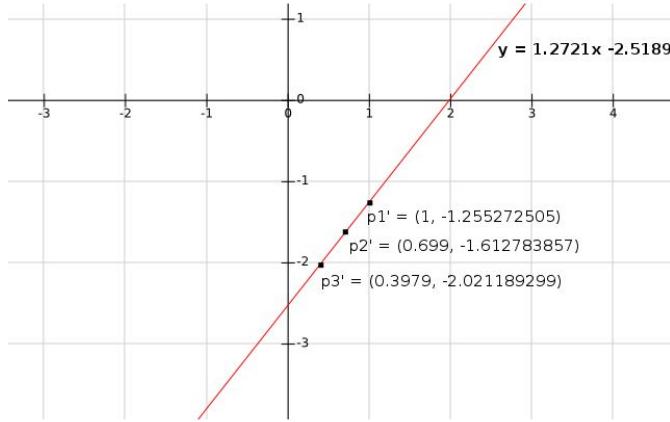
$$p2' = (\log(5), \log(\frac{1}{41})) = (0.699, -1.613)$$

$$p3' = (\log(2.5), \log(\frac{1}{105})) = (0.398, -1. -2.021)$$

Sabemos que la curva de Koch tiene una fractal dimensión teórica es:

$$D = \frac{\log(4)}{\log(3)} = 1.2619$$

Pues bien, si ahora calculamos la recta de regresión sobre $p1'$, $p2'$ y $p3'$ y lo representamos mediante una gráfica obtenemos:



La recta se alinea perfectamente con los 3 puntos con una pendiente, o lo que es lo mismo, la dimensión fractal calculada $D_{bc} = 1.2721 \sim (D = 1.2619)$.

Con este algoritmo tan simple y en tan solo 3 iteraciones obtenemos una aproximación muy ajustada a la dimensión fractal teórica del objeto analizado.

Lo que se va a realizar en este trabajo es una implementación de este algoritmo, llevado a 3 dimensiones, con objetos más complejos y discontinuos, representados como nubes de puntos en un espacio 3D.

Capítulo 8: Desarrollo del algoritmo

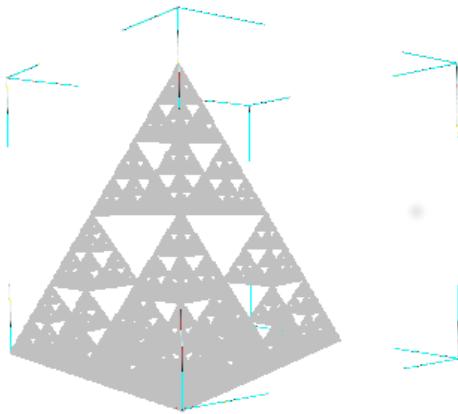
1.- Implementación del algoritmo

Box Counting no tiene ninguna diferencia sustancial en los cálculos al pasar de 2D a 3D, simplemente debemos adaptar las divisiones del espacio por cada iteración, de cuadrados a cubos. Una peculiaridad que a priori parece que sí que puede ser más problemática es la representación de los objetos de forma discontinua.

Se va a describir el algoritmo implementado en una primera fase, y los cambios que se han ido realizando en las siguientes fases buscando mejorar los resultados obtenidos.

En esta fase inicial los pasos realizados para el cálculo de la dimensión fractal son los siguientes:

1. Cálculo del bounding box. Bounding box se traduce como caja delimitadora y esto se traduce en la obtención de dos puntos 3D, $\min(\min_x, \min_y, \min_z)$ y $\max(\max_x, \max_y, \max_z)$:



2. Obtenemos la longitud de la figura en cada eje y guardamos el que tenga mayor longitud $\max(\max_x - \min_x, \max_y - \min_y, \max_z - \min_z)$ y se almacena en la variable “tamaño_max”.
3. Dividimos el valor máximo entre un número de iteraciones especificadas por parámetros. El valor obtenido será el tamaño inicial de los cubos o voxels en los que se fragmentará la figura para realizar los cálculos, y también será el incremento que se realizará en cada iteración sobre el tamaño del voxel. Es importante señalar que las cajas, son de forma cuadrada y no rectangular.
4. Se asigna el valor a las variables “tamaño voxel” e “incremento”.
5. Se realiza la división del bounding box en cubos de tamaño “tamaño voxel”, se realiza el conteo de la cantidad de ellos que contiene algún punto de la nube y se almacena en la variable N.
6. Se almacena en el vector de pares “resultados” el par (o punto 2D) (tamaño voxel, N).
7. Se incrementa tamaño voxel con el valor de la variable incremento.
8. Se vuelve al punto 5 hasta que se hayan completado el número de iteraciones especificado.
9. Calculamos el vector “resultados_log_log” aplicando la fórmula $(\log(1/tamaño_voxel), \log(N))$.
10. Se obtiene la recta de regresión sobre los puntos del vector resultados_log_log.
11. La dimensión fractal calculada es igual a la pendiente de la recta obtenida.

2.- Experimentación y ajuste del algoritmo

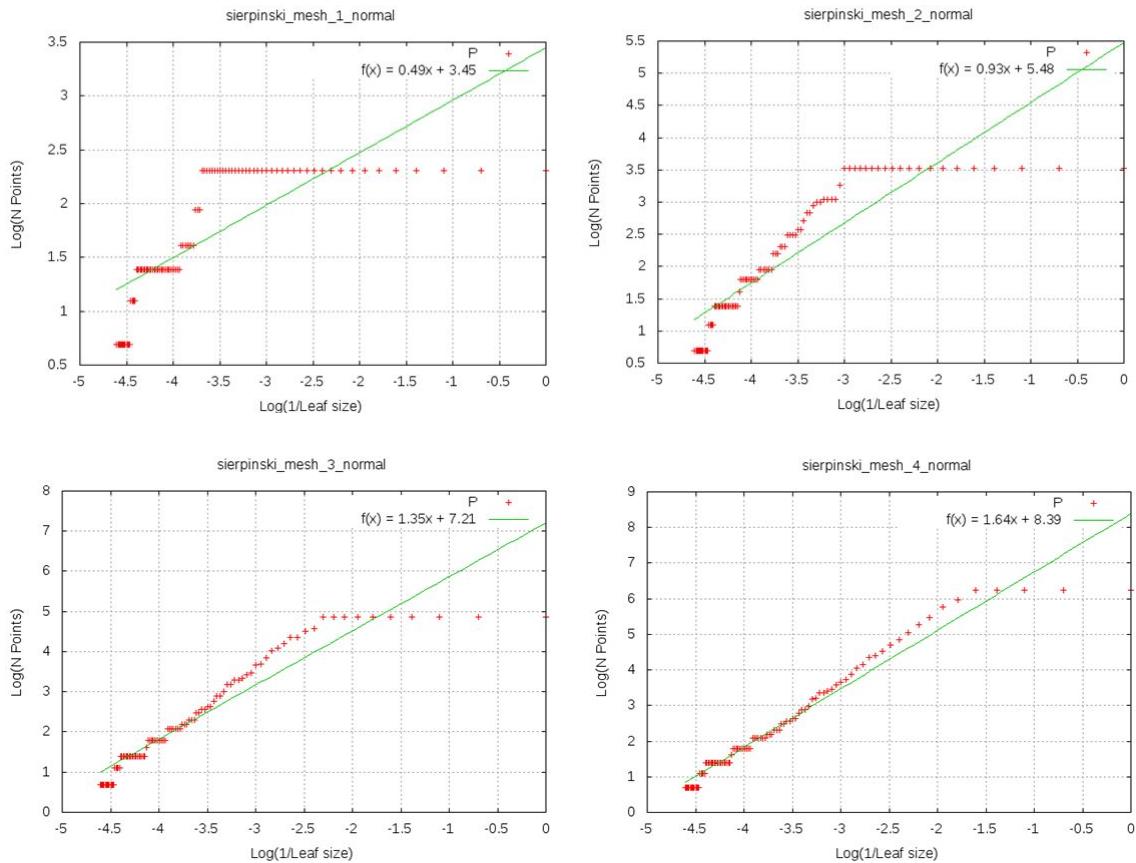
2.1.- Fase 1

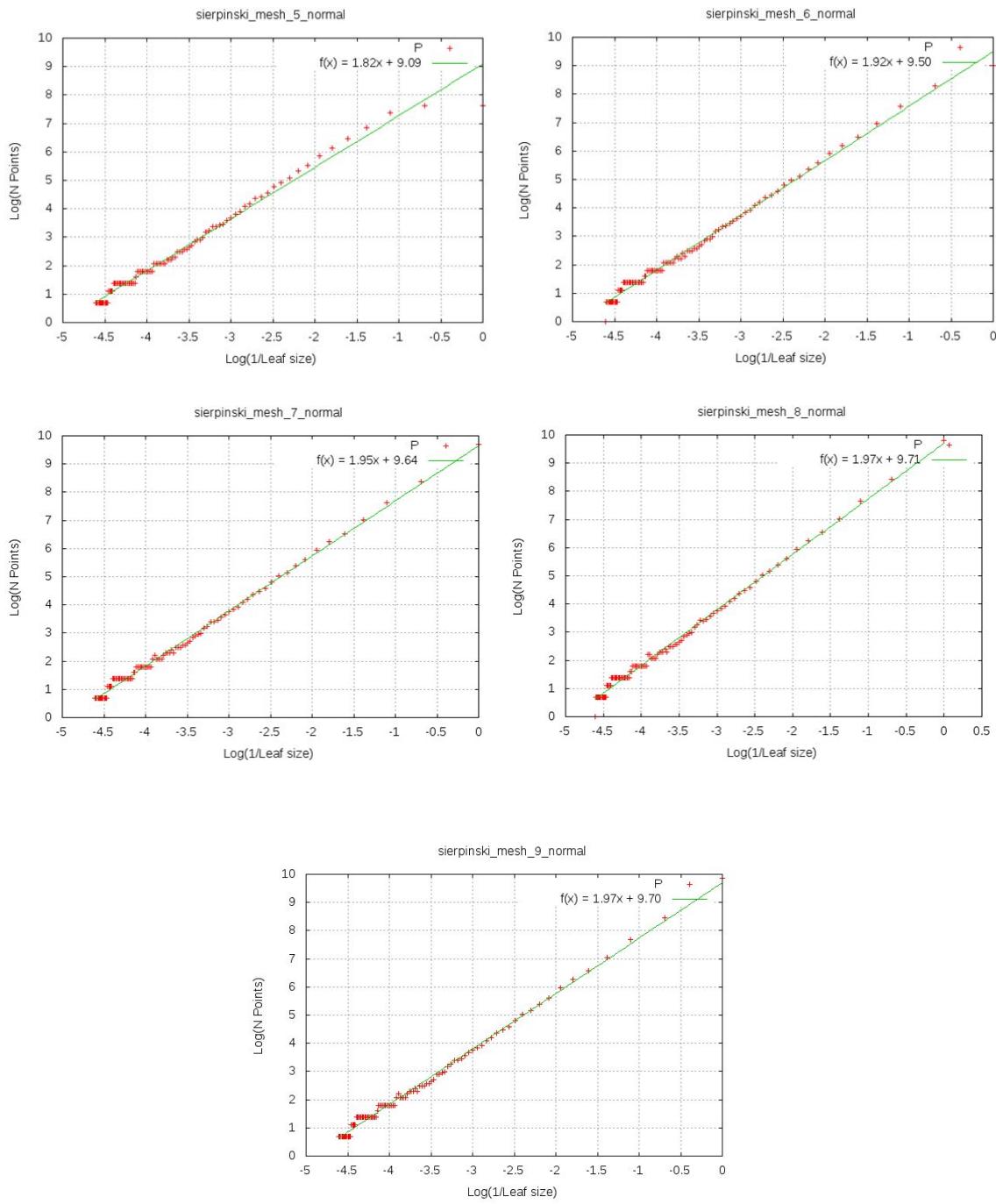
En esta fase se van a realizar pruebas con el algoritmo tal cual se ha descrito anteriormente, realizando los cálculos con 100 iteraciones.

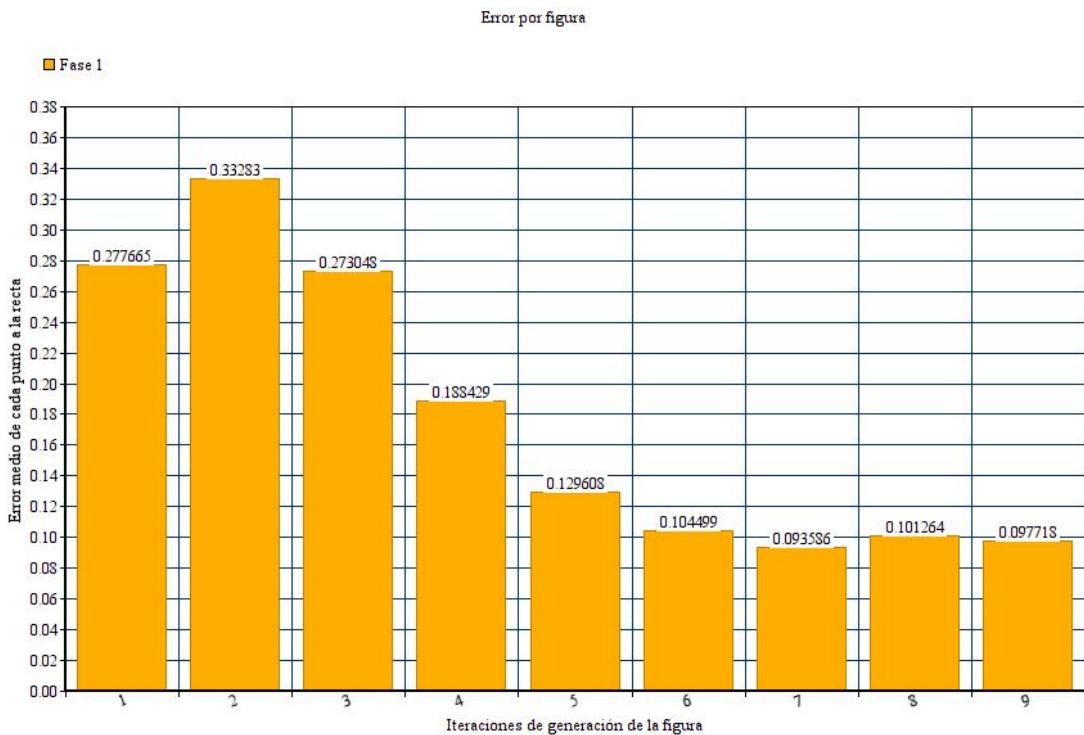
2.1.1.- Resultados

A continuación se muestran las gráficas de los resultados obtenidos con esta versión inicial del algoritmo para el tetraedro de Sierpinski, que recordemos, tiene una dimensión fractal teórica igual a 2. Los resultados mostrados corresponden a la misma figura, pero generada con diferentes iteraciones, de 1 a 9 (el número de iteraciones de la figura correspondiente se indica en el nombre de cada gráfica), con un tamaño de la figura constante para cada iteración.

Para que se entienda bien, recordemos en cómo se generan estas figuras y lo que implica cada iteración. En el caso del tetraedro de Sierpinski, partiendo de un tetraedro, al realizar la primera iteración tendríamos 4 tetraedros formando uno de mayor tamaño. En la segunda iteración se volvería a multiplicar el número de tetraedros por 4, teniendo un total de 16. En la siguiente 64, y así sucesivamente. Los tetraedros están representados por sus vértices, en forma de nube de puntos:







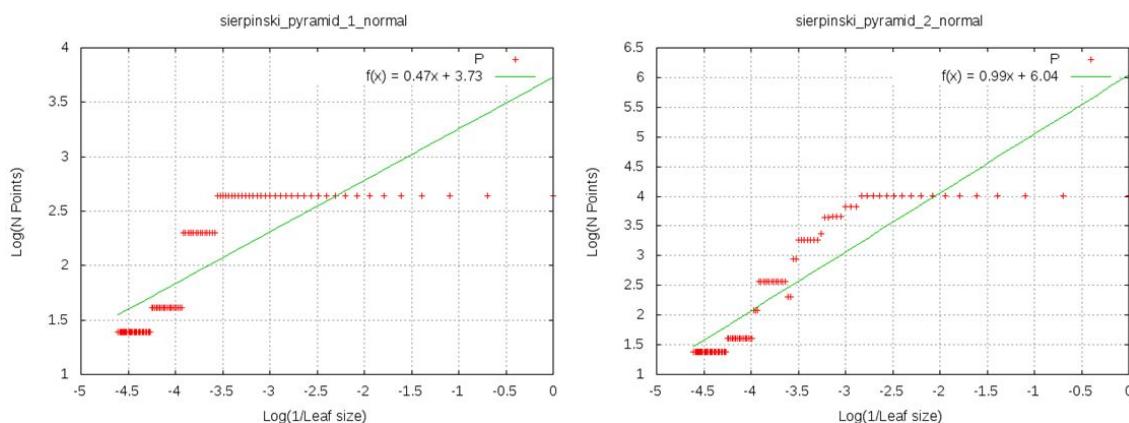
2.1.2.- Conclusiones

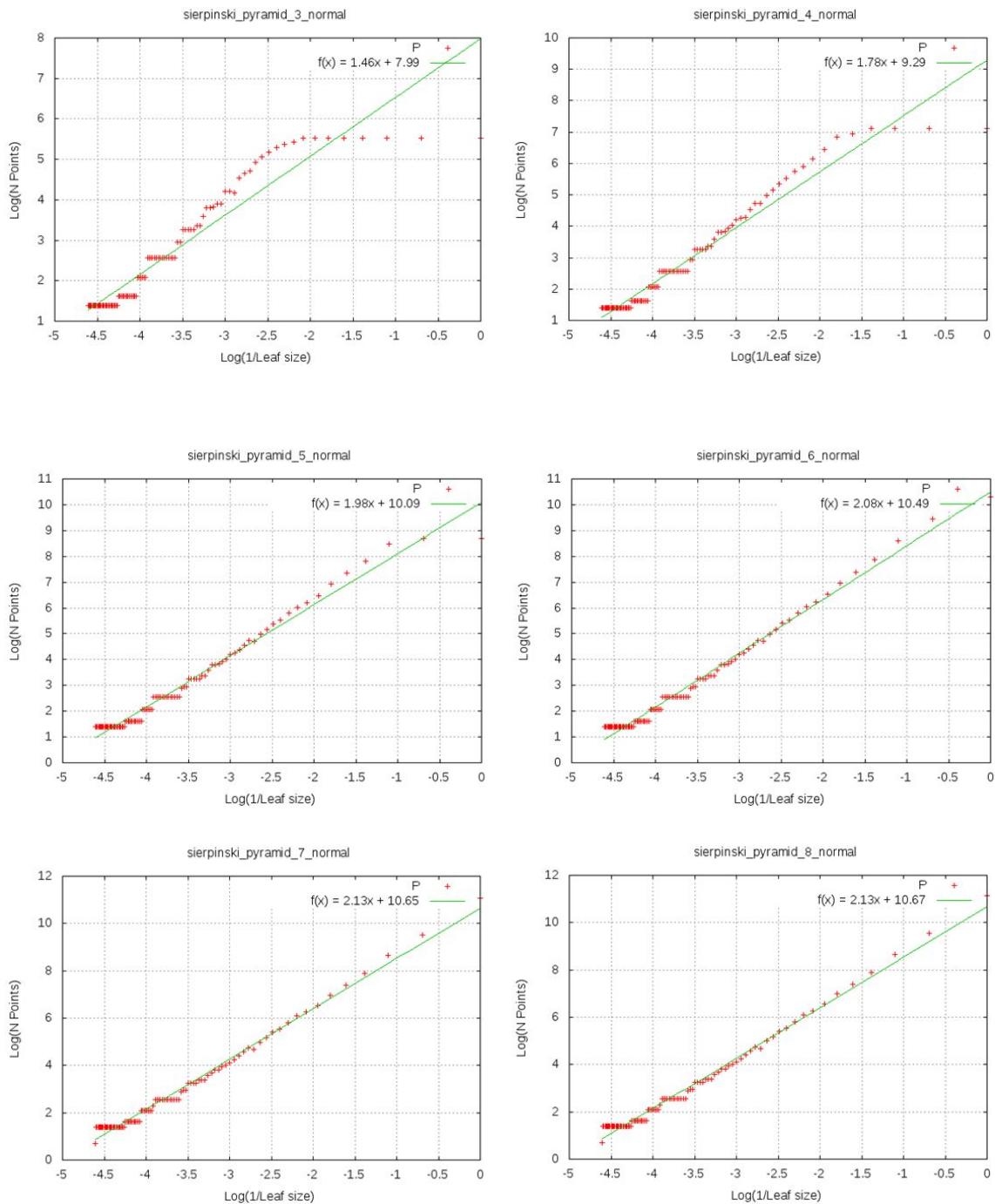
Para una fácil lectura, a continuación se enumeran las conclusiones obtenidas de esta primera experimentación:

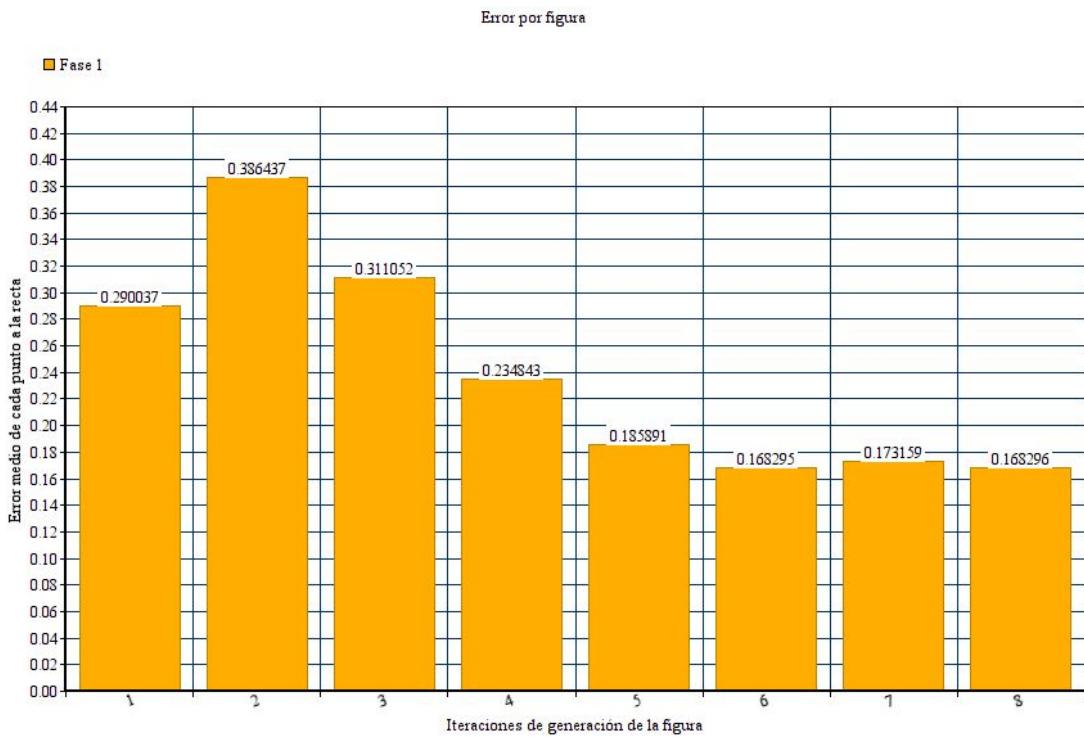
- Las figuras generadas con menos iteraciones (1, 2 y 3) contienen menos puntos por lo que es más difícil obtener datos significativos realizando el mismo número de iteraciones que para las figuras con una mayor cantidad de puntos. Además, contienen menos repeticiones del patrón de la estructura por lo que es más difícil de calcular. Teóricamente, con estas figuras, realizando una generación a partir de la 5-6 iteración se alcanza el punto crítico en el que ya no se puede apreciar diferencias a medida que se realizan más iteraciones. Por estos motivos, en siguientes experimentaciones no se va a dar ninguna atención a resultados para figuras generadas con menos de 4 iteraciones.
- Esto también abre la cuestión de si tiene sentido especificar el número de iteraciones por parte del “usuario” en lugar de que el algoritmo sea capaz de decidir las iteraciones que va a realizar en función de la figura, pero de momento este punto se va a dejar para más adelante.
- Conforme se van obteniendo los resultados para las figuras con mayores iteraciones, se observa que los valores de la dimensión fractal empiezan a converger, aproximándose

cada vez más entre el resultado de una figura a otra. También se va estabilizar el error cometido en la regresión lineal.

- Con esta versión del algoritmo, se obtienen resultados más aproximados al resultado teórico ($D = 2$) a medida que más veces existe el patrón estructural en la figura, además de que el error sobre la recta de regresión va disminuyendo.
- Los puntos (x,y) no se alinean de una manera muy precisa con la recta de regresión calculada. Se aprecia que muchos de los puntos sí que siguen una tendencia pero hay muestras que crean ruido en el cálculo de la regresión lineal, y por tanto no se obtiene la pendiente que marcaría la dimensión fractal correcta.
- El algoritmo parece válido ya que los resultados obtenidos con las figuras más iteradas para la dimensión fractal se aproximan bastante bien a la dimensión teórica, pero se debe ajustar mejor ya que las gráficas muestran que existe demasiado error en las muestras a la hora de calcular la regresión lineal.
- La variación del tamaño del voxel en cada iteración del algoritmo parece afectar, sobre todo, en algunos intervalos de la gráfica. Esto tiene una relación con la distribución de los puntos de la nube.
- Para unas figuras, los tamaños de voxel iniciales (más a la derecha en la gráfica) provocan ruido y para otras ocurre para los tamaños finales de voxel (más a la izquierda en la gráfica). De aquí sale la siguiente conclusión.
- El tamaño de voxel inicial y el final afectan de una manera u otra en función de la figura que se está analizando.
- Se muestran a continuación los resultados para la pirámide de Sierpinski para ver si las conclusiones extraídas del análisis del tetraedro también son válidas el caso de la pirámide, cuya dimensión fractal, recordemos, es igual a 2.3219:

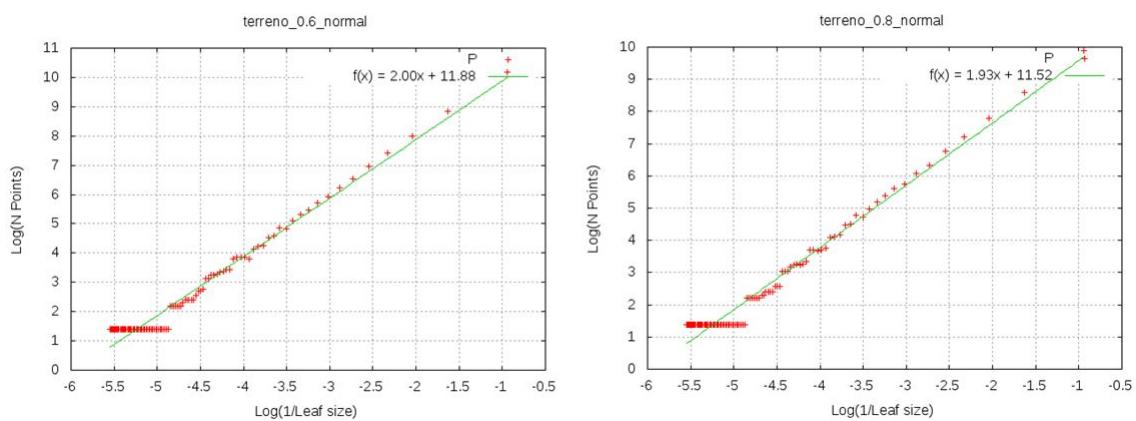


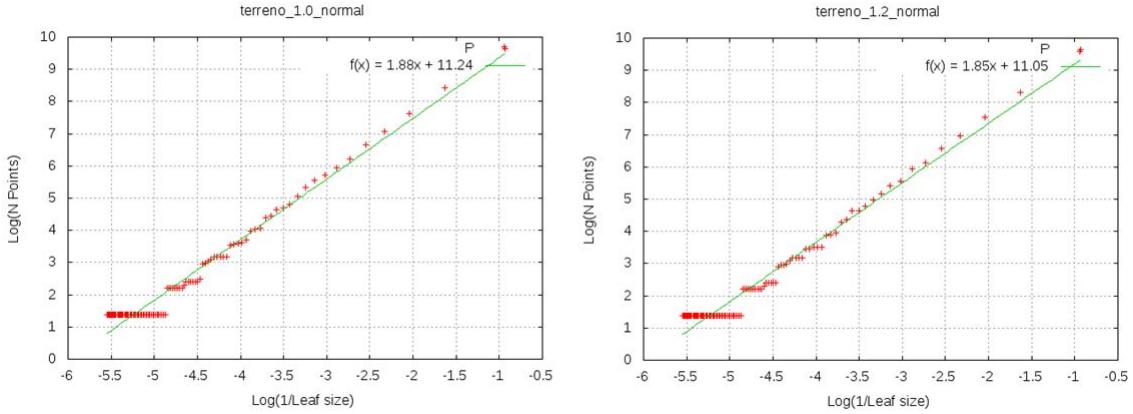




Efectivamente, tanto las gráficas de resultados como la del error son muy similares para el tetraedro y la pirámide de Sierpinski, con la diferencia de que en las gráficas de la pirámide podemos ver un aumento del ruido en las muestras de los tamaños de voxel finales (más cercanos al tamaño de la figura original). Eso refuerza alguna de las conclusiones sobre el tamaño del voxel, su variación en cada iteración y el número de iteraciones a realizar deben estar en función de la figura que se está analizando en cada momento.

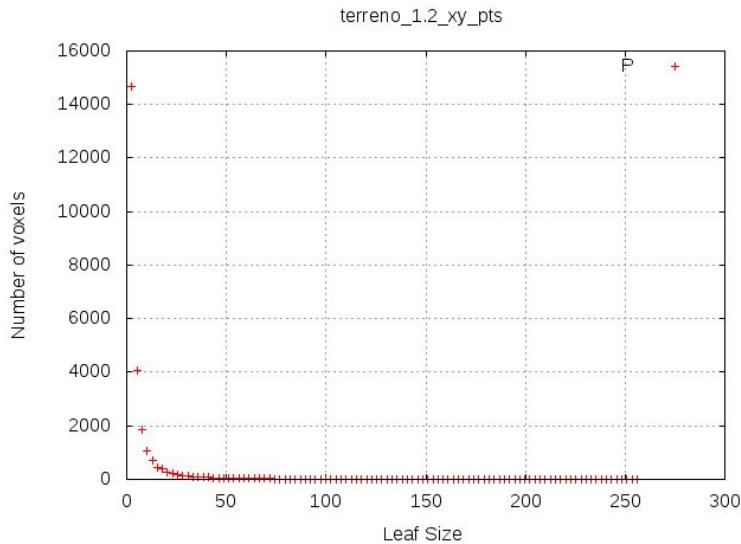
Veamos ahora qué resultados se producen al procesar algunos de los terrenos fractales (ordenados de más rugoso a más suave):





En este caso vemos más claramente como una gran parte de las iteraciones realizadas por el algoritmo (las de mayor tamaño de voxel) son contraproducentes ya que no aportan información y afectan negativamente al cálculo de la recta de regresión.

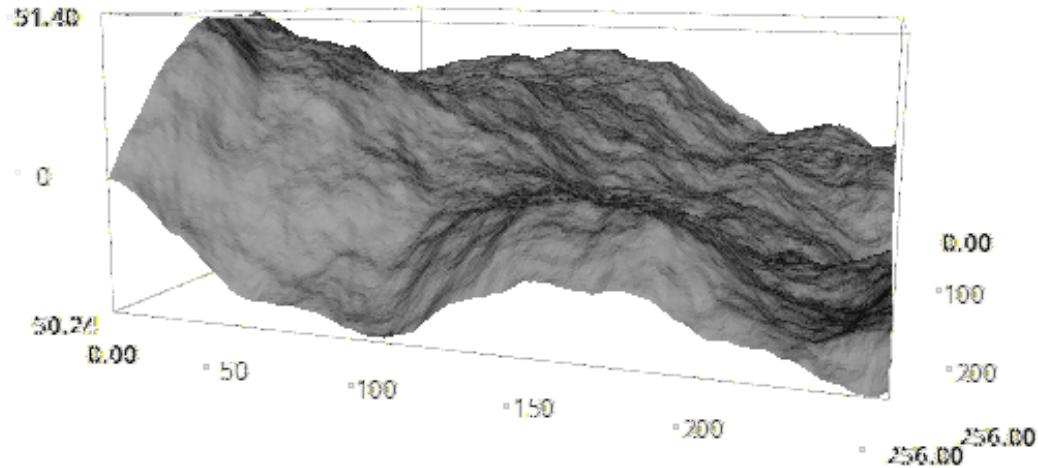
Es fácil darse cuenta que cuando el tamaño de los voxels es mayor a la variable calculada *tamaño_max/2*, no se puede captar prácticamente información ya que habrá muy poca variación en los resultados de las siguientes iteraciones. Una gráfica que puede ayudarnos a entender esto mejor es la siguiente:



Aquí vemos en el eje y el número de voxels que contienen algún punto de la nube y en el eje x el tamaño del voxel. Se puede ver que en las primeras iteraciones se está perdiendo la mayor parte de los puntos que forman el objeto, en este caso el terreno, por lo tanto en el resto de las iteraciones no obtenemos casi información ya que lo que buscamos es captar la variabilidad de los resultados en las diferentes iteraciones, y cuando no hay variabilidad, se está introduciendo error.

También existe un problema con objetos de este tipo ya que los puntos que forman el terreno se distribuyen mayormente a lo largo de dos de los tres ejes de coordenadas, como vemos en la

siguiente imagen, el tamaño en los ejes x y z (256) es cinco veces mayor que el tamaño en el eje y (51.4):



Esto implica que cuando se están usando véxeles de tamaño mayor a 51.4 solo se capta información en los ejes x y z.

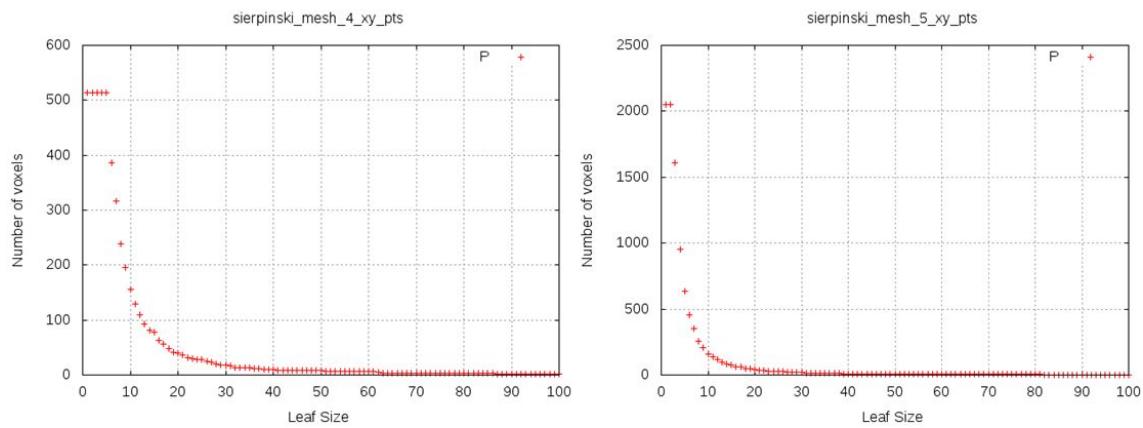
2.2.- Fase 2

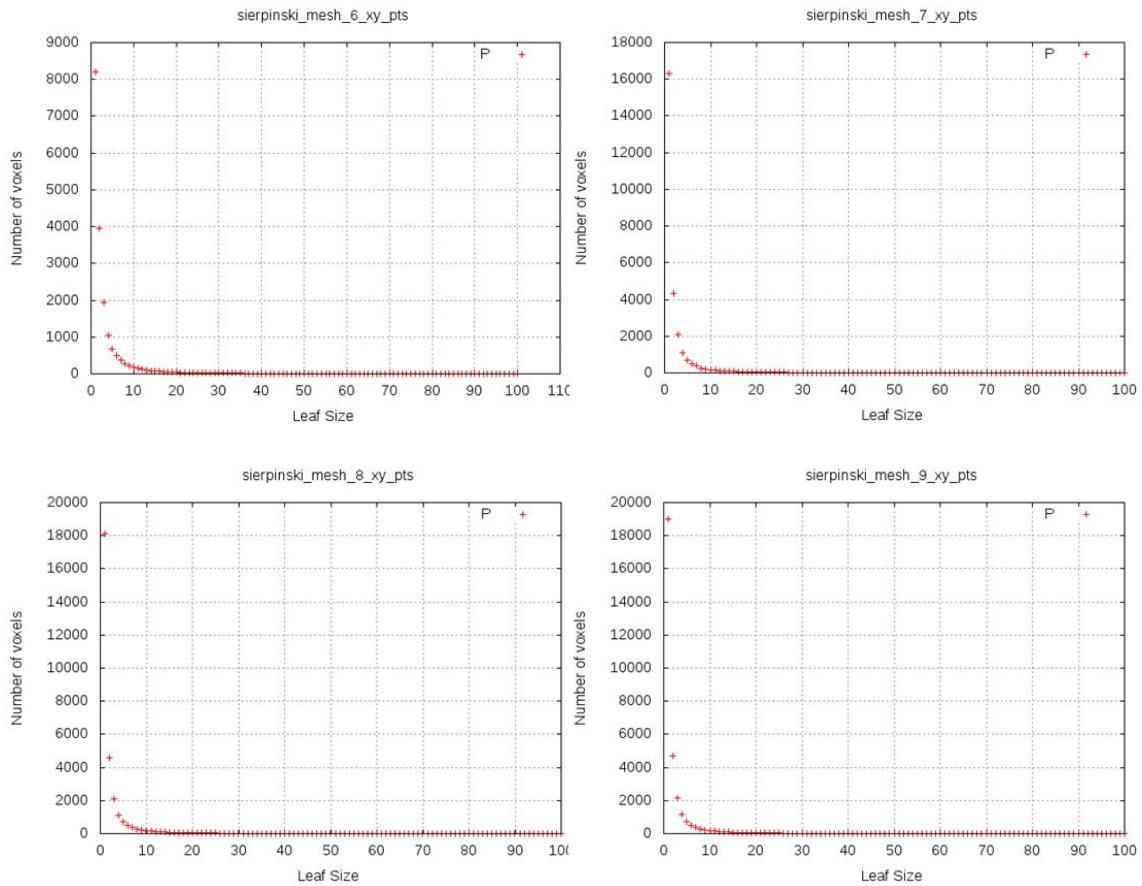
2.2.1.- Problemas a resolver

En esta fase se va a intentar dar una solución para obtener mayor variabilidad de los resultados en las iteraciones con tamaño de véxeles mayores.

Vamos a comprobar también con otras figuras si tenemos el mismo problema de la pérdida de información al realizar las primeras iteraciones.

Tetraedro de Sierpinski:





Debido a que las diferentes nubes de puntos que representan el tetraedro de Sierpinski que estamos analizando tienen todas el mismo tamaño, pero cada una tiene más detalle que el anterior, lo que nos está diciendo este grupo de gráficas es que si utilizamos solo el tamaño del objeto para decidir el tamaño de los véxeles para cada iteración, estaremos perdiendo mucha información en las figuras que tienen más detalle, y es precisamente lo contrario que queremos, queremos captar el patrón con el máximo detalle posible.

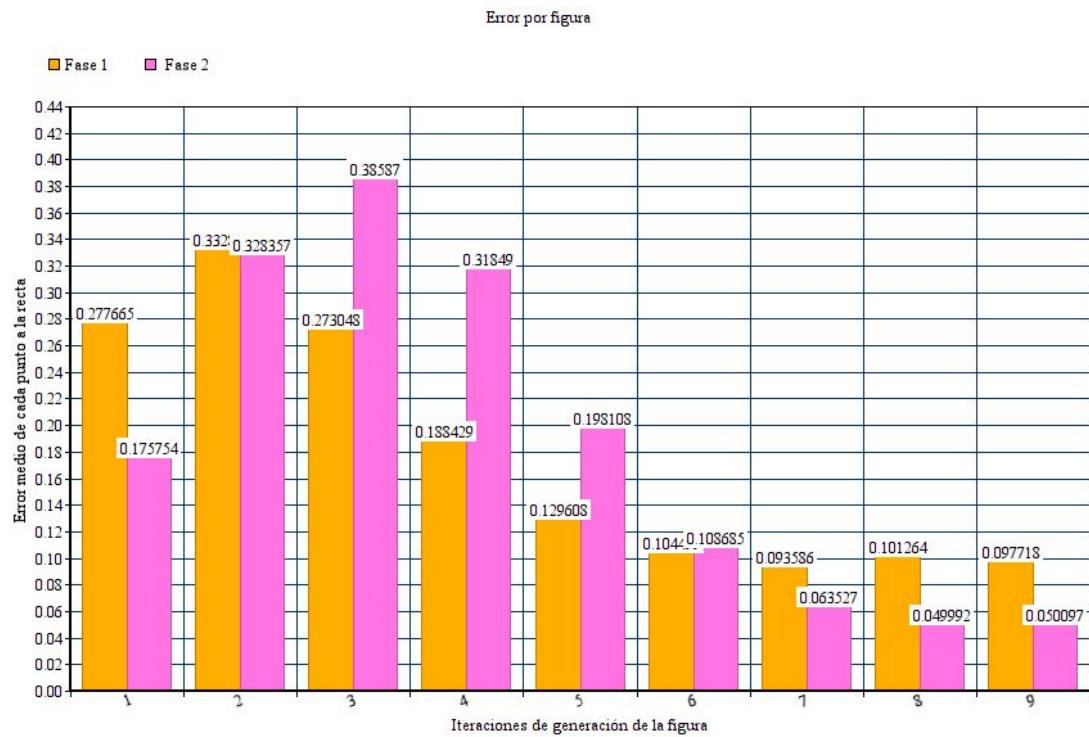
También evidencian que iterar hasta el tamaño máximo de la figura no aporta más información si no que es contraproducente.

2.2.2.- Modificaciones

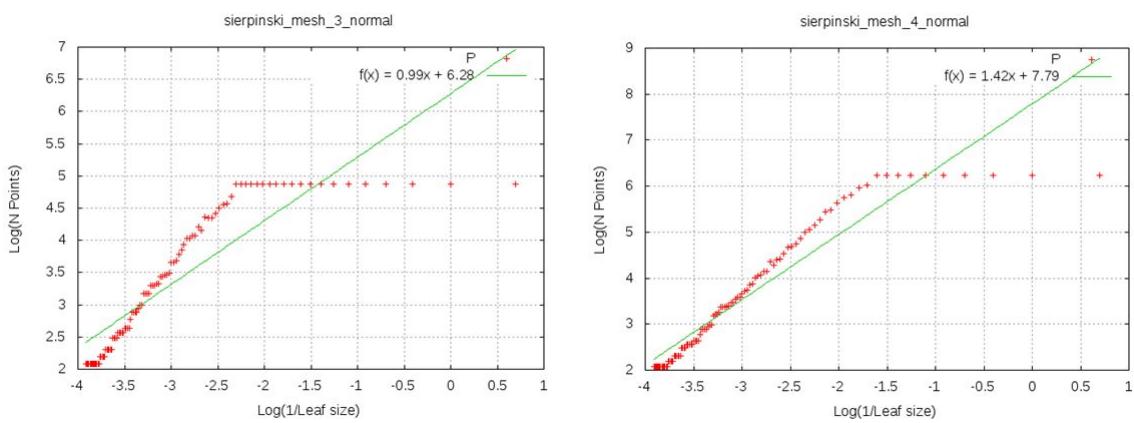
Esta segunda fase se va a restringir el tamaño máximo de los véxeles, es decir, el tamaño en la última iteración, a la mitad del tamaño máximo de la figura y ver si obtenemos mayor variabilidad en las diferentes iteraciones.

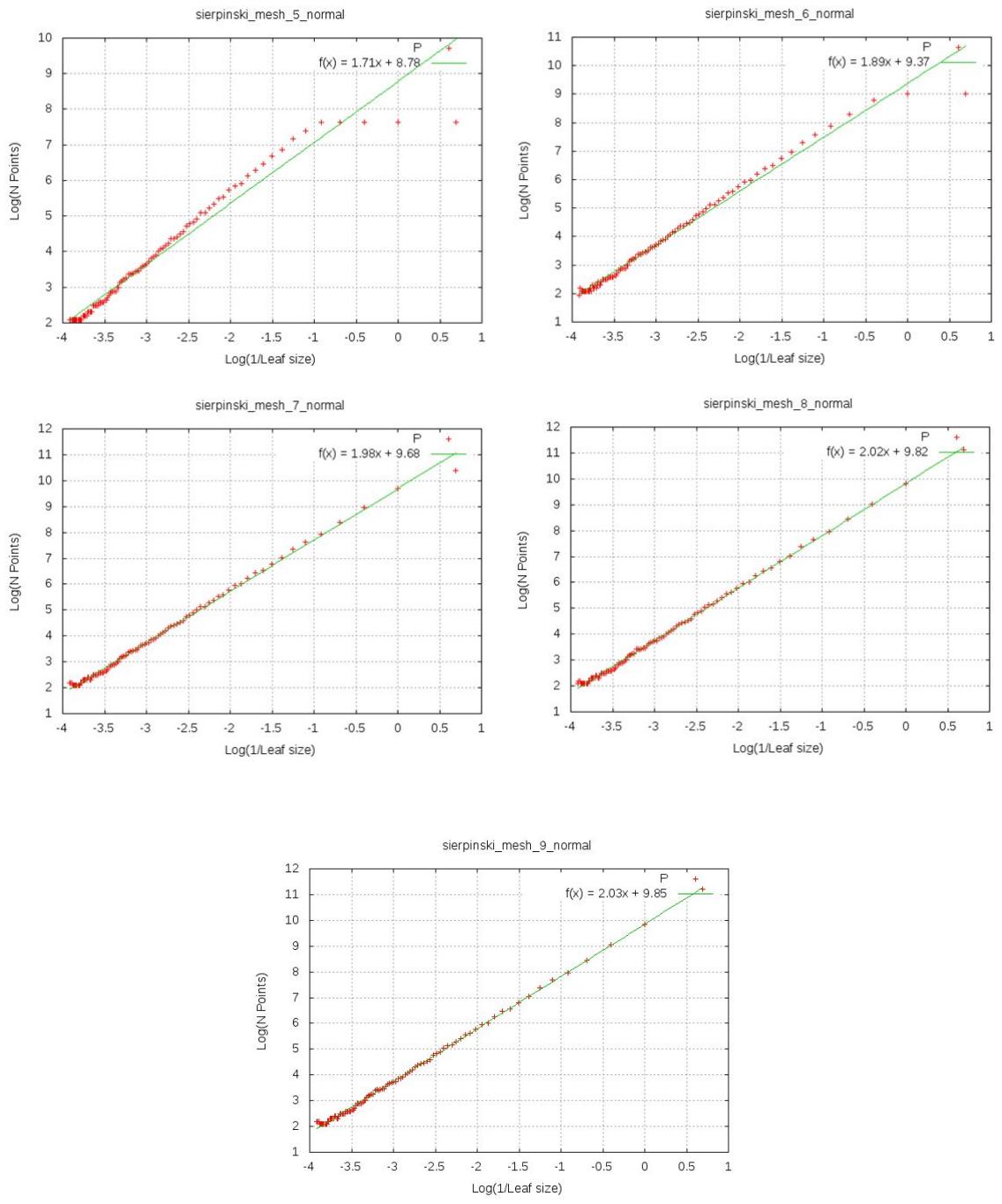
2.2.3.- Resultados

Empezamos con el tetraedro de Sierpinski, haciendo una comparativa del error obtenido en la regresión lineal entre la fase 1 y la fase 2:

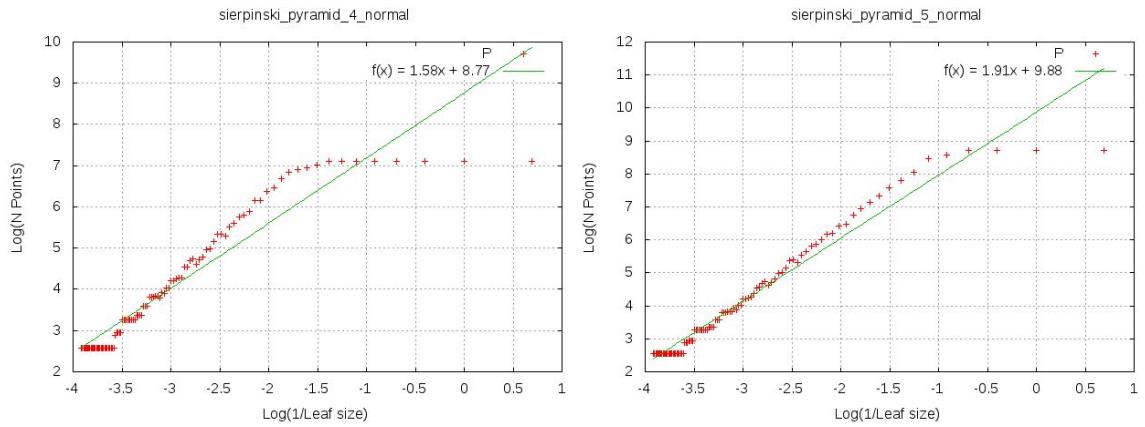


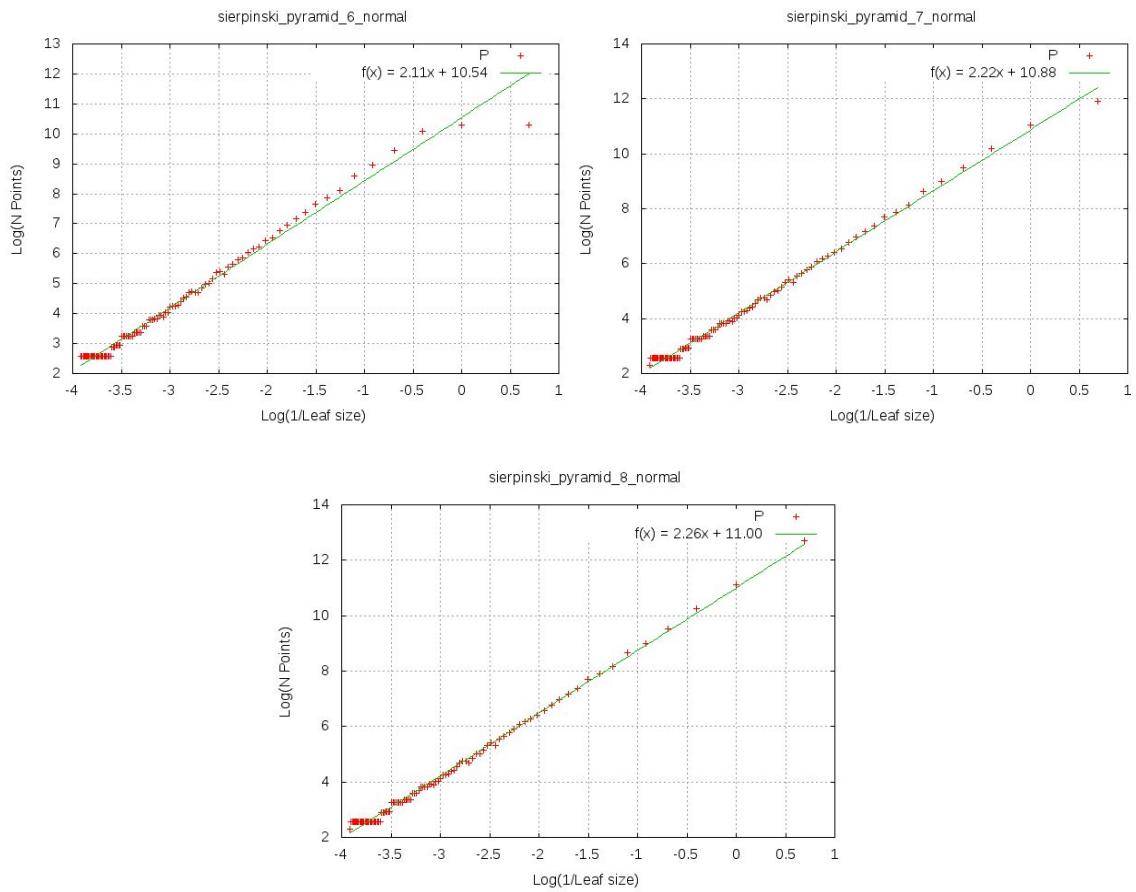
Se puede observar como el error ha aumentado de forma considerable con respecto a la fase 1 en las figuras 3, 4 y 5. En la figura 6 se obtienen errores similares, y después para las figuras 7, 8 y 9 con mayor nivel de detalle empieza a verse una mejora. Veamos las gráficas de regresión lineal de las figuras mencionadas:



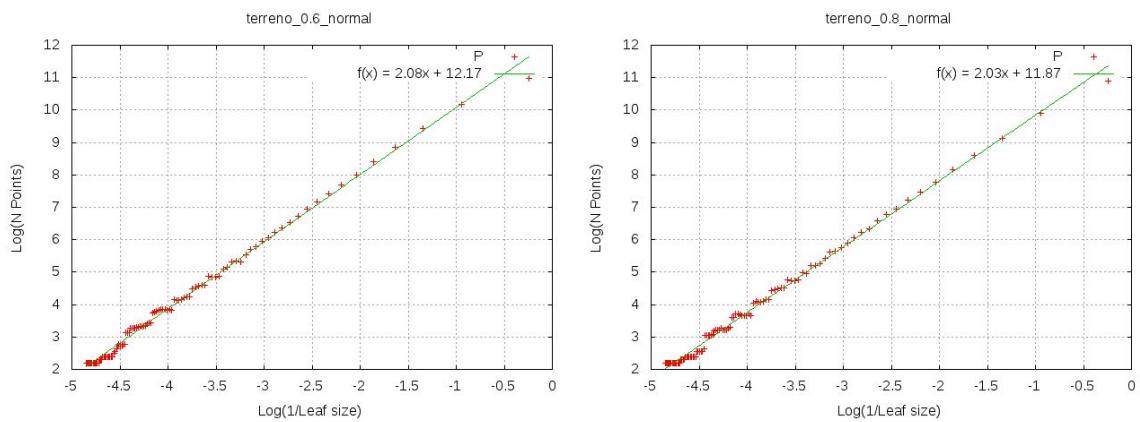


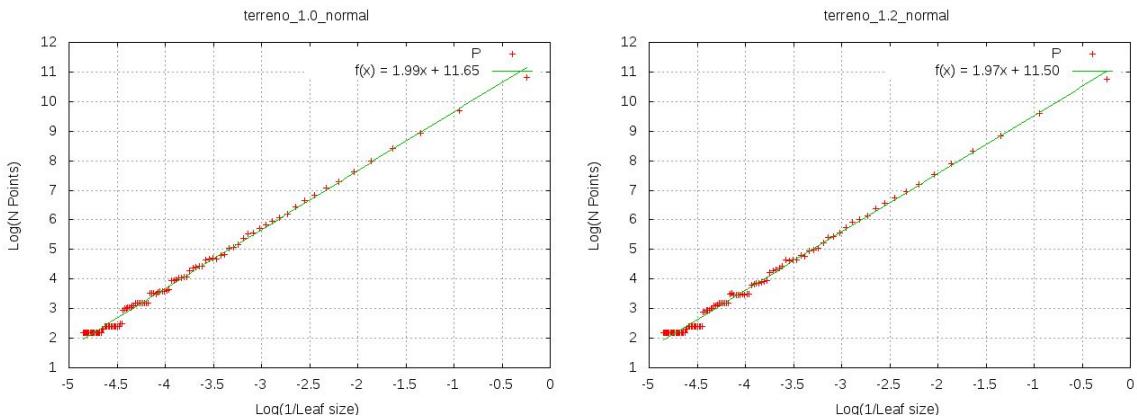
Resultados para la pirámide de Sierpinski:





Resultados para los terrenos:





2.2.4.- Conclusiones

A continuación se enumeran las conclusiones obtenidas de esta experimentación:

- Al limitar el tamaño máximo de vóxel a $tamaño_max/2$ podemos ver que efectivamente los resultados de las últimas iteraciones (más a la izquierda en la gráfica) muestran mayor variabilidad que en la fase 1.
- En el caso de los terrenos, se ha eliminado por completo un grupo de puntos que se alineaban de forma horizontal en los tamaños más grandes de vóxel en el caso de la fase 1, y se aprecia una mejor alineación de los puntos sobre la recta de regresión calculada.
- Para el caso de la pirámide, vemos que ha mejorado pero sigue habiendo un grupo de puntos alineados horizontalmente en la parte izquierda de las gráficas, y los resultados siguen sin aproximarse al resultado teórico, seguramente en gran parte debido a esto.
- Al mismo tiempo que estamos obteniendo mejores resultados en las iteraciones con tamaños más grandes, estamos introduciendo error en las que tienen tamaños más pequeños en el caso de figuras con menos densidad o detalle del tetraedro. Esto se debe a que al calcular el tamaño inicial de vóxel y el incremento en cada iteración en función al tamaño máximo, en esta fase dos tendríamos tamaños iniciales e incrementos más pequeños para una misma figura, por lo que parece que se están haciendo las divisiones en vóxeles a un tamaño más pequeño que la resolución del objeto, por lo que no se obtiene variación en los resultados de las primeras iteraciones.
- Aunque los resultados obtenidos sigan sin ser ideales, parece una buena decisión el limitar el tamaño máximo de vóxeles, aunque será necesario tomar otras decisiones para seguir ajustando el algoritmo.
- Parece también que sería buena idea ajustar los tamaños más pequeños de vóxel en función de la densidad de la nube de puntos que se está procesando.

2.3.- Fase 3

2.3.1.- Problemas a resolver

El objetivo de esta fase es similar a la fase dos, pero en este caso se pretende ajustar el tamaño inicial de los vóxeles a un tamaño mínimo, que estará en función de la densidad del objeto, y así solventar el error que se está introduciendo en los resultados al realizar iteraciones del algoritmo con tamaños demasiado pequeños.

Se va a mantener la modificación de la fase 2 ya que, aunque hay que seguir ajustando, ofrece mejoras frente a la fase 1 y parece de sentido común.

2.3.2.- Modificaciones

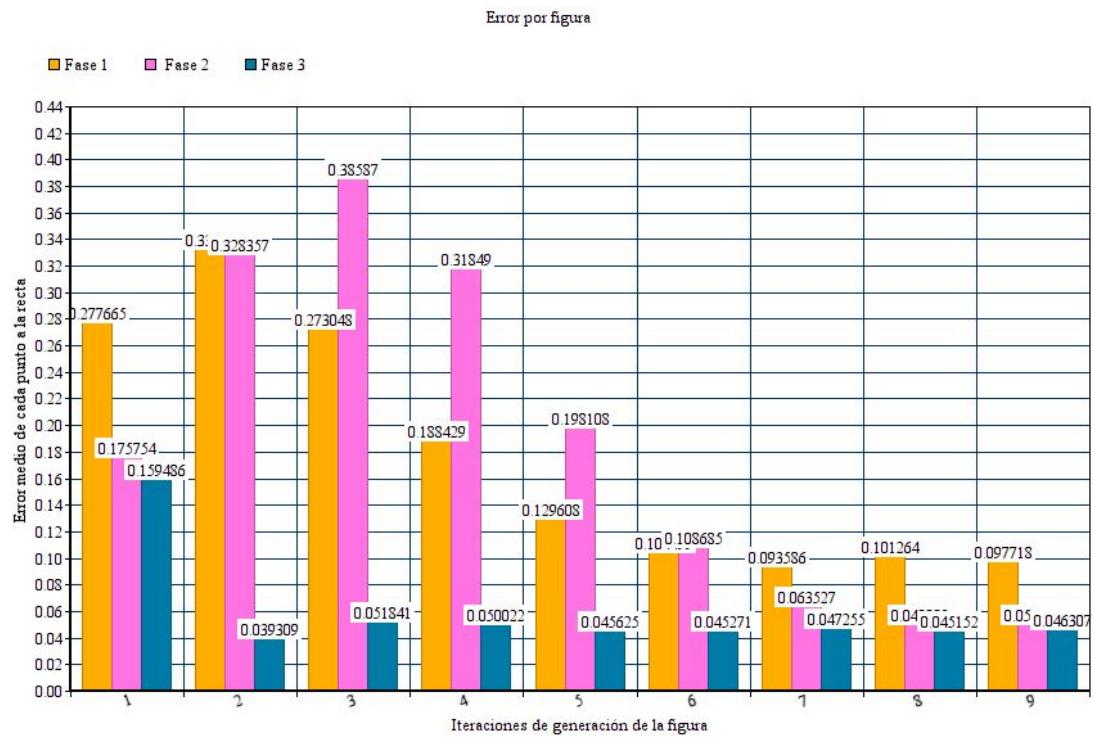
En el caso de las nubes de puntos 3D, y en lo que afecta al algoritmo que se está implementando, podríamos utilizar como medida de resolución máxima del objeto la distancia entre los puntos que componen su superficie.

Una forma de no utilizar tamaños demasiado pequeños es restringir el tamaño inicial del algoritmo en función de la media de distancias entre vecinos más cercano, esto es, el sumatorio de la distancia de cada punto a su vecino más cercano y dividido entre el número de puntos que contiene la nube. Como tampoco tiene sentido llegar a un tamaño tan pequeño, se va a usar el doble de este valor como tamaño inicial en esta fase, y el incremento por cada iteración será calculado como:

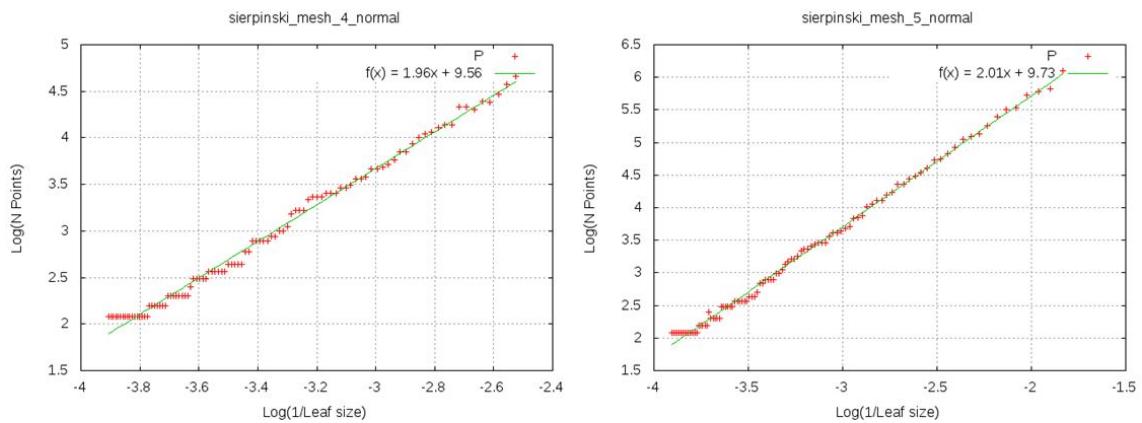
$$(tamaño_final - tamaño_inicial) / iteraciones$$

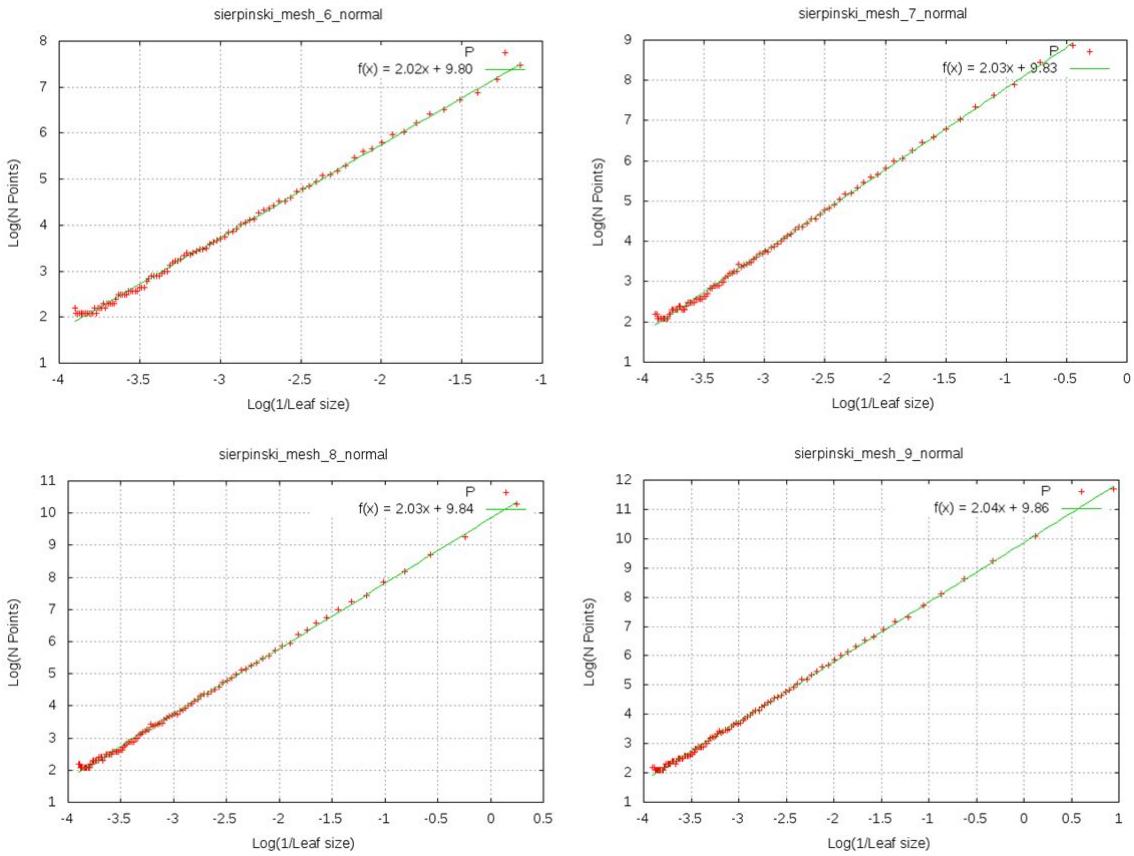
2.3.3.- Resultados

Empezamos igual que en la fase 2 haciendo una comparativa del error en la regresión lineal en las distintas fases de experimentación para el tetraedro de Sierpinski:

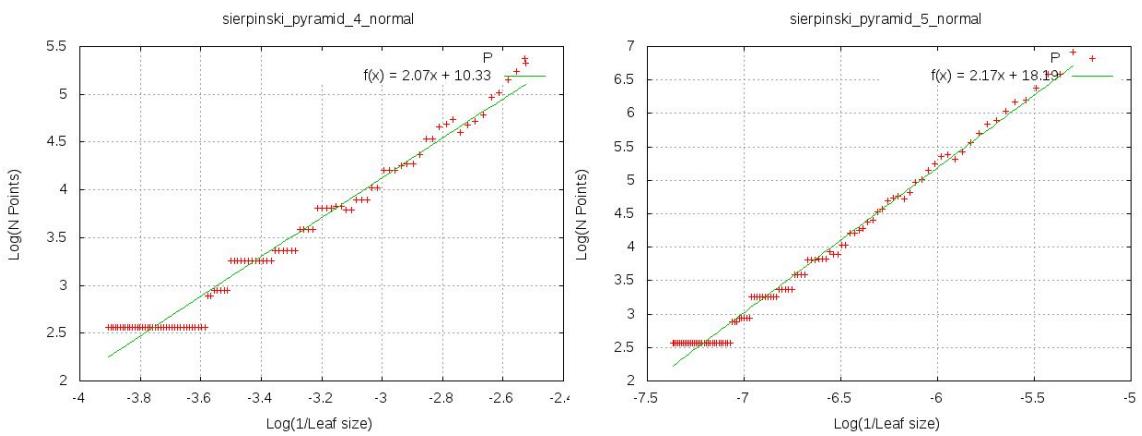


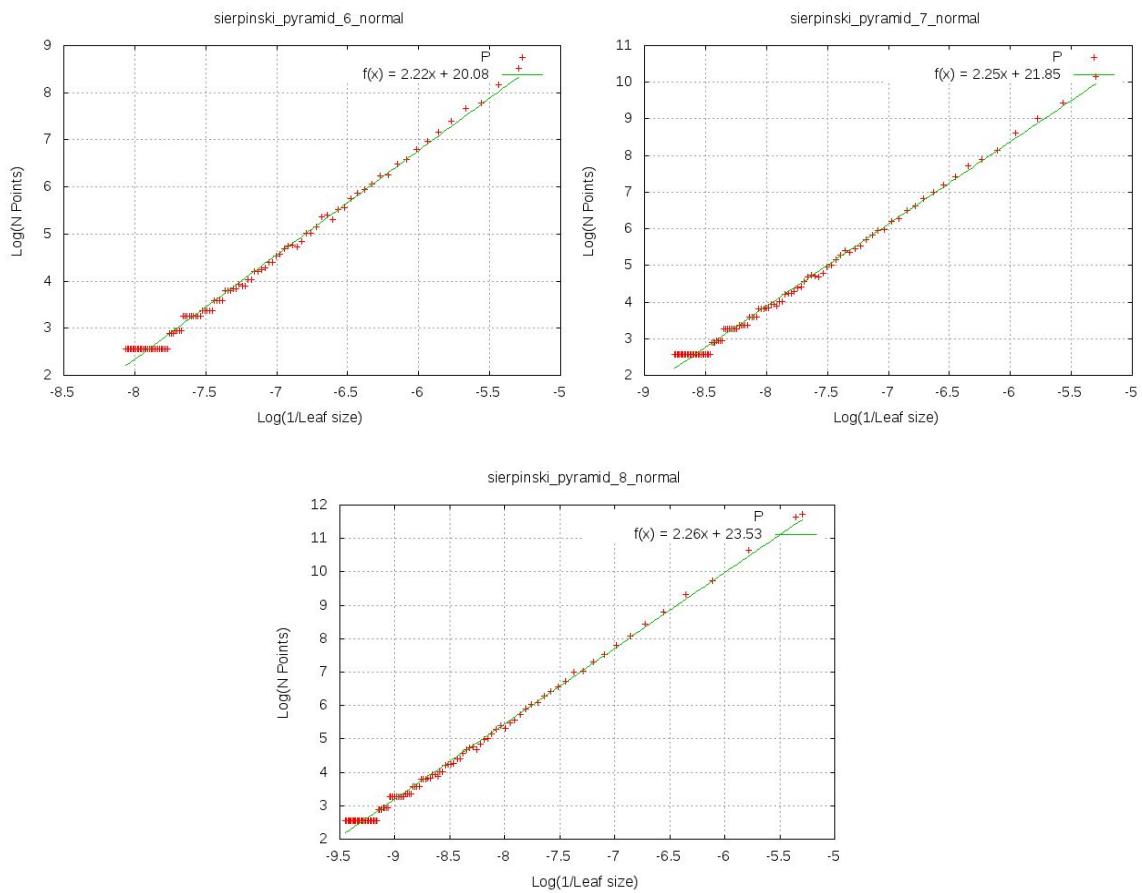
En esta gráfica podemos ver que el cambio realizado mejora muy considerablemente los resultados de la regresión lineal en cuanto al error obtenido. Veamos cómo son las gráficas generadas con los resultados de la fase 3 para el tetraedro de Sierpinski:



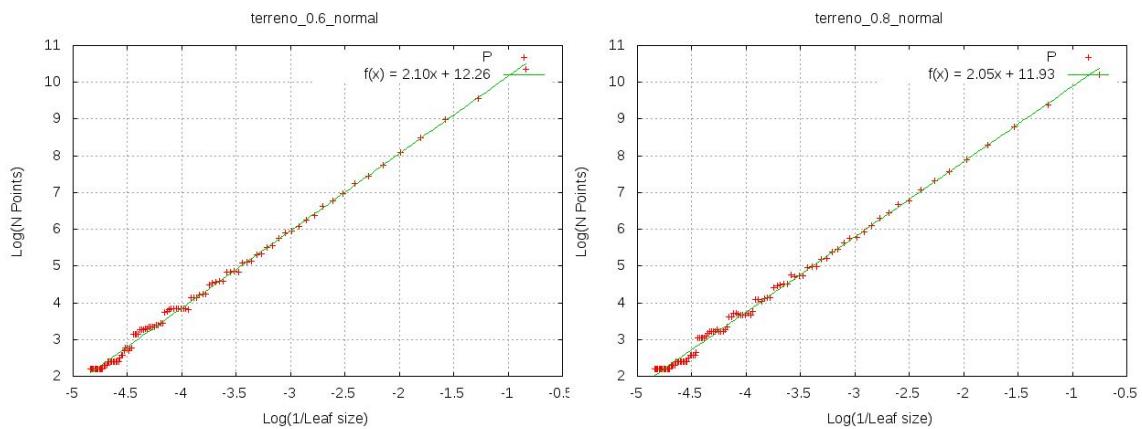


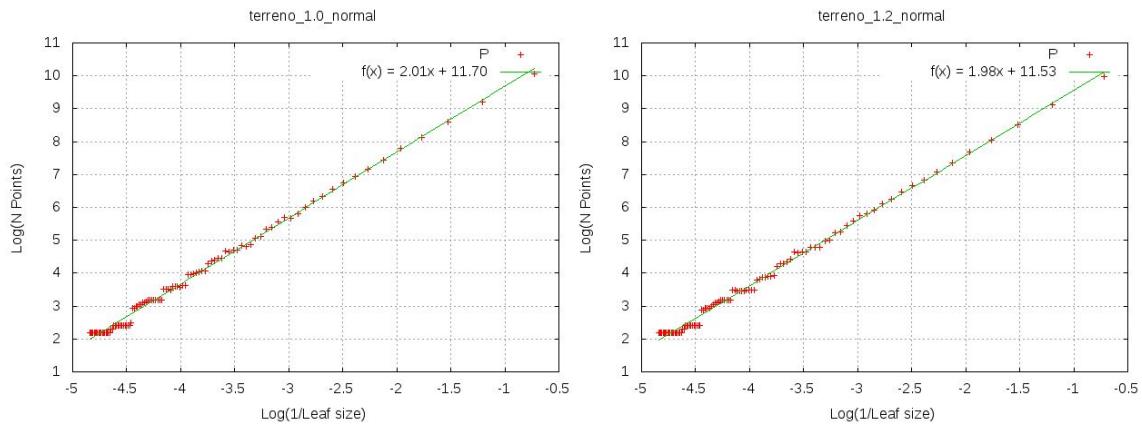
Resultados para la pirámide de Sierpinski:





Se muestran también resultados obtenidos sobre los terrenos fractales:





2.3.4.- Conclusiones

A continuación se enumeran las conclusiones obtenidas de esta fase:

- En cuanto al error producido en la regresión lineal con los resultados del algoritmo, tenemos una mejora en todos los casos del tetraedro de Sierpinski, en especial para el caso de los que contienen menos repeticiones del patrón geométrico que define la figura, exactamente como se pretendía hacer.
- Si hacemos una comparativa de las gráficas de regresión lineal en la fase 2 y la fase 3 para una misma figura, se puede perfectamente apreciar porque se está reduciendo tanto el error, y es que en la fase dos los puntos correspondientes a las primeras iteraciones están provocando una inclinación muy fuerte sobre la recta de regresión. Cuando no tenemos esos puntos, la recta puede ajustarse mucho mejor al conjunto de puntos de forma global y de esta forma reducir tanto el error.
- Podemos ver que esa mejora se ve también reflejada en una mejor aproximación de la dimensión fractal calculada respecto a la dimensión fractal teórica de nuestro objeto, y por lo tanto, gracias a este cambio el algoritmo da mejores resultados y está bastante más ajustado que en fases anteriores.
- El partir de un tamaño de voxel inicial mayor hace que el incremento en cada iteración sea diferente a las anteriores fases, lo que también está afectando a los resultados
- En el caso de los terrenos, no se ven apenas diferencias ya que estos no presentaban el problema que se quería solventar en esta fase.
- El algoritmo puede mejorar o empeorar mucho en función de cómo se elijan los tamaños de voxel inicial y final. Hemos visto que estos parámetros tienen mucho peso sobre los resultados que se producirán, por lo que no es descabellado pensar que el incremento realizado en cada iteración también pueda ser determinante.

- Aunque con mejoras en los resultados, seguimos viendo que hay parte de los puntos que no se acaban de alinear de forma suave sino que tienen cierta forma escalonada. Esto son iteraciones que nos están introduciendo error.
- En este punto se pueden barajar distintas opciones. Pensar una manera de incrementar el tamaño de voxel en cada iteración distinta a como se hace actualmente, o intentar utilizar un método de eliminación de ruido o métodos de suavizado que nos proporcionen un mejor conjunto de puntos sobre el que calcular la recta de regresión.

2.4.- Fase 4

2.4.1.- Problemas a resolver

En esta fase vamos a realizar una experimentación variando el número de iteraciones realizadas por el Box Counting y viendo cómo afecta a los resultados.

La idea es probar con la intención de conseguir una mejor alineación de los puntos resultantes en cada iteración, y por tanto, menos error y un mayor acierto en el cálculo de la dimensión fractal.

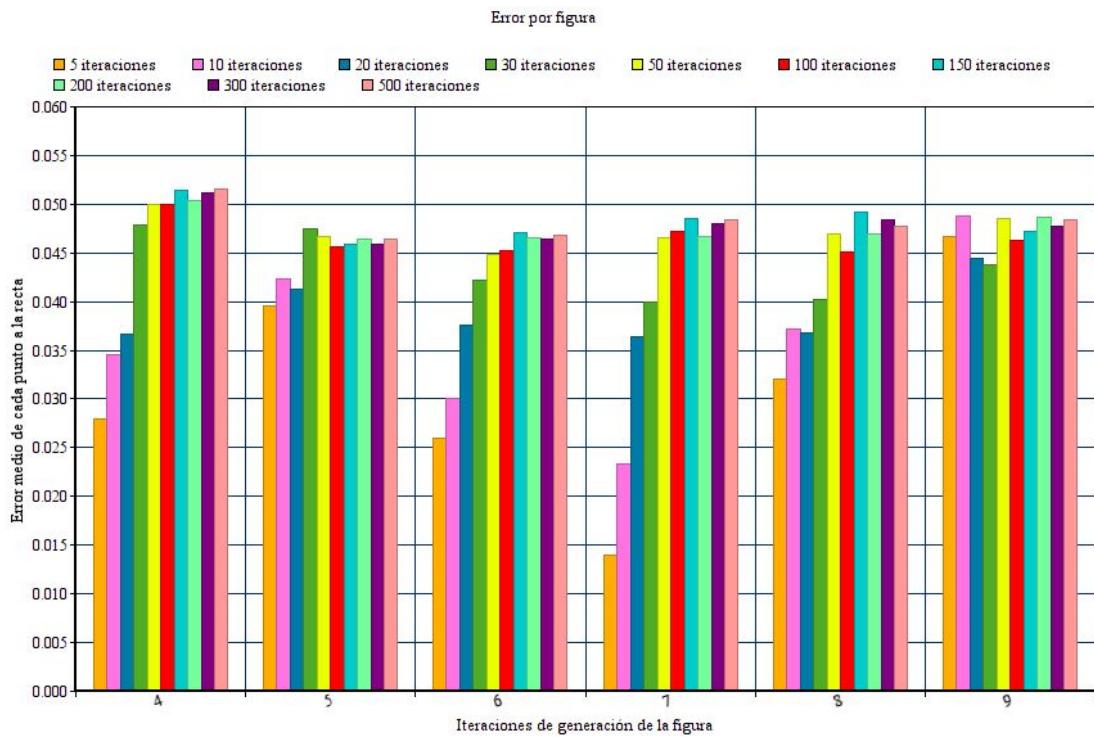
2.4.2.- Modificaciones

El algoritmo ya comienza a ofrecer unos resultados más fiables por lo que no sufre modificaciones en esta fase, simplemente se varían las iteraciones realizadas para observar los efectos producidos.

2.4.3.- Resultados

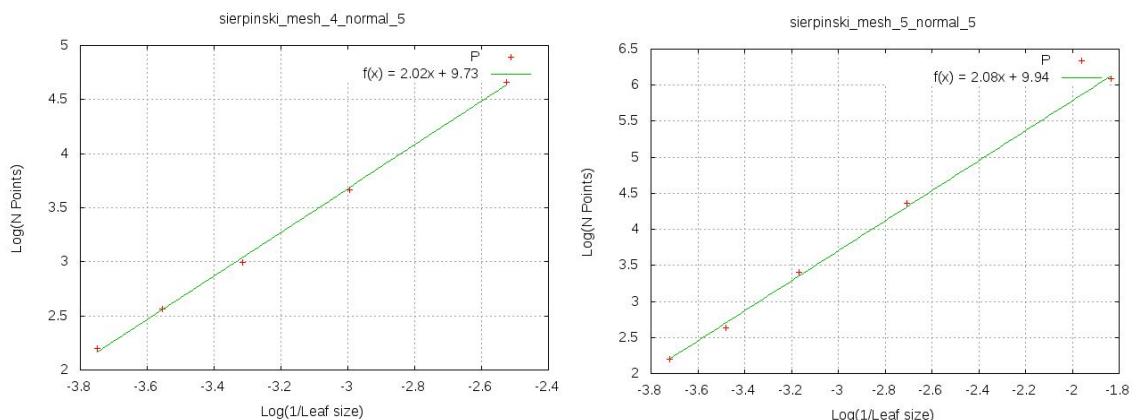
Se ha realizado, para los distintos modelos del tetraedro de Sierpinski, la ejecución del algoritmo con 5, 10, 20, 30, 50, 100, 150, 200, 300 y 500 iteraciones.

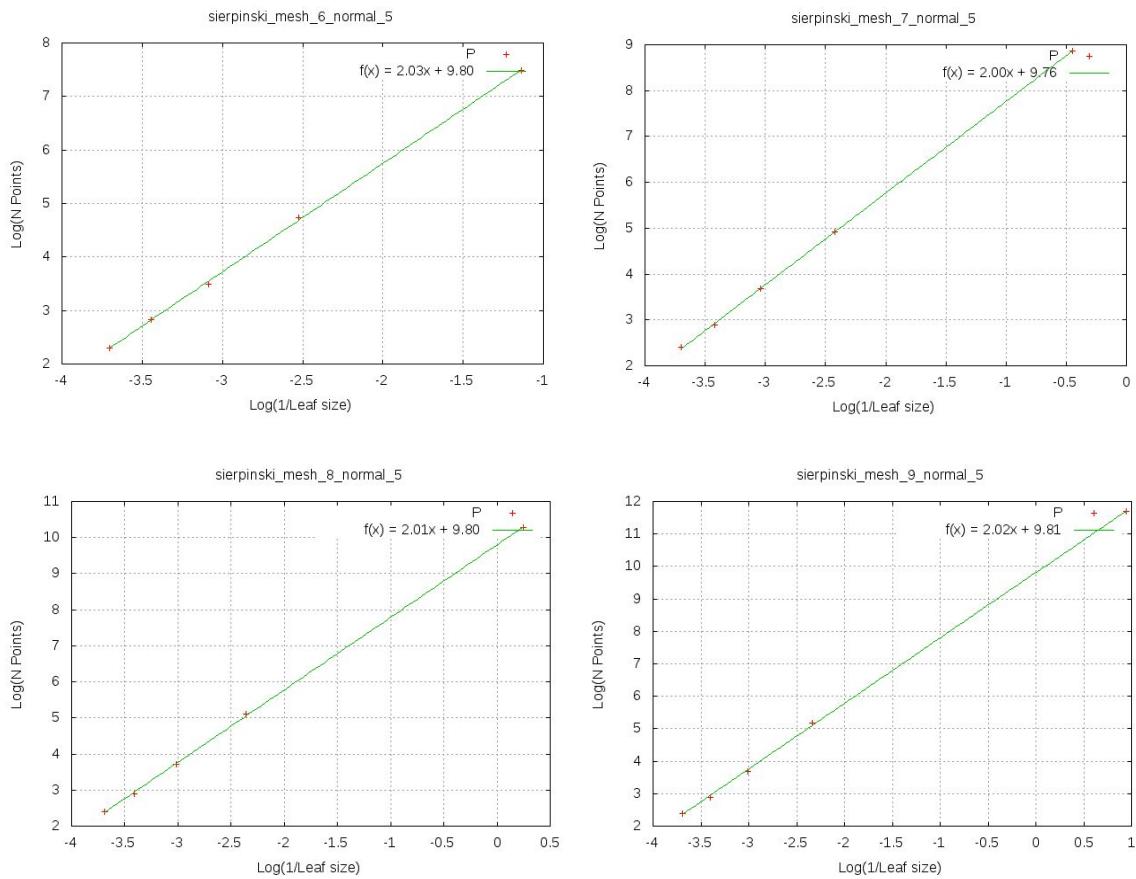
Comenzamos mostrando una gráfica comparativa del error de la regresión lineal producido en cada caso de prueba



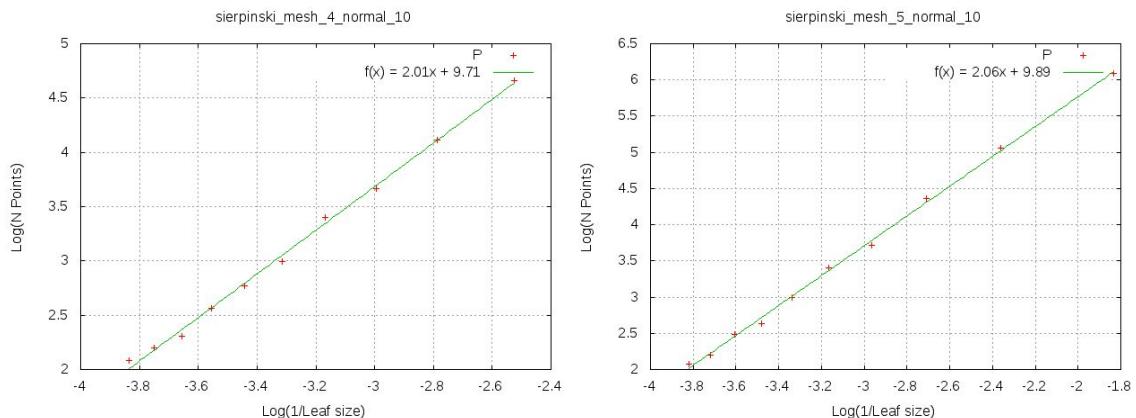
Se muestran a continuación, las gráficas de regresión lineal del tetraedro de Sierpinski:

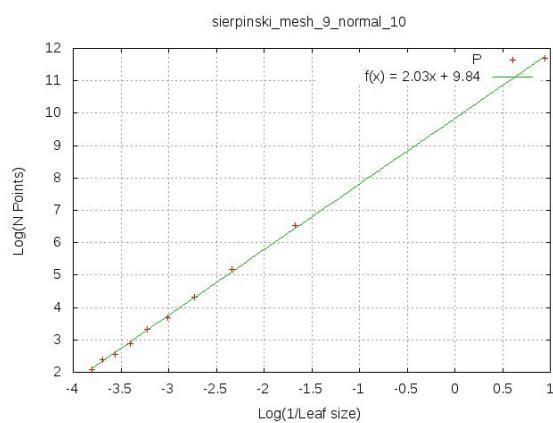
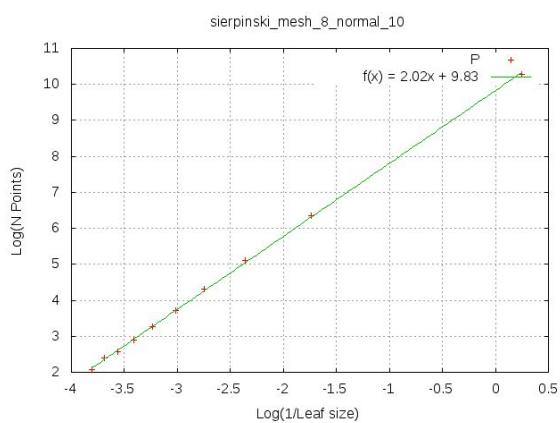
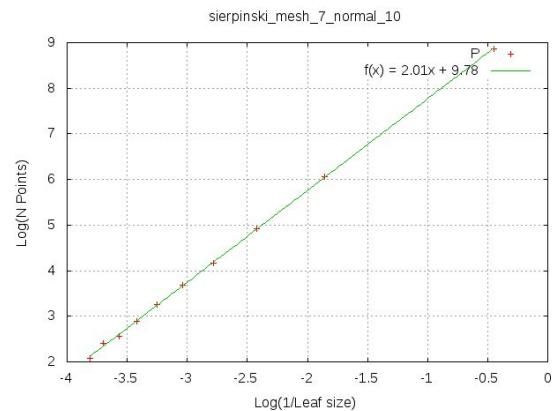
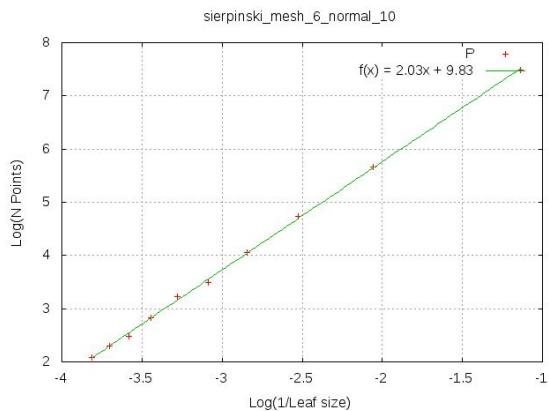
5 iteraciones



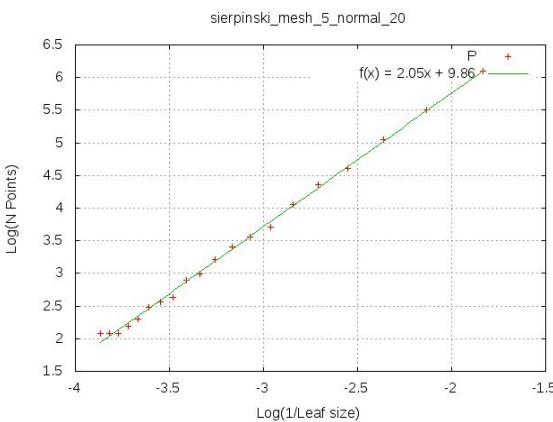
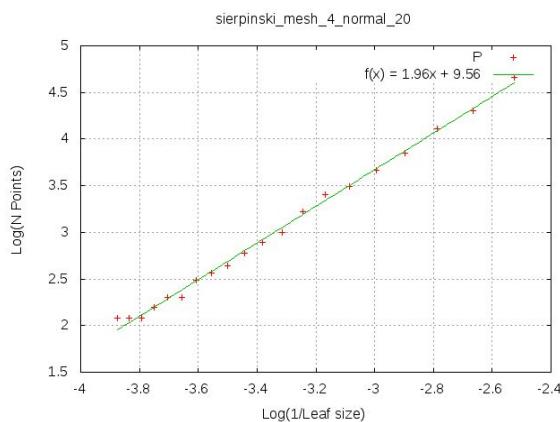


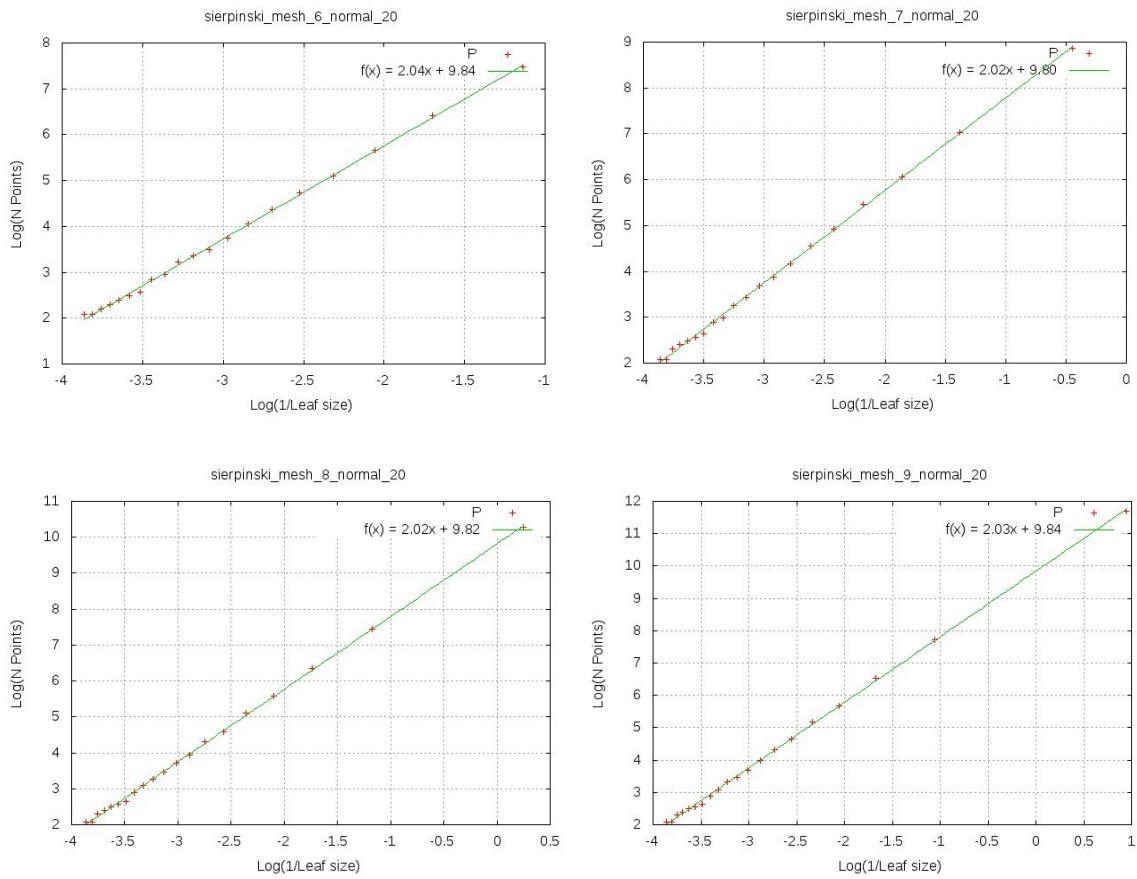
10 iteraciones



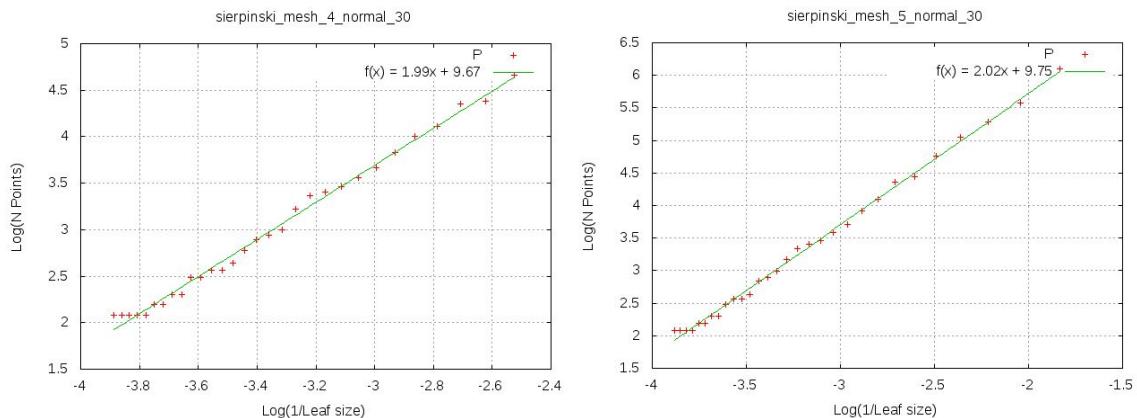


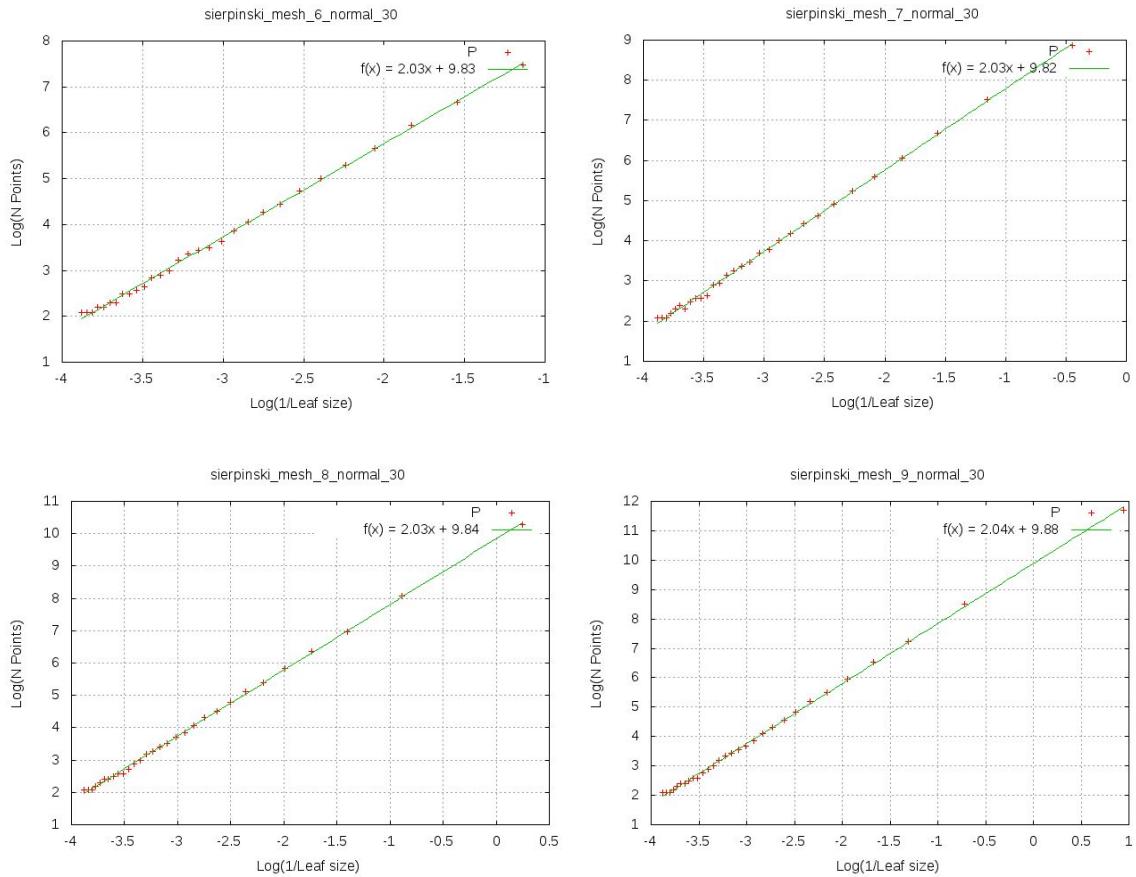
20 iteraciones



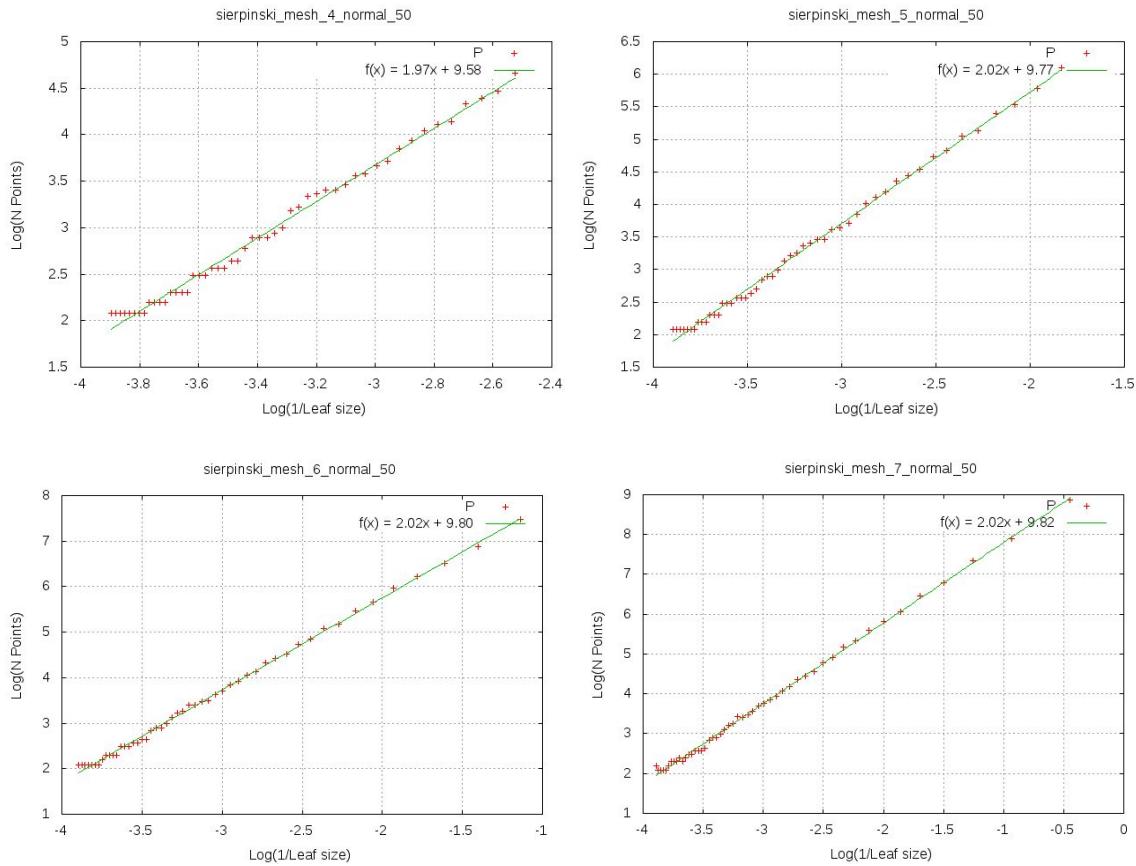


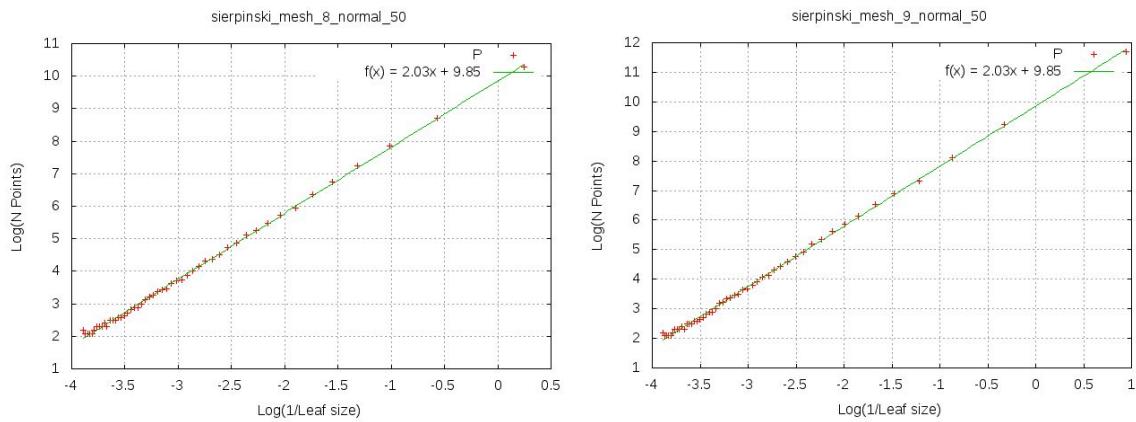
30 iteraciones



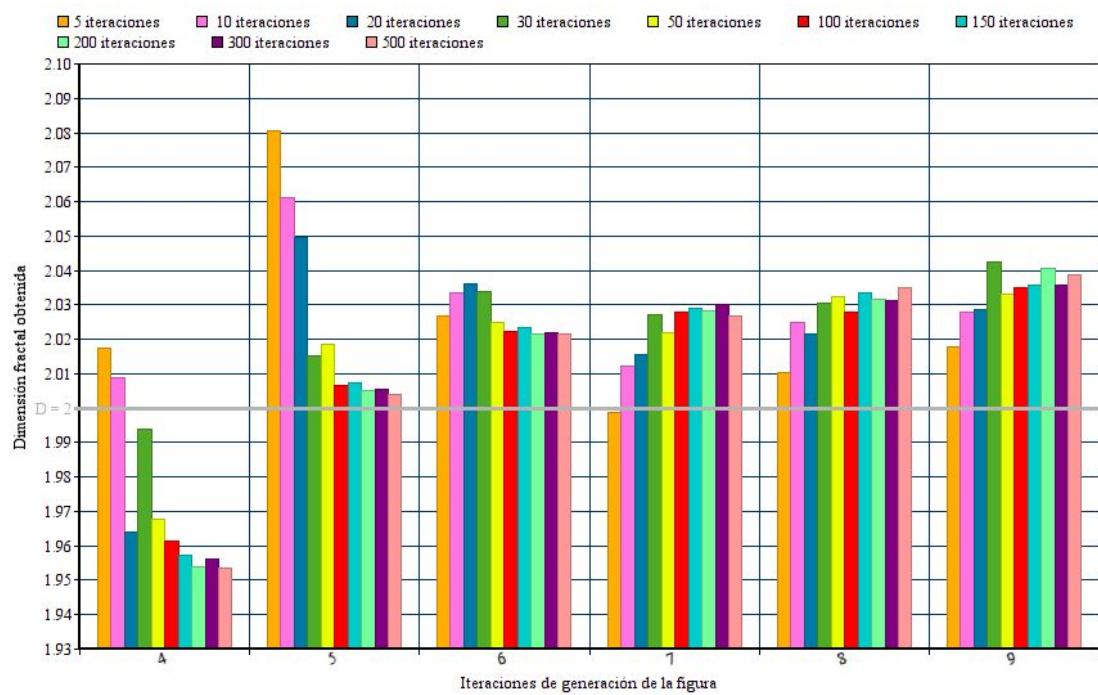


50 iteraciones



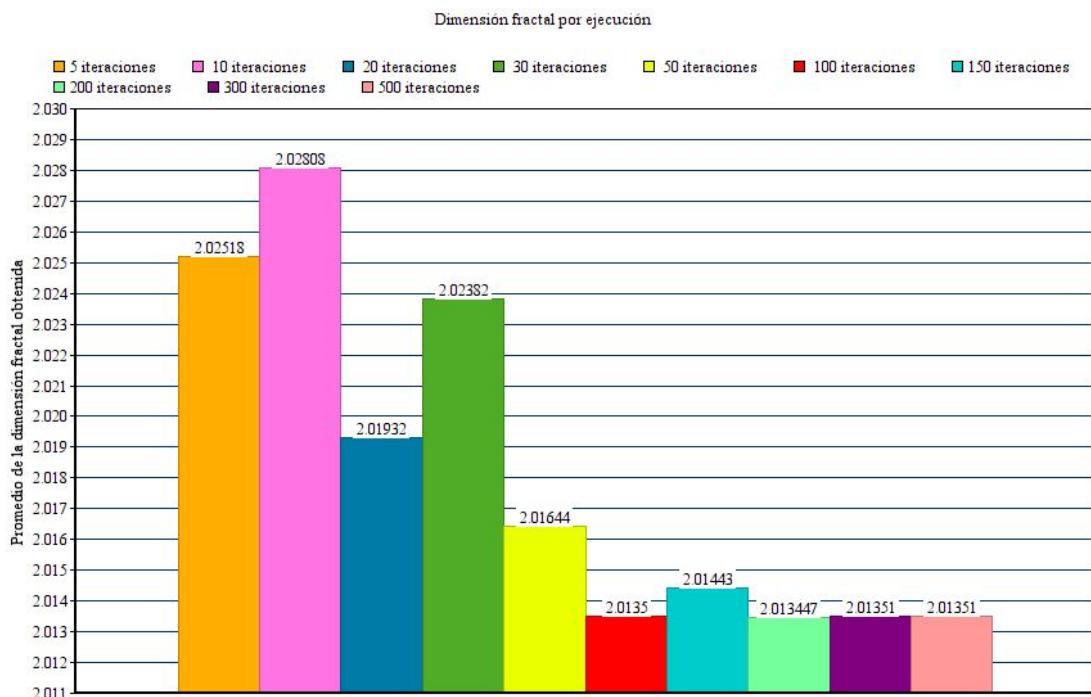


Comparativa de la dimensión fractal obtenida



Promedio de dimensión fractal

La siguiente gráfica representa la media de la dimensión fractal obtenida para todas las figuras, por cada ejecución con diferente número de iteraciones.



2.4.4.- Conclusiones

A continuación se enumeran las ideas y conclusiones surgidas de la experimentación:

- Observando la gráfica comparativa del error promedio en la regresión lineal podríamos pensar que cuantas más iteraciones se hacen en el algoritmo más error se introduce y por tanto tendremos mejores resultados.
- Observando las gráficas de regresión lineal, también se podría llegar a pensar que no es necesario realizar muchas iteraciones del algoritmo ya que realizando 5-10 ya se obtienen resultados bastante ajustados.
- Es en las dos últimas gráficas, sobre todo la última que muestra el promedio de la dimensión fractal calculada de todas las figuras por cada ejecución del algoritmo donde podemos ver que aunque obteniendo más error en la regresión lineal, la ejecución de la fase 3 con 100, 200, 300 y 500 iteraciones son las que muestran resultados más exactos y estables.
- El error en la regresión lineal es un dato importante pero no llega a ser determinante, cuando la diferencia no es muy sustancial, sobre todo en este caso, ya que es más difícil aproximar una recta a 100 puntos que a 5.

- Aún cometiendo un poco más de error en la regresión, vemos que al realizar más iteraciones tenemos más información y esto ofrece mayor estabilidad y fiabilidad en los resultados.
- A partir de 100 iteraciones no se observa ninguna mejora, ni en el error producido, ni en la dimensión fractal calculada, por lo que podemos concluir que realizar más de 100 iteraciones no tiene sentido ya que es más costoso computacionalmente y no aporta nada.

2.5.- Fase 5

2.5.1.- Problemas a resolver

Durante toda la experimentación en fases previas se ha intentado establecer los parámetros y el funcionamiento del algoritmo de forma que sea lo más preciso posible pero de una forma genérica ya que en estas pruebas se están usando figuras muy simétricas y regulares pero el objetivo es poder aplicarlo luego sobre cualquier forma como podría ser el modelo 3D de una persona humana, por lo que no se ha entrado mucho en las particularidades de las gráficas de cada modelo porque no merece la pena ajustar el algoritmo en base a los problemas particulares de cada figura, si no de los fenómenos más comunes en los modelos analizados, y es lo que se ha hecho hasta el momento.

Con todo esto me refiero a que en este punto, el algoritmo parece estar bastante ajustado, pero seguimos viendo diferentes problemas de alineación de los puntos en las gráficas dependiendo del modelo y de las iteraciones con las que se ha generado, especialmente en la pirámide no se han conseguido tan buenos resultados como para el tetraedro.

Como parece que no se pueden establecer los parámetros de una forma general que sirva para todas las figuras, en esta fase se va a probar a utilizar un método de ajuste a modelos geométricos llamado RANSAC.

RANSAC

RANSAC es una abreviatura de “RANdom SAmple Consensus”, que podría traducirse como “consenso de una muestra escogida al azar”.

Es un algoritmo de estimación robusta que permite hallar un modelo matemático a partir de datos contaminados con numerosos valores que no se ajustan al modelo (datos atípicos o “outliers”).

El algoritmo RANSAC busca el mejor modelo considerando todos los puntos de contorno incluidos aquellos que no se ajustan al modelo buscado. Para ello, selecciona aleatoriamente muestras de N de puntos, siendo N el número de puntos necesarios para establecer los

parámetros del modelo: (2 para una recta, 3 para una circunferencia, 5 para una elipse, etc.). En nuestro caso, nuestro modelo es la recta de regresión de los puntos obtenidos por el algoritmo Box Counting.

Una vez calculados los parámetros para cada muestra, se evalúa el conjunto de consenso que es el número de puntos del contorno original próximos al modelo calculado dentro de una tolerancia preestablecida.

Si el nuevo resultado es mejor, entonces se reemplaza el resultado almacenado por el nuevo.

Algoritmo RANSAC para la detección de rectas

Se repiten los siguientes pasos K veces

1. Valores iniciales
 - a. Se selecciona aleatoriamente de entre todos los puntos una muestra de 2 puntos.
 - b. Se calculan los parámetros de la recta que pasa por los 2 puntos, en nuestro caso, se obtiene la fórmula de la recta $y = mx + b$ que pasa por ambos puntos.
2. Verificación
 - a. Determinar el conjunto de consenso, es decir, los puntos de las muestras que se ajustan a la recta hallada con una tolerancia de error ϵ y obtener el tamaño N del conjunto.
 - b. Si el número de inliers en el conjunto de consenso es el más alto encontrado, guardar el modelo actual.

2.5.2.- Modificaciones

Implementación del algoritmo RANSAC:

El algoritmo ha sido implementado conforme a las especificaciones del punto anterior, asignando los siguientes parámetros:

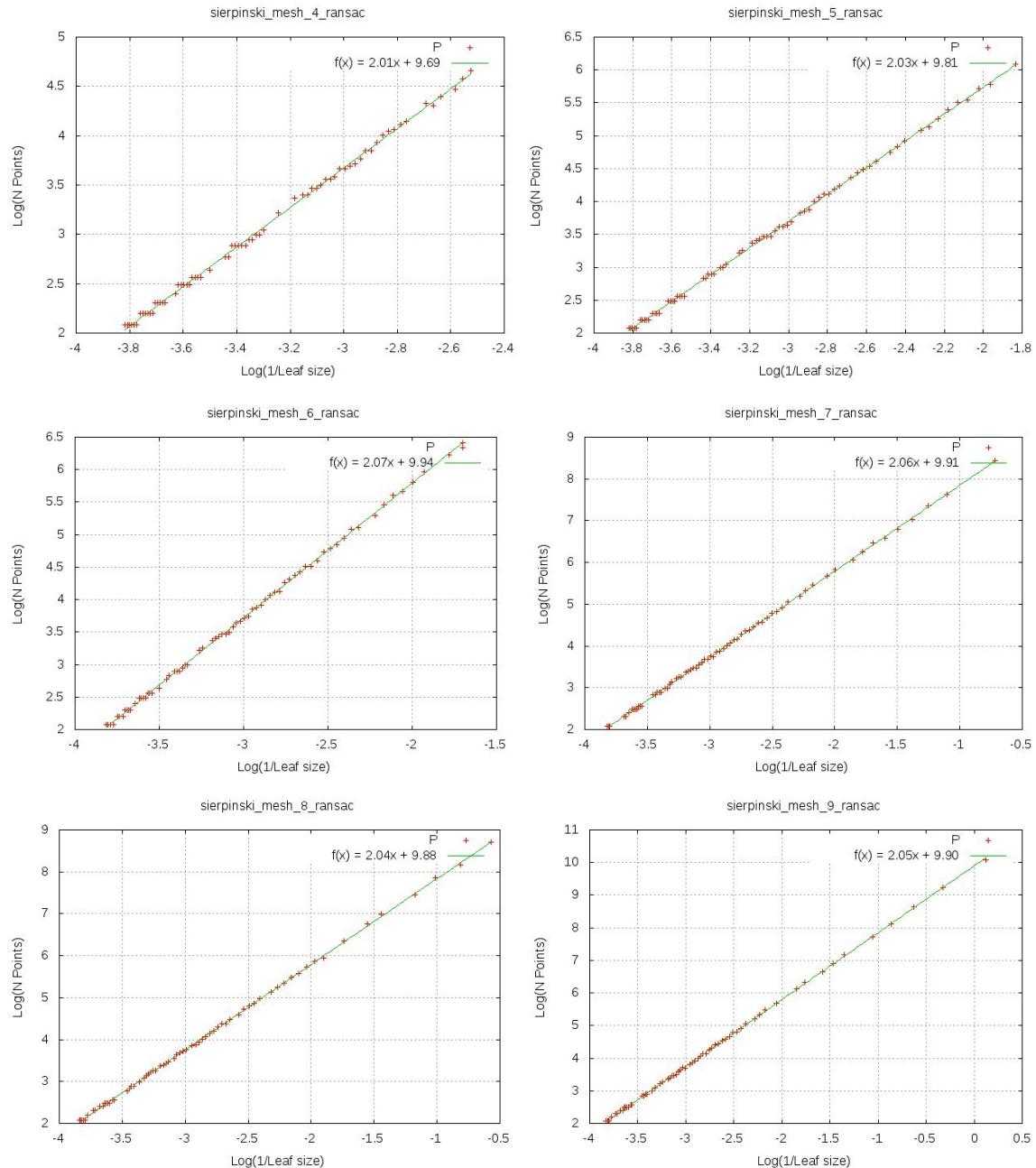
- Tolerancia ϵ : Sobre el conjunto total de puntos (100 ya que se van a usar 100 iteraciones para Box Counting) se calcula la recta de regresión y el error promedio cometido para cada punto. Ese error promedio será ϵ en el cálculo de RANSAC.
- Tamaño N del conjunto de consenso: Al utilizar el error medio de los puntos del conjunto completo, se va a restringir el tamaño del conjunto de consenso como mínimo a la mitad del tamaño del conjunto inicial. Por lo tanto $N = \frac{100}{2} = 50$
- Número de iteraciones K: Se utilizará un número de iteraciones $K = 50$ ya que según los fundamentos teóricos de RANSAC es un número más que suficiente de iteraciones para detectar una recta.

El conjunto final devuelto por RANSAC será el conjunto con más inliers encontrados que cumplan las restricciones de tolerancia y tamaño $\geq N$.

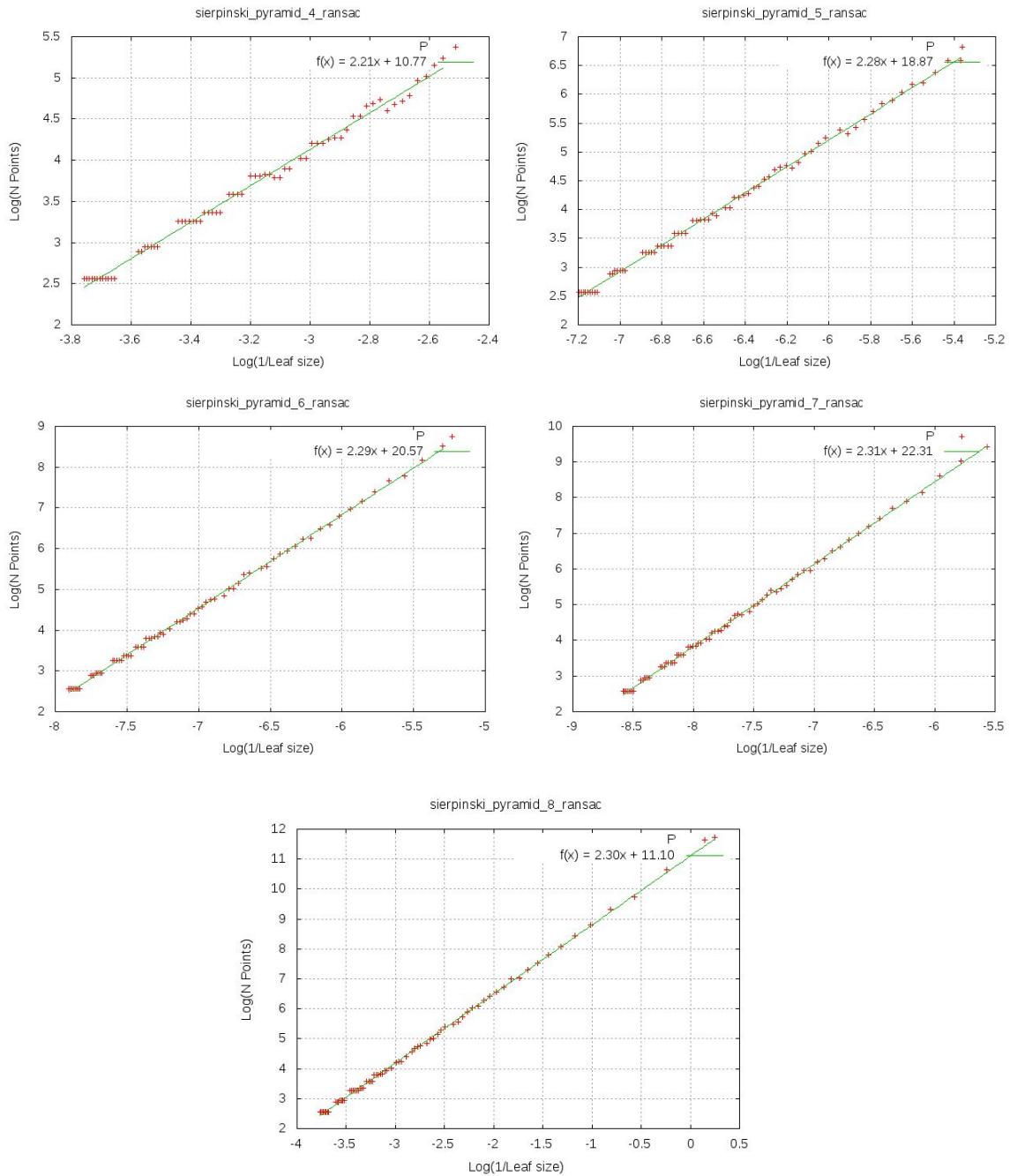
Con RANSAC implementado, lo que se va a realizar es el cálculo de Box Counting conforme a la fase 3 y a continuación se hará un filtrado de los resultados con RANSAC y se imprimirán las gráficas correspondientes.

2.5.3.- Resultados

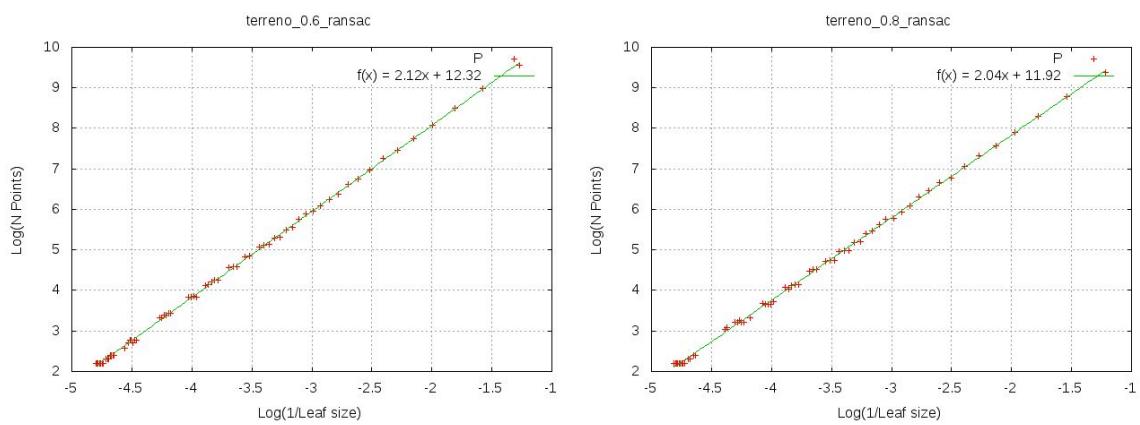
Resultados para el tetraedro de Sierpinski

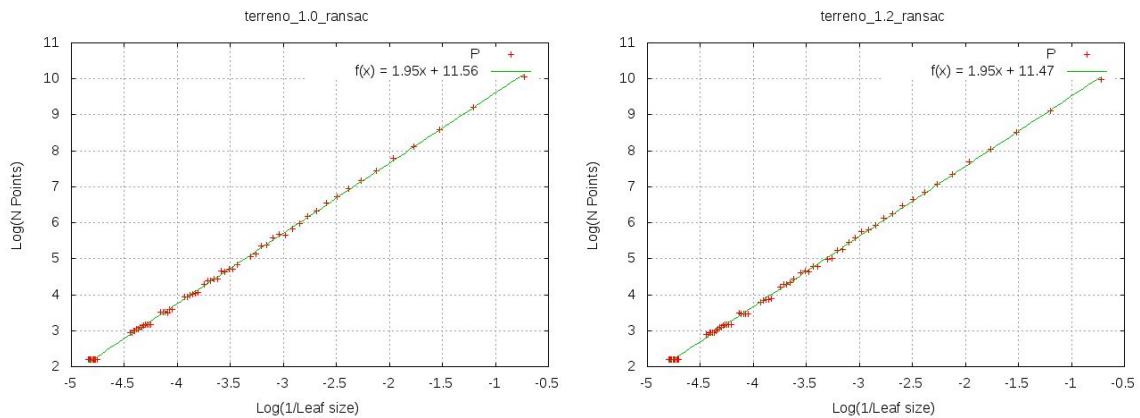


Resultados para la pirámide de Sierpinski:

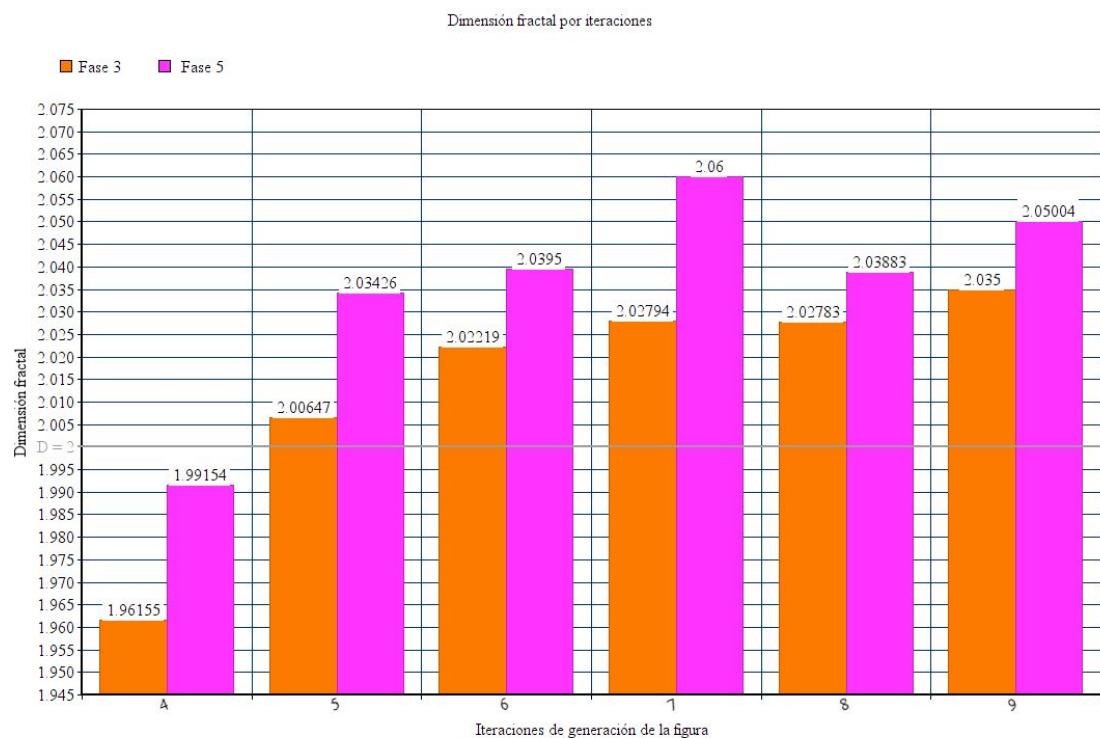


Resultados para los terrenos rugosos:

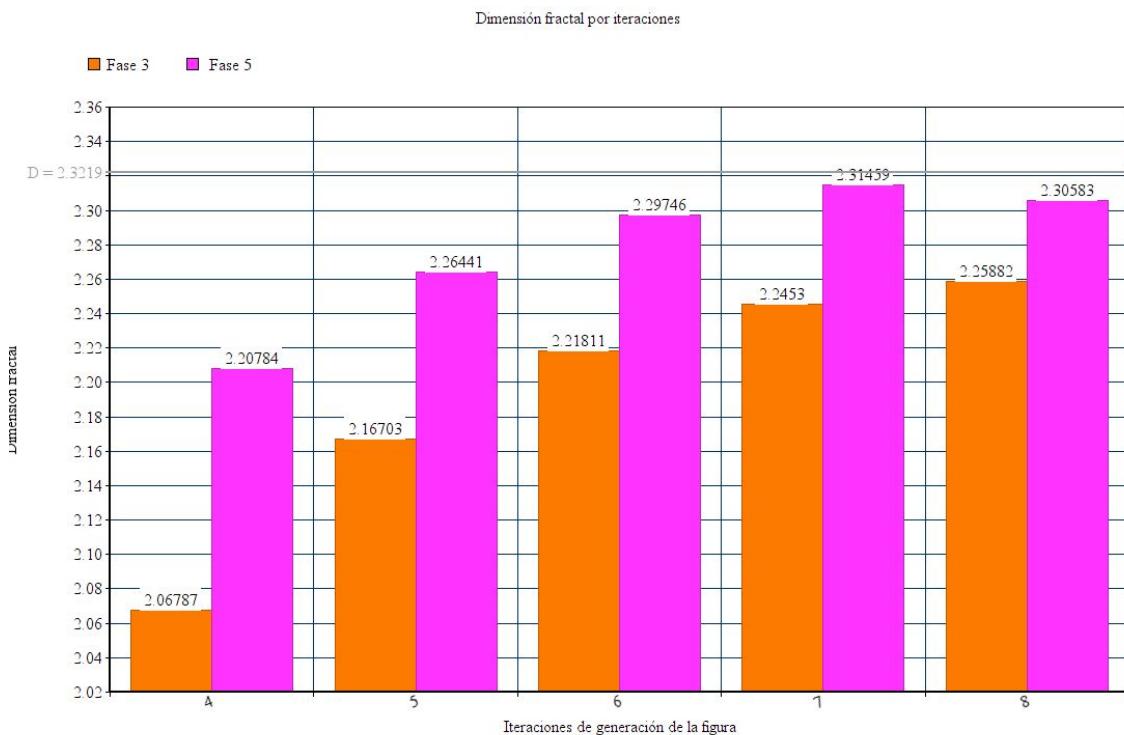




Comparativa de la dimensión fractal calculada entre fase 3 y fase 5 para el tetraedro:



Comparativa de la dimensión fractal calculada entre fase 3 y fase 5 para la pirámide:



2.5.4.- Conclusiones

- Por un lado, vemos claramente que para todos los modelos, los puntos se alinean mucho mejor con la recta de regresión.
- En el caso del tetraedro, aunque los puntos del conjunto devuelto por RANSAC están mejor alineados, vemos que la dimensión fractal obtenida se aleja más de la dimensión teórica ($D = 2$) que en la fase tres, por lo que los resultados no son buenos en este caso.
- En el caso de la pirámide, vemos que en la fase 3 los resultados se alejaban bastante de la dimensión fractal teórica ($D = 2.3219$) y utilizando RANSAC se ha obtenido una mejor aproximación a esta. Si nos fijamos en las gráficas de la recta de regresión para los distintos modelos de la pirámide de Sierpinski en la fase 3, existía un grupo de puntos que se alineaba horizontalmente en la parte izquierda de la gráfica, y con RANSAC se han eliminado casi completamente.
- En el caso de los terrenos rugosos, podemos ver que para los modelos *terreno_1.0* y *terreno_1.2* se obtiene la misma dimensión fractal. Esto no es correcto ya que existe una diferencia de rugosidad entre ellos.
- Al realizar el filtrado con RANSAC obtenemos conjuntos que se alinean de una mejor manera, pero en este caso no implica que esto se aproxime mejor al modelo real ya que sigue teniendo una dependencia sobre cómo se han obtenido los puntos originales.

- Parece que las decisiones para mejorar los resultados finales tendrían que estar más enfocadas a la salida producida por Box Counting y no a la manipulación y el ajuste de estos.
- Al haber producido resultados algo contradictorios, no se considerará el algoritmo en esta fase como una solución válida para todos los modelos.
- RANSAC podría ser útil en casos en los que se detectara demasiado error cometido en el cálculo de la recta de regresión.
- Parece también que no va a ser fácil poder obtener una solución muy precisa en los resultados utilizando Box Counting y que habrá que tratar de ajustar el algoritmo para obtener una solución lo más aproximada posible, pero que no va a ser exacta para cualquier modelo que sea procesado.

2.6.- Fase 6

2.6.1.- Problemas a resolver

En esta fase se va a insistir en la misma cuestión que en la fase 2 y es intentar que no se produzcan ese alineamiento horizontal de puntos por realizar iteraciones con tamaños demasiado grandes.

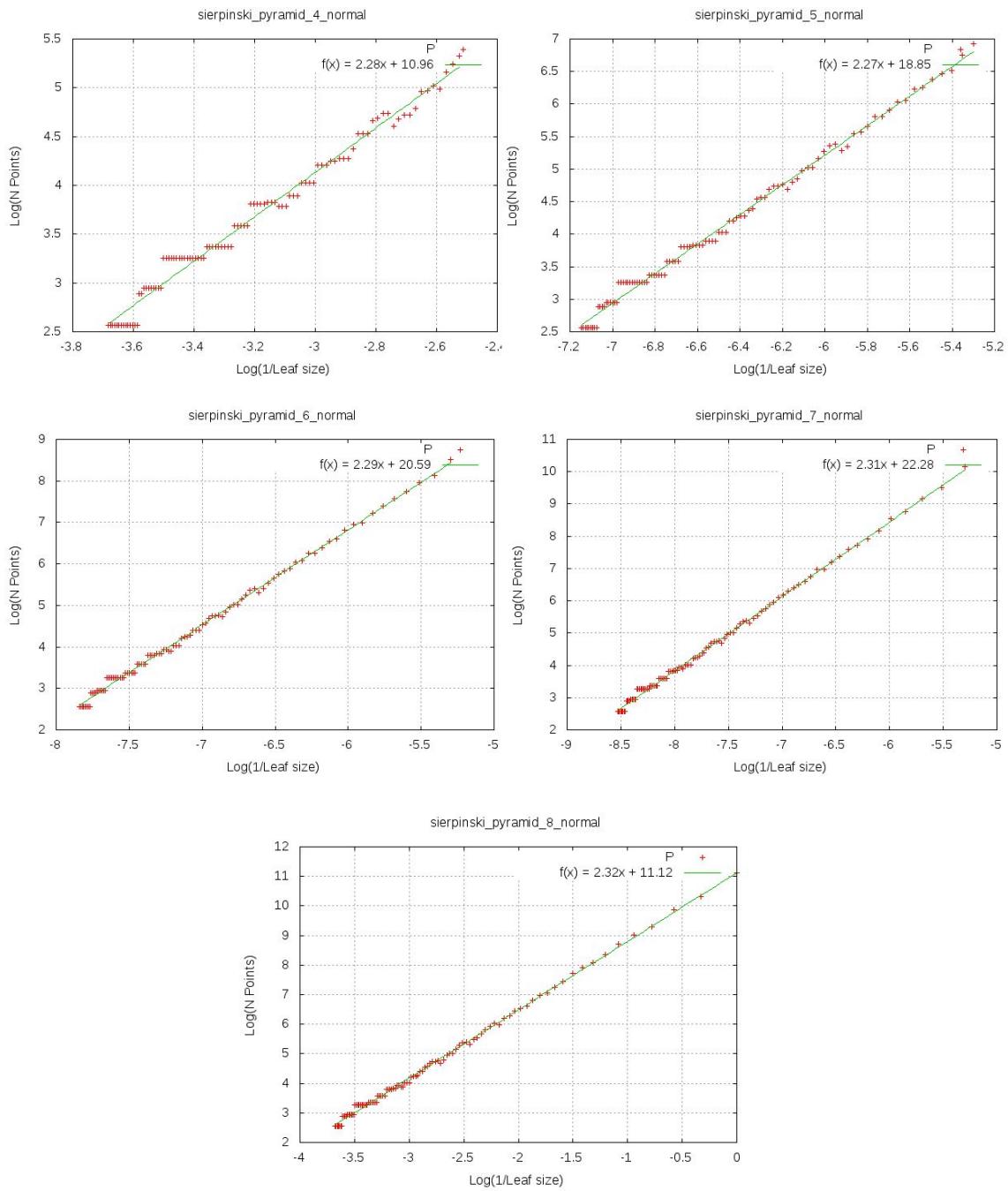
2.6.2.- Modificaciones

Se va a hacer hacer una prueba similar a la de la fase 2, vamos a reducir más aún el tamaño máximo de voxel a utilizar y va a ser igual a $tamaño_max * 0.4$. Esto es, el tamaño de voxel en la última iteración será el 40% de la medida máxima en los 3 ejes del modelo analizado.

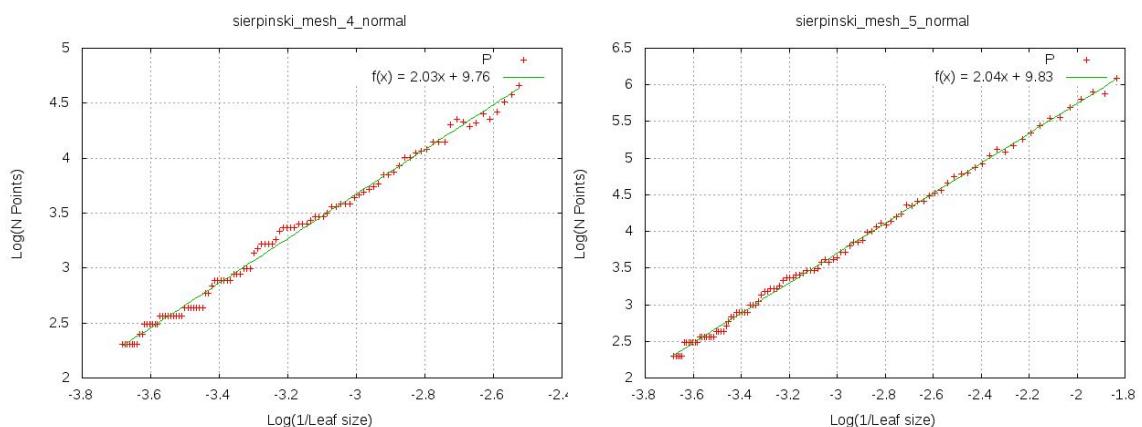
Aunque este problema casi ha desaparecido para los modelos del tetraedro, sigue causando problemas en el resultado de los modelos de la pirámide, así que se va a reducir más el tamaño máximo de voxel intentando que mejore los resultados de la pirámide y no afecte negativamente a los resultados del tetraedro.

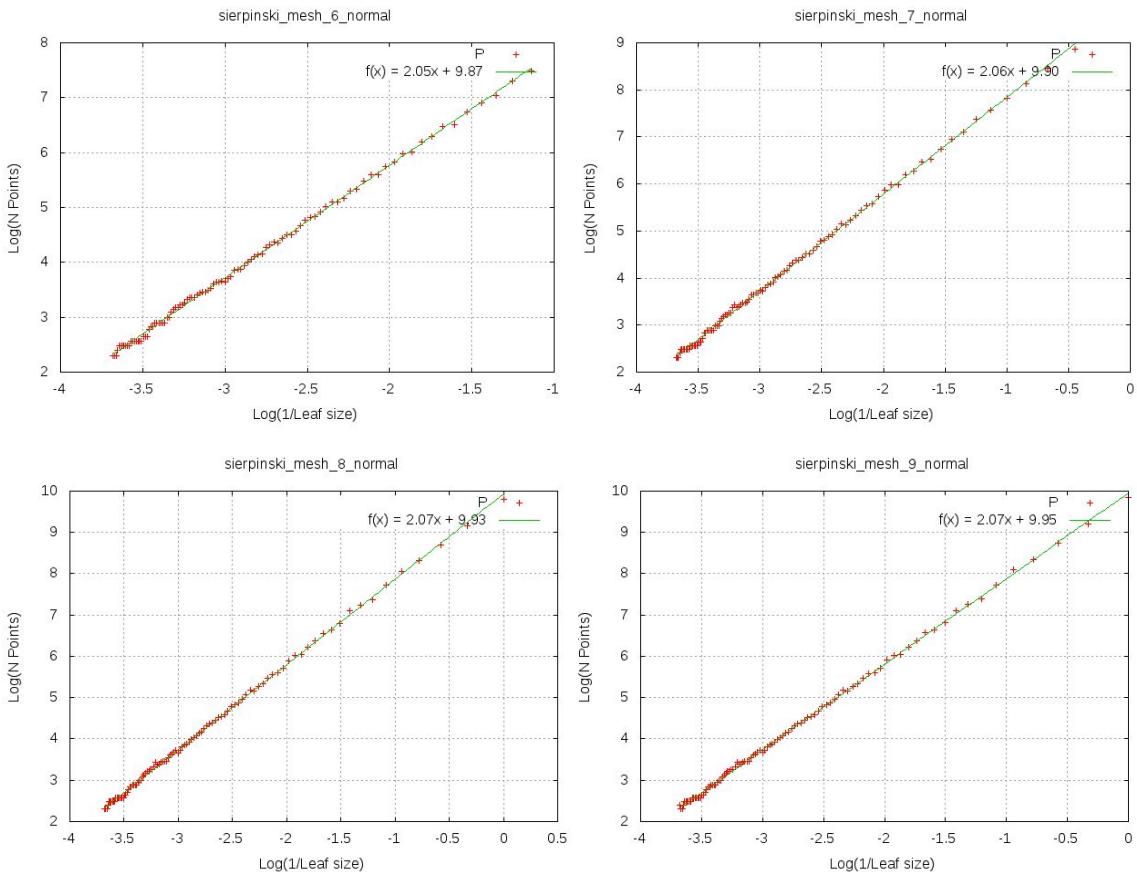
2.6.3.- Resultados

Resultados para la pirámide de Sierpinski:



Resultados para el tetraedro de Sierpinski:





2.6.4.- Conclusiones

- Los resultados en esta fase han sido muy parecidos a los de la fase anterior filtrando con RANSAC.
- En los modelos de la pirámide se ha conseguido el objetivo que era evitar que se obtuviera un grupo de puntos alineados horizontalmente que desviaran el resultado del cálculo de la recta de regresión. Esto se ha traducido en una mejora sustancial en la dimensión fractal calculada, aproximándose mucho a la dimensión teórica.
- Surge el mismo problema para el tetraedro que cuando hemos usado RANSAC. Los puntos de la gráfica parece que se alinean de una mejor forma, pero el resultado producido se aleja unas décimas más respecto a la dimensión fractal teórica que en otras fases de experimentación anteriores.
- Recapacitando sobre lo que ocurre con el tetraedro de Sierpinski en esta fase y en la anterior y observando las gráficas de la fase 3, vemos en estas gráficas que aunque el resultado se aproxima más al teórico, existen puntos alineados horizontalmente que están afectando al cálculo de la recta de regresión.

Lo que estamos intentando captar es la tendencia de los puntos a alinearse sobre una recta, por lo que estos puntos alineados horizontalmente que pueden apreciarse

claramente a la vista no deberían formar parte de las muestras utilizadas para el cálculo de la recta de regresión.

Es por ello que aunque veamos resultados a priori mejores en la dimensión fractal calculada en la fase 3 para el tetraedro de Sierpinski, se van a dar por válidas las soluciones de las fase 5 y 6 ya que aunque los resultados no sean tan buenos como los que nos gustaría, sí que se observa una alineación bastante buena de los puntos sobre la recta de regresión.

- Teniendo todo lo anterior en cuenta, es posible que la mejor solución hasta el momento sea la de la fase 6 ya que al utilizar RANSAC lo que se está haciendo es obtener el conjunto de puntos que mejor se alinean con su recta de regresión calculada. Es posible que se esté captando muy bien el patrón, teniendo cuenta que el propio algoritmo Box Counting no es perfecto ni exacto, y quizás estamos llegando al punto en que no se pueden obtener resultados mucho mejores por las limitaciones propias del algoritmo teórico.

2.7.- Fase 7

2.7.1.- Problemas a resolver

En esta fase, más que resolver problemas detectados en fases anteriores, se va a proponer una modificación en la forma de obtener el tamaño de voxel para cada iteración y el número de iteraciones a realizar.

2.7.2.- Modificaciones

La solución que se propone en esta fase consiste en las siguientes modificaciones:

En el planteamiento del algoritmo desde la primera fase, se ha obtenido el tamaño de los voxels en cada iteración empezando desde un tamaño mínimo hasta un tamaño máximo, dividiendo la diferencia entre el número de iteraciones, y realizando un incremento igual en cada iteración. Ahora lo que se va a realizar es el cálculo en cada iteración i dividiendo *tamaño_max* entre *i+1*, esto es:

- $i = 1 \rightarrow \text{tamaño_voxel} = \text{tamaño_max} / 2$
- $i = 2 \rightarrow \text{tamaño_voxel} = \text{tamaño_max} / 3$
- $i = 3 \rightarrow \text{tamaño_voxel} = \text{tamaño_max} / 4$

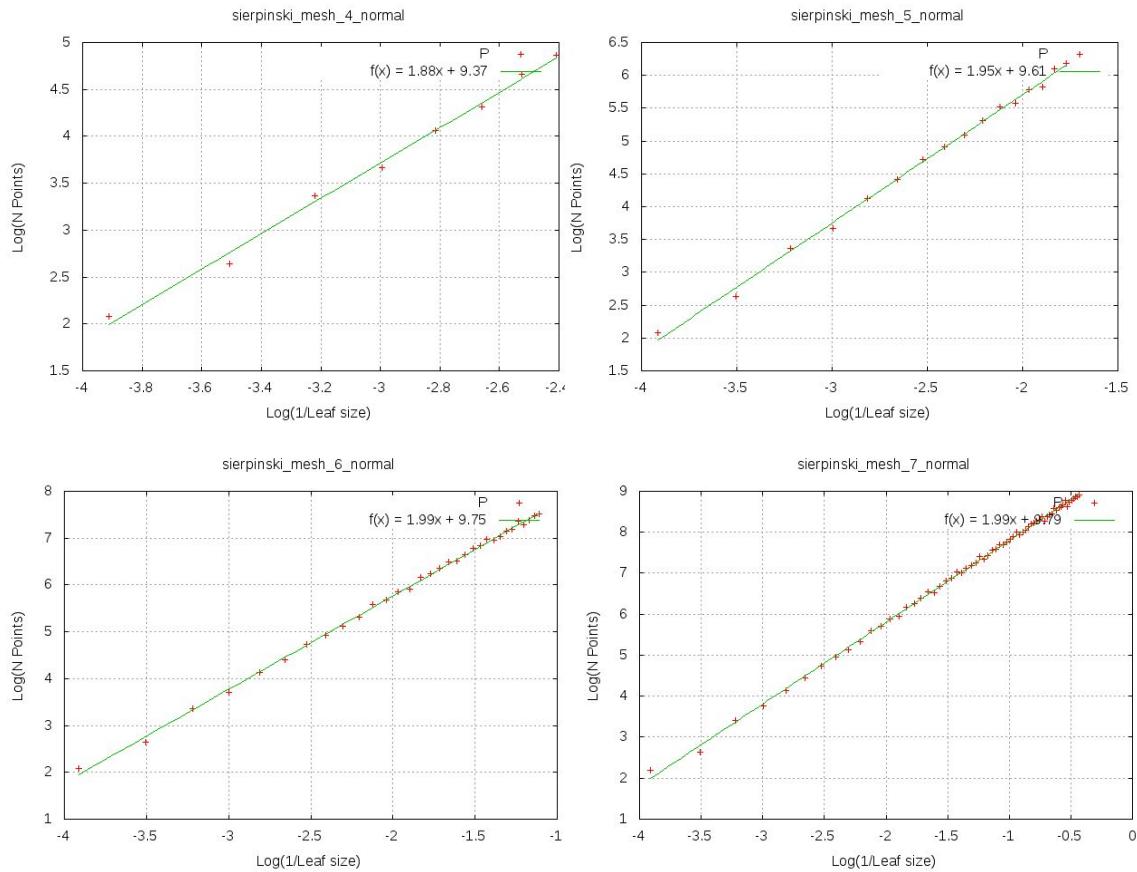
Se van a realizar iteraciones hasta llegar al tamaño máximo de resolución de la nube de puntos 3D, recordemos que este dato viene dado por la distancia media entre el vecino más cercano de cada punto de la nube.

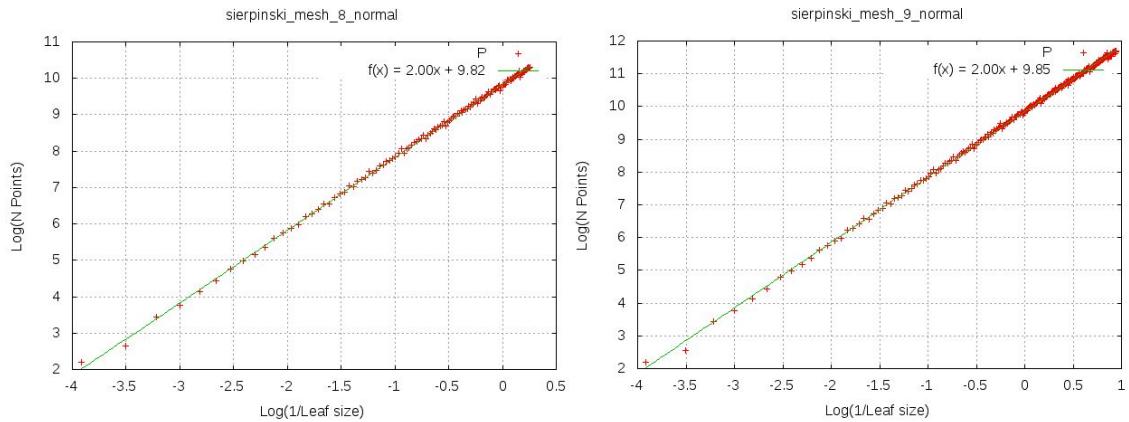
De esta forma conseguimos dos objetivos que han sido valorados en las conclusiones de distintas fases de experimentación:

- Tenemos un número de iteraciones variable según las particularidades de cada figura, de forma automática e independiente de la decisión que tome el “usuario” que recordemos es el que tiene que indicar al algoritmo cuantas iteraciones realizar.
- Tenemos un tamaño de voxel, en todas las iteraciones, a escala siempre del tamaño máximo del objeto y puede resultar en una mejor alineación de los puntos sobre la recta de regresión.

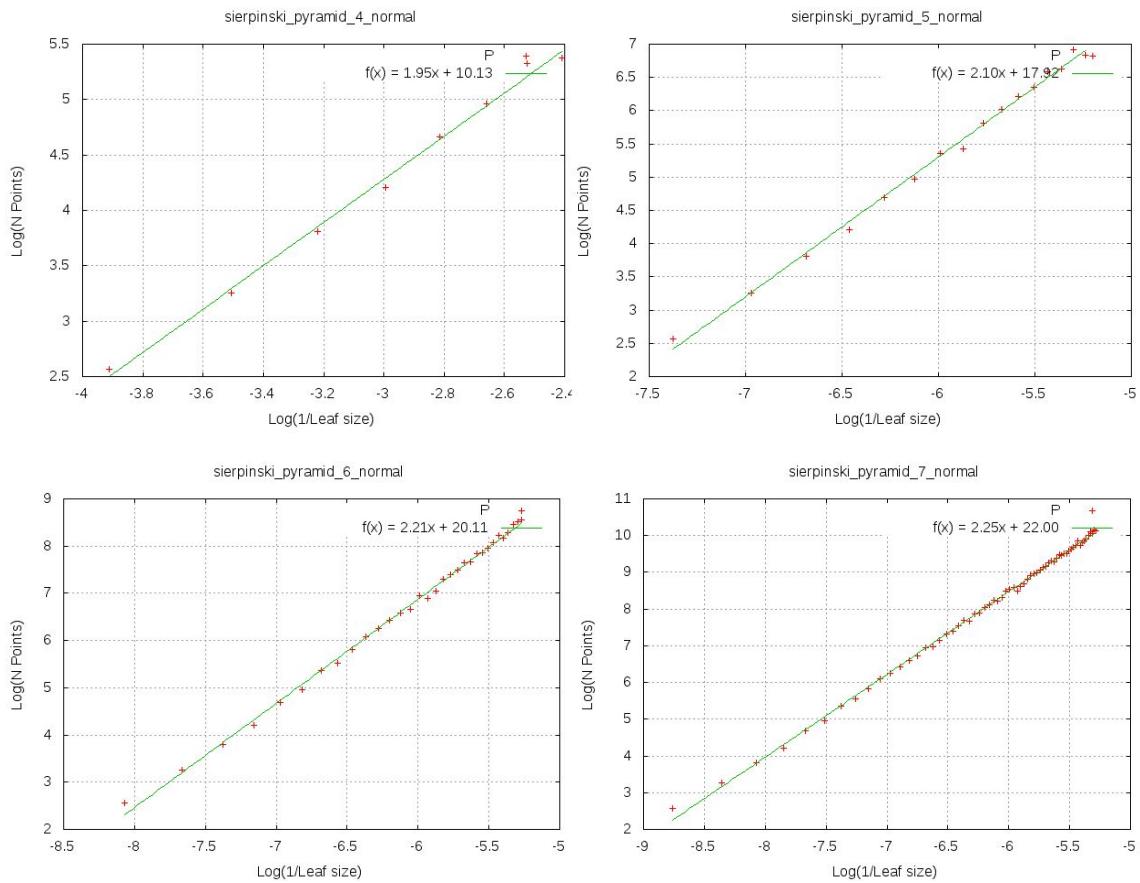
2.7.3.- Resultados

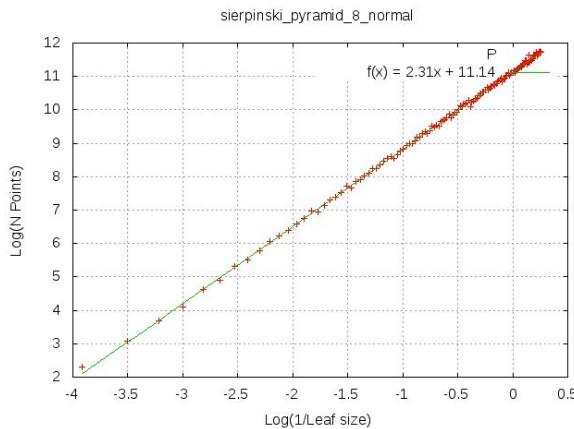
Resultados para el tetraedro de Sierpinski:



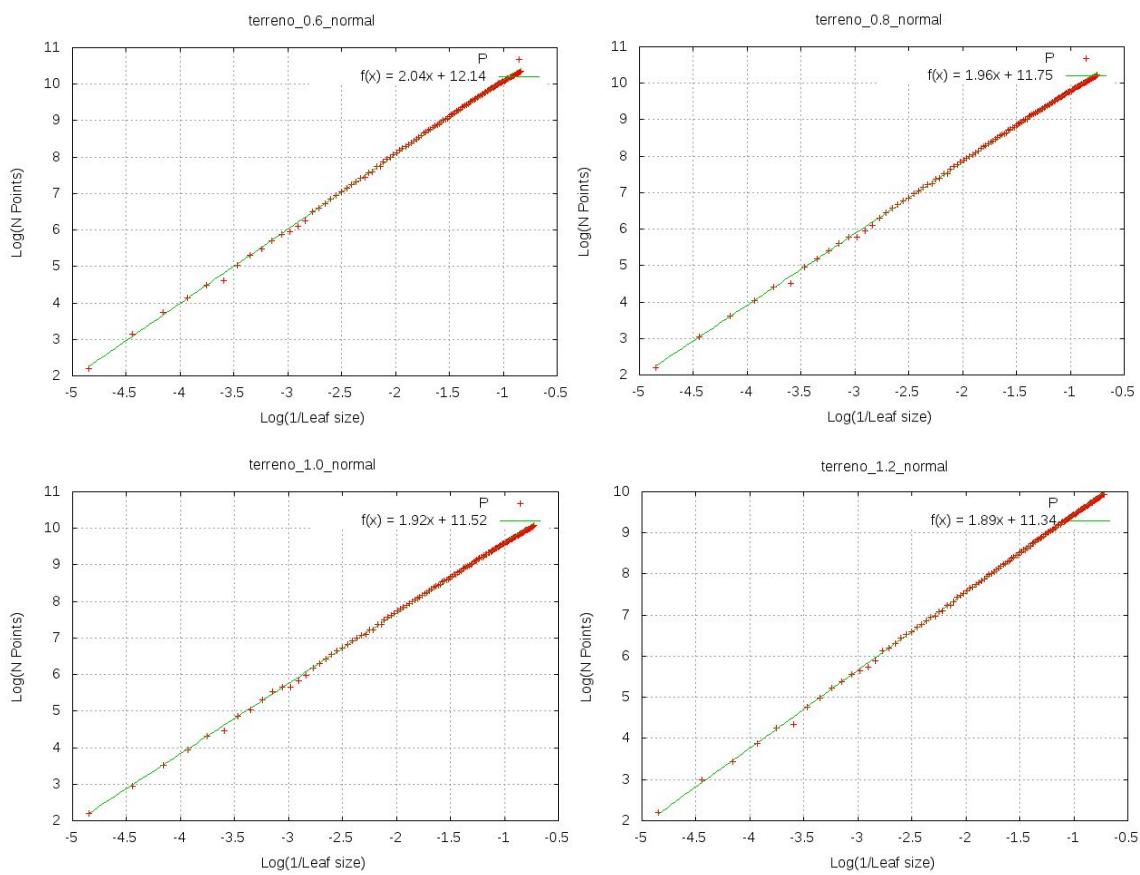


Resultados para la pirámide:





Resultados para los terrenos rugosos:



2.7.4.- Conclusiones

- Vemos en los resultados de esta fase una alineación de los puntos increíblemente buena con la recta de regresión, para todas las figuras.
- En el caso del tetraedro, vemos que, aunque en las figuras generadas con 4 y 5 iteraciones tenemos pocas muestras debido a la densidad de los modelos y la dimensión fractal obtenida no es demasiado aproximada. Para las figuras generadas con 6, 7, 8 y 9 iteraciones obtenemos unos resultados estables y muy buenos.

Esto son muy buenas noticias, ya que, de cara a aplicar el algoritmo a objetos reales, los modelos 3D que los representarán tendrán una densidad más próxima a las figuras generadas con más iteraciones, y no con la densidad que tienen las figuras generadas con 4 o menos iteraciones.

- En el caso de la pirámide, vemos que hasta la figura generada con 8 iteraciones no se alcanza un resultado bien aproximado y válido.

El problema que tenemos con éste modelo es que resulta muy costoso realizar la generación a partir de la iteración 8 y no se tienen los componentes hardware que permitan realizar estos cálculos en un tiempo factible, ya que en la iteración 8 la nube estaría compuesta de X puntos y se multiplicaría por 5 en la nube obtenida en la iteración 9. Esto implica un gran coste computacional tanto al generar como al procesar la nube.

Se podría suponer, aunque sin llegar a verificarlo, que si tuviéramos modelos de la pirámide con más de 8 iteraciones, ocurriría el mismo fenómeno que para el tetraedro y es la estabilización del resultado a un número muy aproximado a la dimensión fractal teórica en las figuras más densas, o que contienen más veces el patrón geométrico de estructura.

- Para profundizar en cuán buena es esta implementación del algoritmo, se podrían generar nuevos modelos fractales cuya dimensión fractal sea conocida y realizar una experimentación más extensa.
- Quizás se podría intentar aplicar RANSAC sobre el conjunto de puntos producidos por esta implementación del algoritmo y ver si se obtiene una mejora en los resultados

2.8.- Fase 8

2.8.1.- Problemas a resolver

Dado el éxito en los resultados obtenidos para la fase 7, en esta fase se va a realizar una pequeña experimentación realizando una variación respecto a la anterior, con la intención de que el algoritmo sea capaz de aproximar la dimensión fractal de los modelos menos densos de una forma más precisa. Sobre todo lo que se pretende es que el algoritmo sea capaz de dar un buen resultado para los modelos de la pirámide generados con menos de 8 iteraciones.

2.8.2.- Modificaciones

Se va a modificar la forma de obtener el tamaño de voxel de cada iteración del Box Counting haciendo variaciones más pequeñas de una iteración a otra respecto a la fase 7.

Ahora el procedimiento será el siguiente:

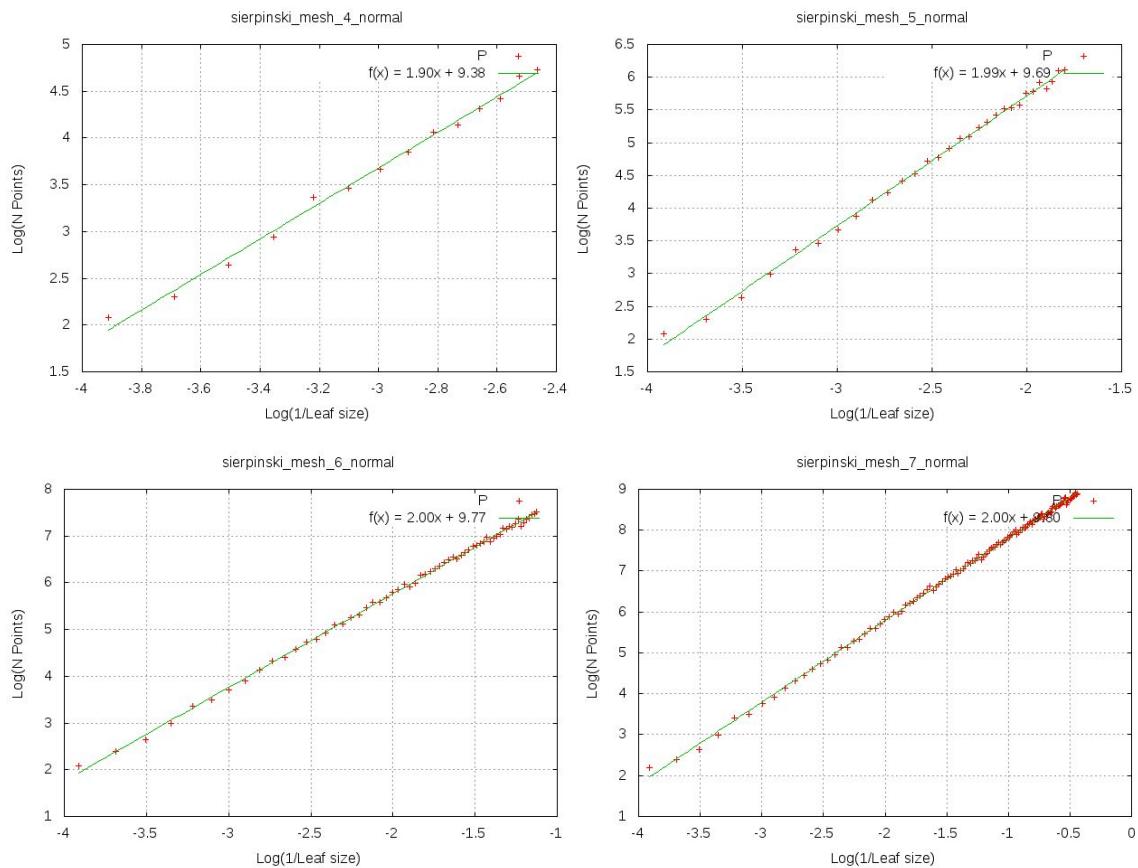
- $i = 1 \rightarrow \text{tamaño_voxel} = \text{tamaño_max} / 2$
- $i = 2 \rightarrow \text{tamaño_voxel} = \text{tamaño_max} / 2.5$
- $i = 3 \rightarrow \text{tamaño_voxel} = \text{tamaño_max} / 3$
- $i = 4 \rightarrow \text{tamaño_voxel} = \text{tamaño_max} / 3.5$

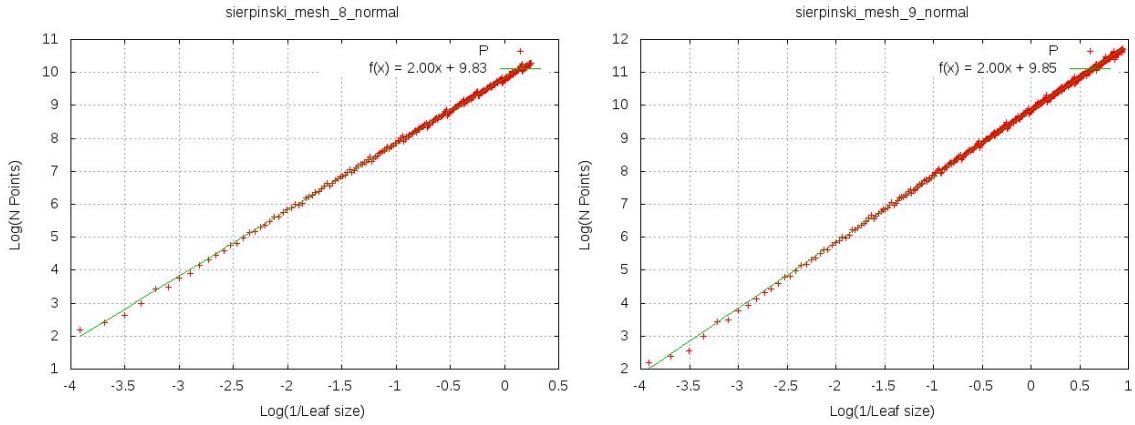
Y así sucesivamente hasta alcanzar la máxima resolución del objeto.

Esto permitirá tener el doble de iteraciones para una misma figura que en la fase 7, consiguiendo así obtener más información para las figuras que tienen menos densidad.

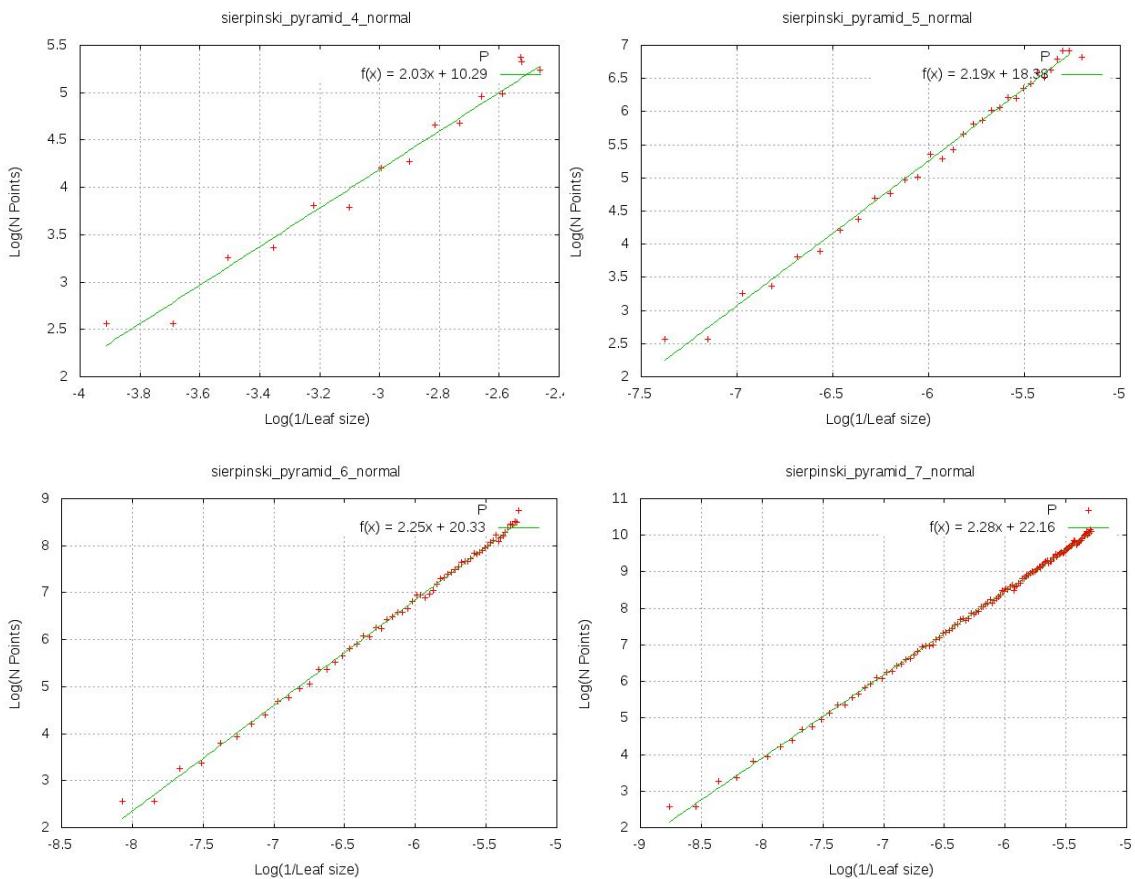
2.8.3.- Resultados

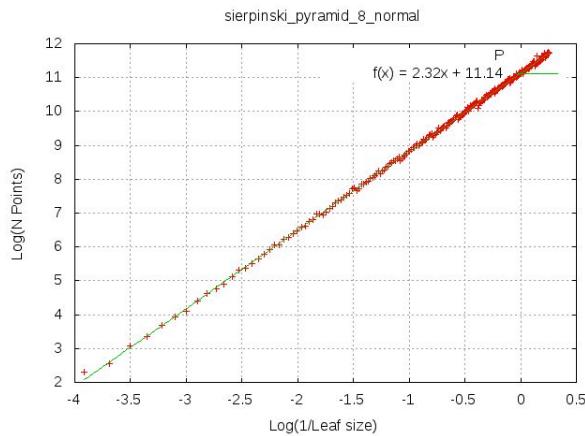
Resultados para el tetraedro de Sierpinski:



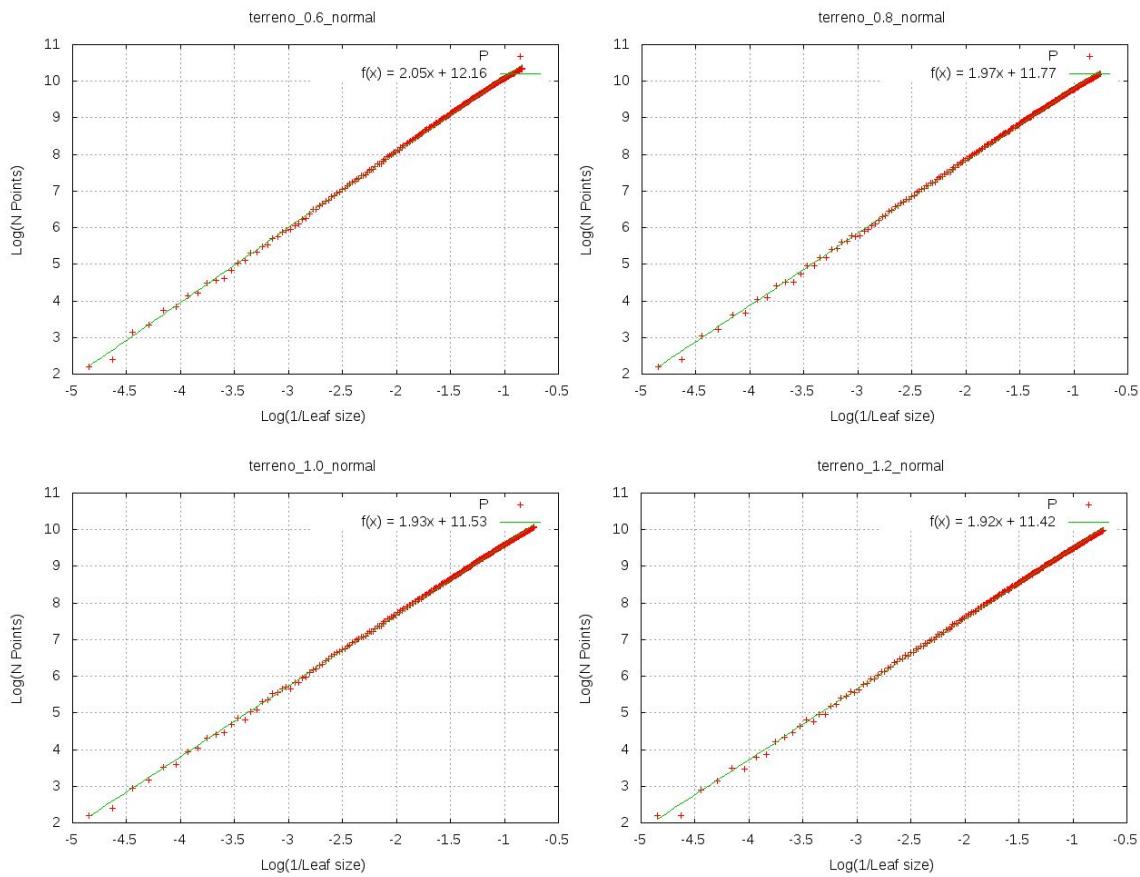


Resultados para la pirámide de Sierpinski:

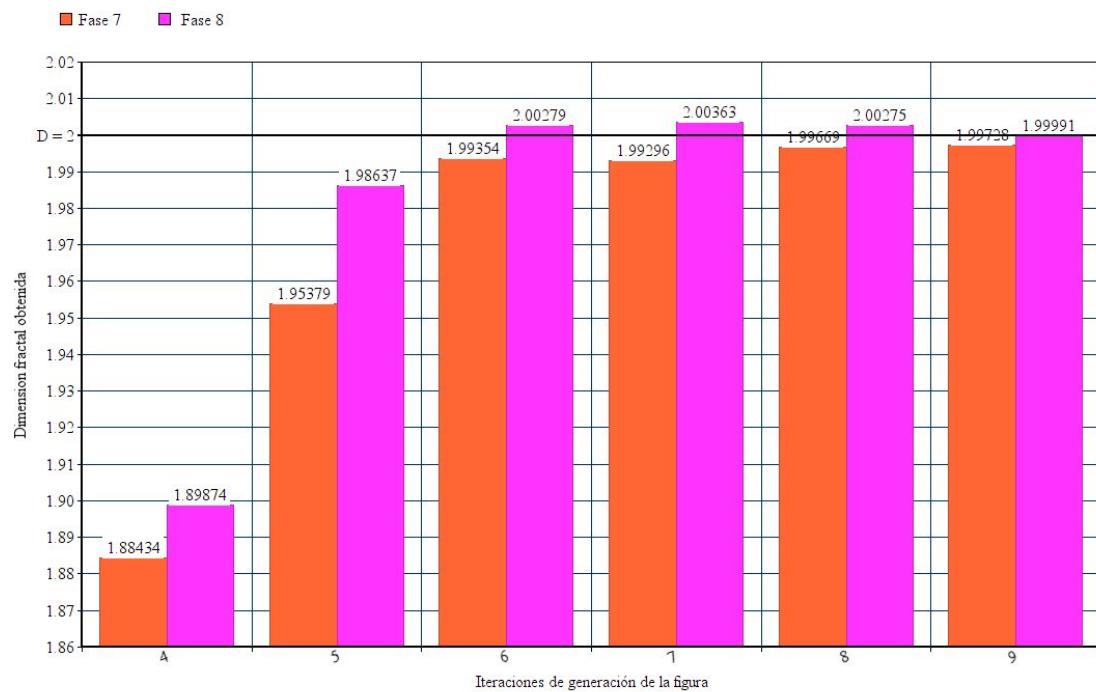




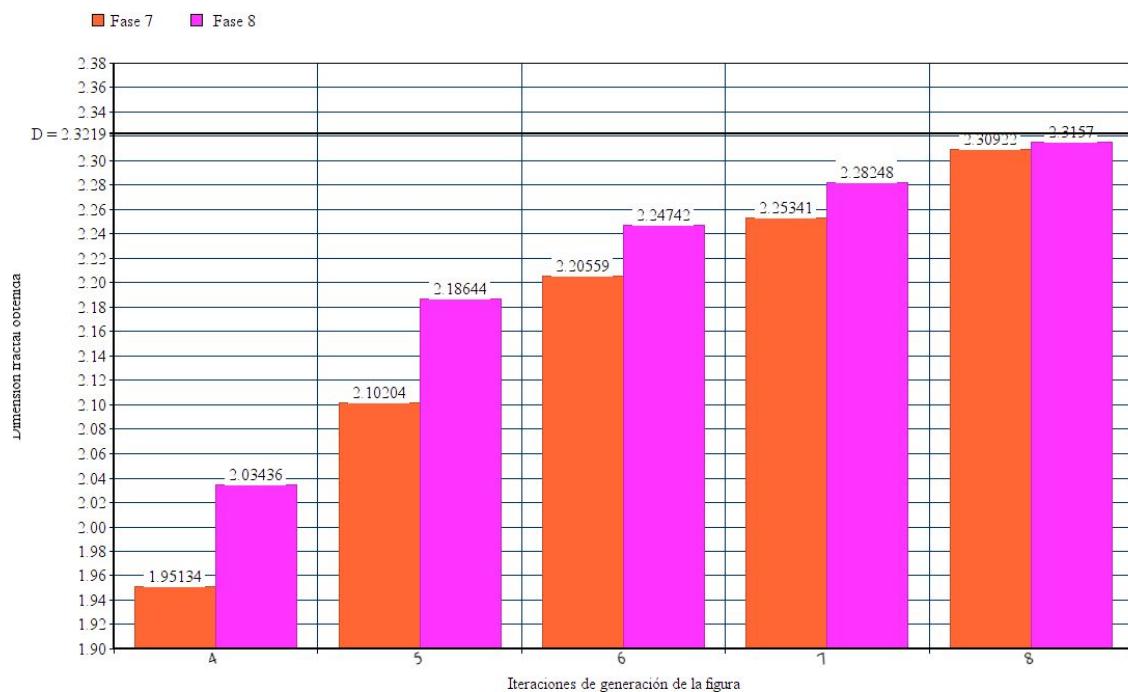
Resultados para los terrenos rugosos:



Comparativa de la dimensión fractal obtenida para cada figura del tetraedro en las fases 7 y 8:



Comparativa de la dimensión fractal obtenida para cada figura de la pirámide en las fases 7 y 8:



2.8.4.- Conclusiones

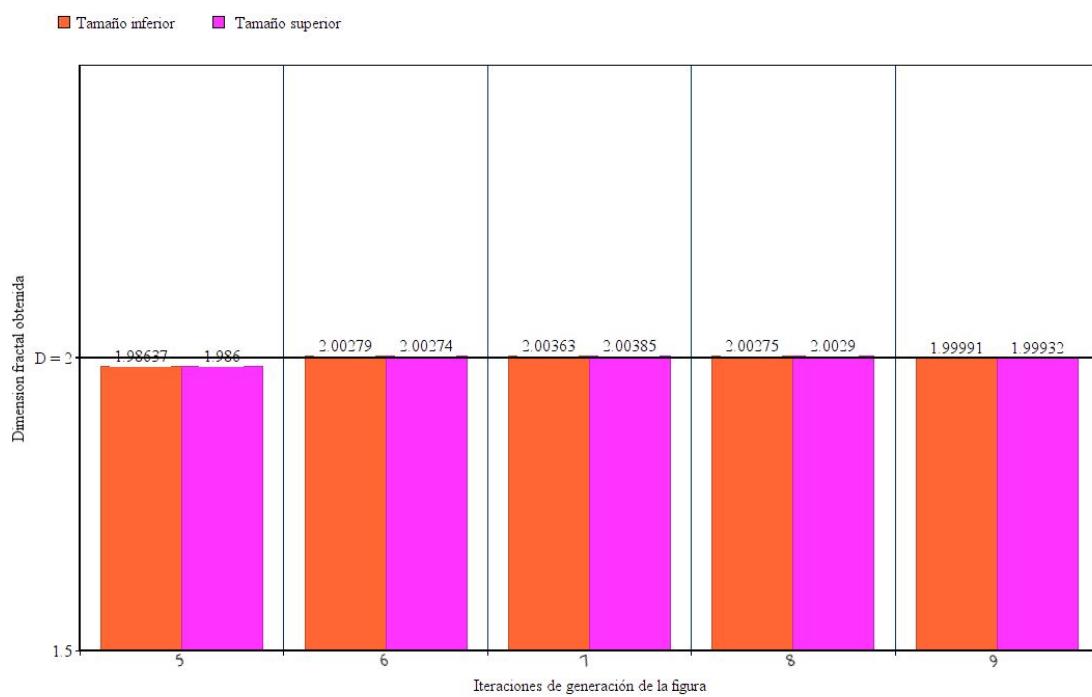
- Hemos llegado a lo que parece una muy buena solución del algoritmo, ya que vemos una alineación casi perfecta de todas las muestras con su recta de regresión.
- En el caso del tetraedro de Sierpinski, hemos obtenido como resultado 2.00 en las figuras 6, 7, 8 y 9, y en la 5 el resultado es 1.99, por lo que los cálculos son exitosos.
- En el caso de la pirámide de Sierpinski, parece que siguen sin obtenerse resultados muy precisos hasta la figura 7 donde ya se aproxima bastante, y en la 8 que se obtiene la dimensión fractal exacta. En cualquier caso, vemos una mejoría en la fase 8 respecto a la 7, y por tanto esta solución es, a priori, mejor.
- Para el caso de los terrenos, también hay una alineación perfecta de los puntos obtenidos por el Box Counting y se puede apreciar como va disminuyendo la dimensión fractal conforme el terreno es menos rugoso.
- Para la experimentación que se ha planteado, de ajustar el algoritmo en base al análisis de los resultados de estos modelos, los resultados de esta fase son prácticamente los mejores posibles. Se da entonces por concluida la etapa de ajuste del algoritmo, siendo esta fase la solución final.

3.- Experimentación y validación de la solución obtenida

3.1.- Independencia de los resultados frente a el tamaño de la figura

Para que el algoritmo sea válido, es necesario confirmar que los cálculos del Box Counting no se ven afectados por el tamaño de la figura analizada. Si hubiera variación en los resultados para una misma figura con diferentes tamaños, significa que el algoritmo realmente no es fiable, ya que el tamaño de un objeto es independiente de su dimensión fractal.

Aquí vamos a comparar algunos resultados del tetraedro de Sierpinski con tamaños diferentes para el mismo modelo:

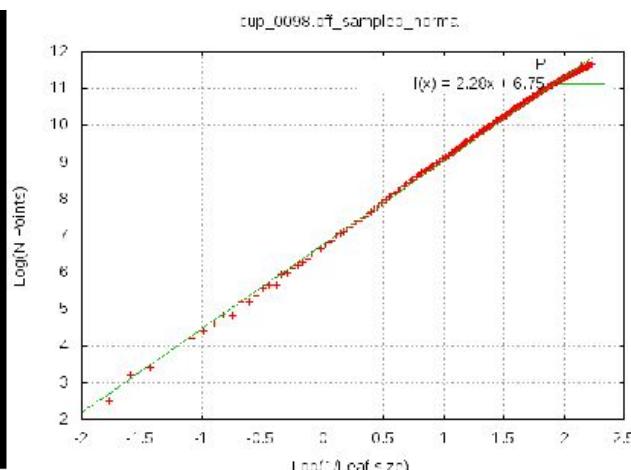
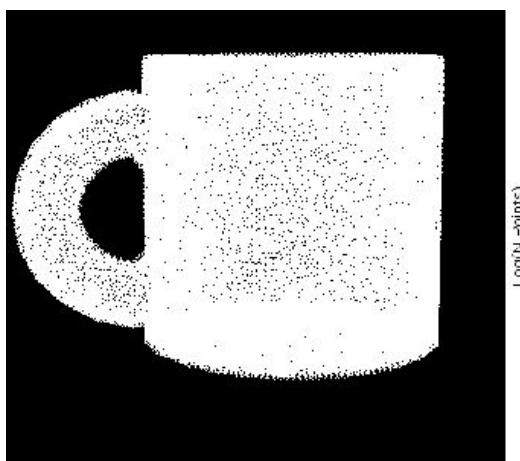
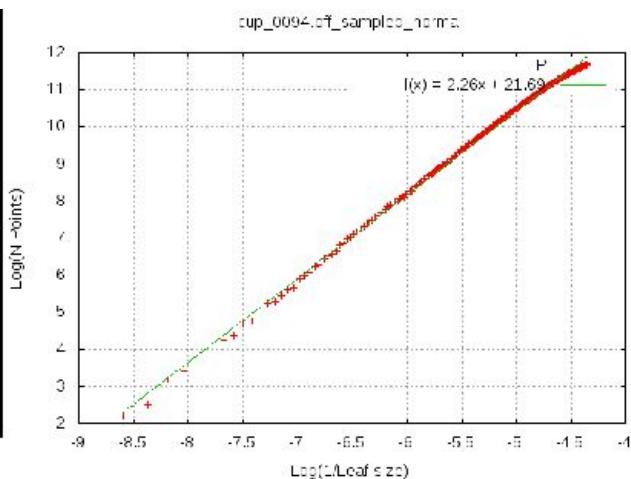
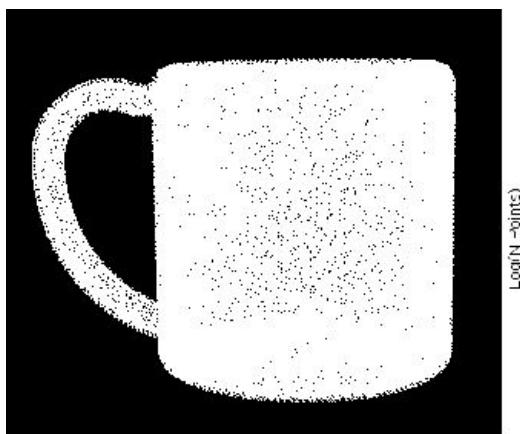


Efectivamente, vemos los mismos resultados para una misma figura con diferente tamaño, excepto en los últimos decimales.

3.2.- Experimentación con modelos reales

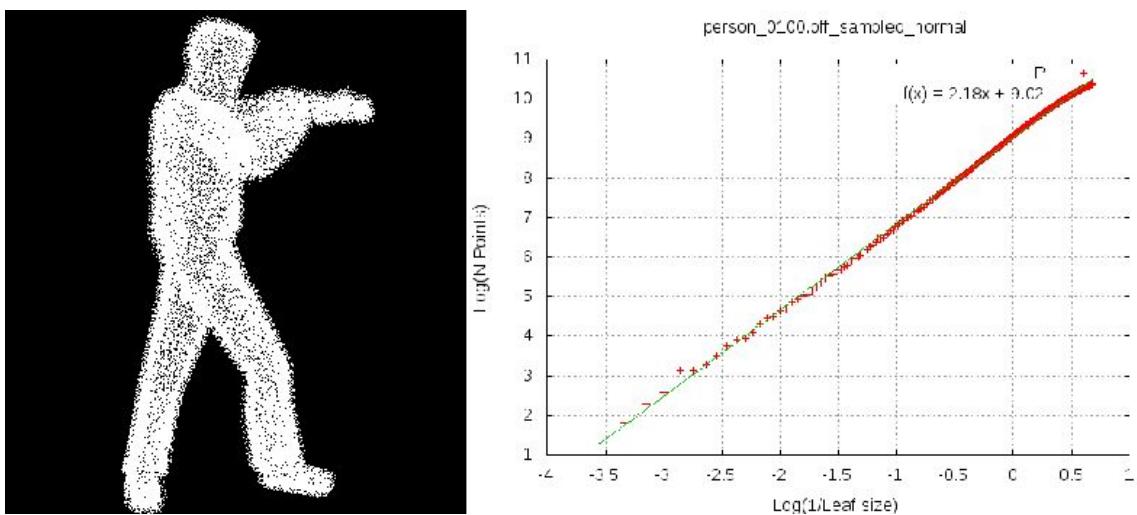
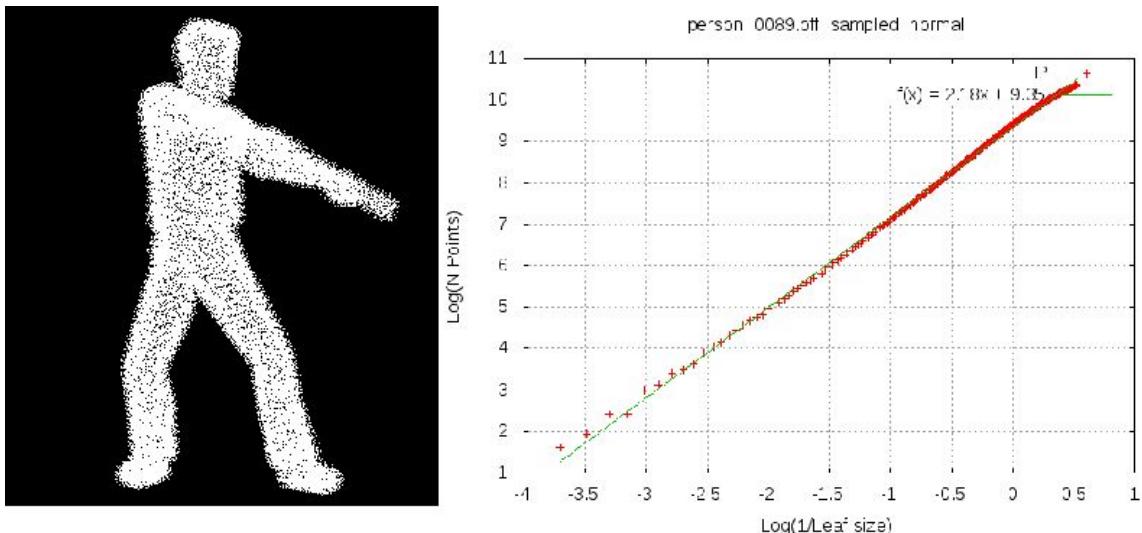
Ahora se va a poner a prueba el algoritmo procesando modelos reales, pertenecientes al dataset . Lo que tenemos que observar es si, con figuras reales, también obtenemos gráficas donde los puntos se alinean de la misma manera que para los modelos matemáticos y los terrenos con los que ya se ha realizado experimentación exitosa, y que los resultados muestran coherencia según la forma analizada.

A continuación se van a mostrar modelos de tazas y la gráfica obtenida para los resultados del algoritmo en cada uno de ellos:



Vemos que las gráficas son muy buenas en cuanto a la alineación de los puntos, y también vemos coherencia con los resultados. La segunda taza tiene el asa más gruesa y obtiene una dimensión fractal mayor que para la primera. Esto es correcto debido a que la dimensión fractal nos dice en qué medida el objeto llena el espacio que ocupa.

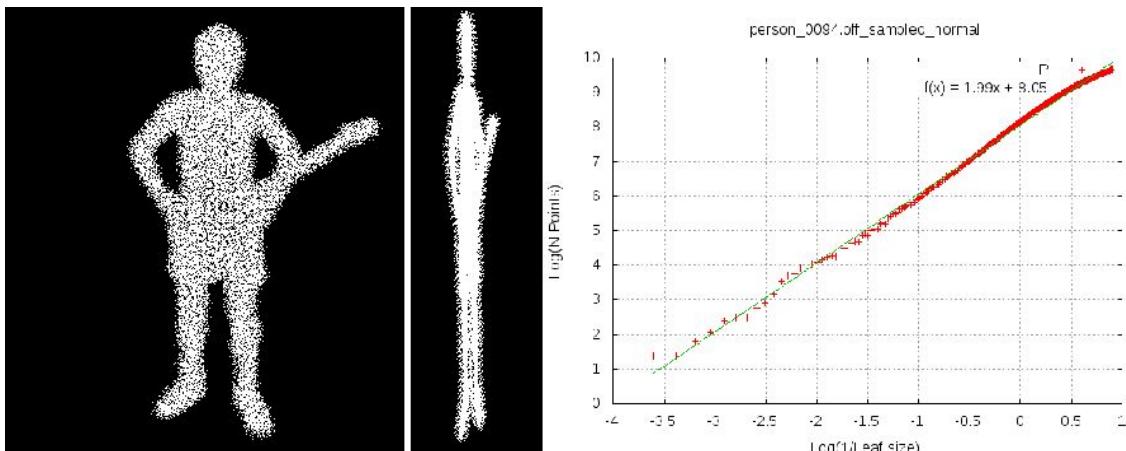
Aquí se muestran gráficas para modelos de personas humanas:



En estos dos modelos podemos ver un hombre sosteniendo un arma similar a una escopeta o un rifle. En el primero lo sostiene apuntando con un ángulo inclinado hacia abajo, y en el otro lo mantiene horizontalmente. La dimensión fractal obtenida es 2.18 para los dos.

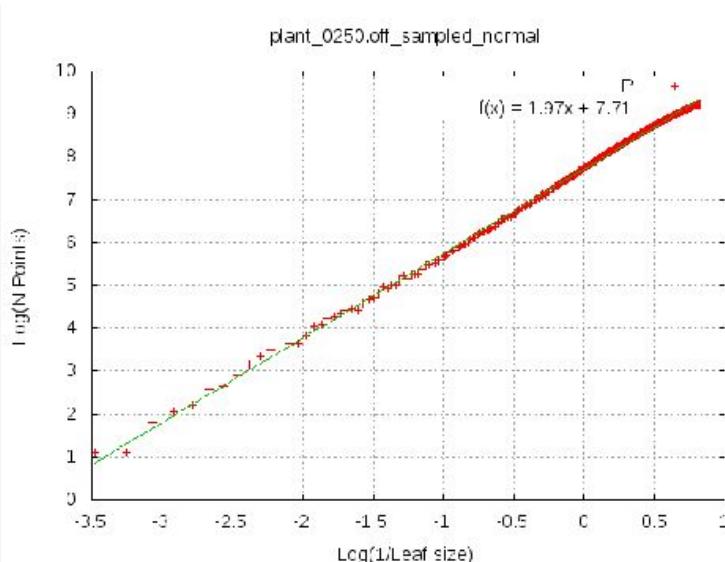
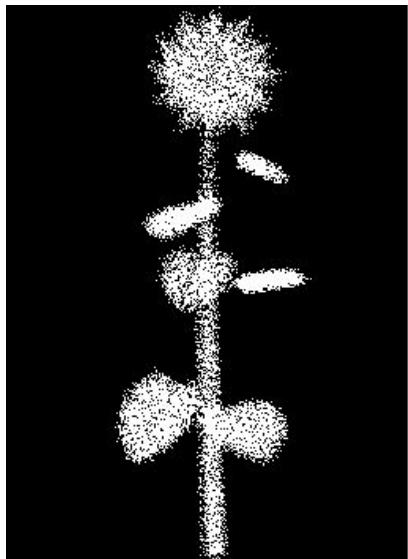
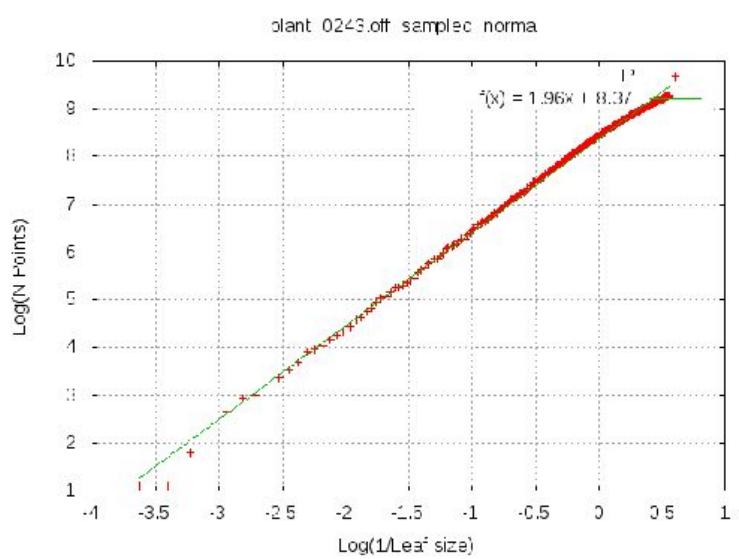
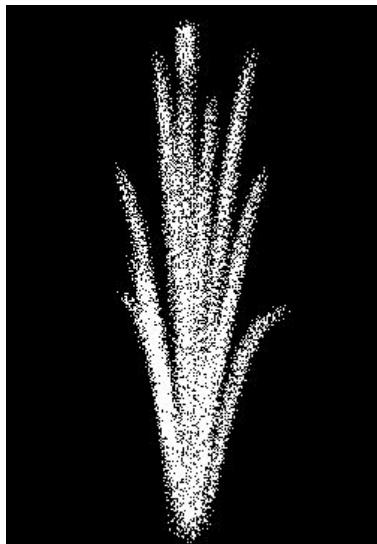
Esto son muy buenas noticias ya que este es el funcionamiento correcto.

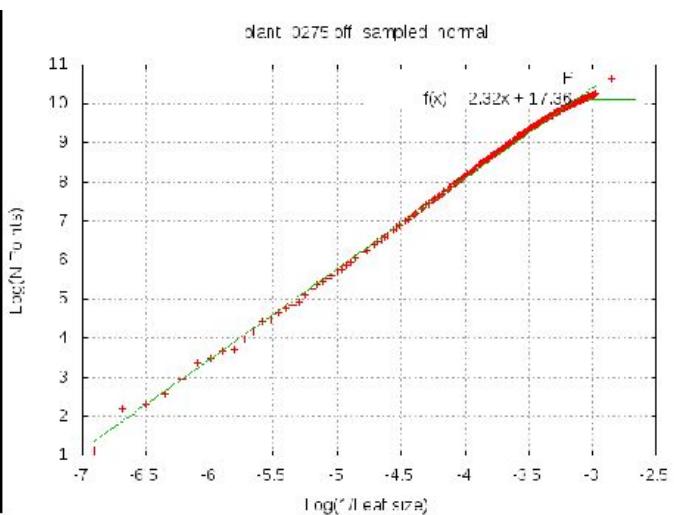
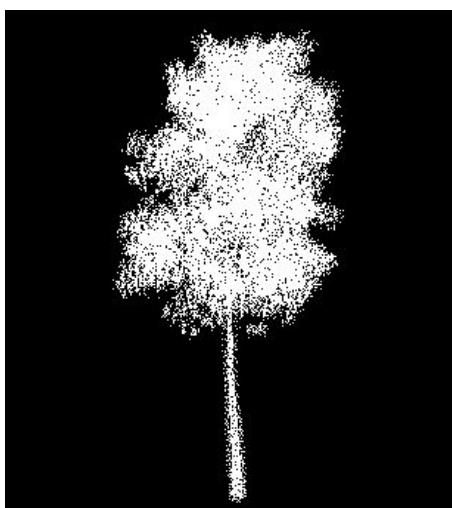
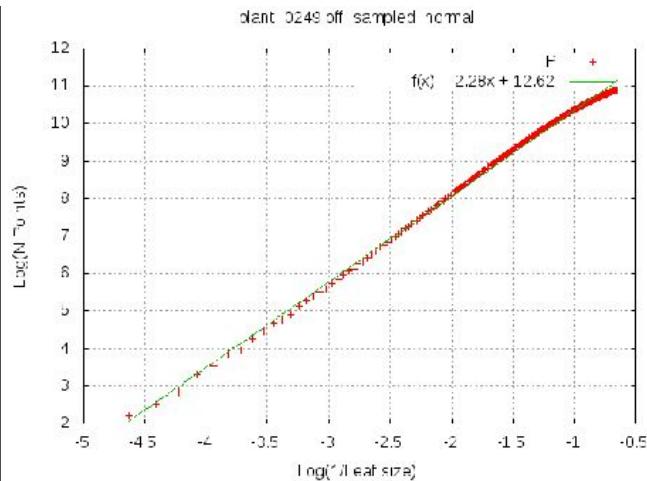
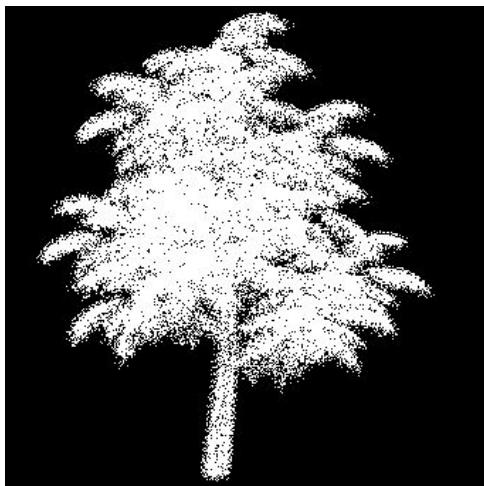
A continuación se muestran los resultados para una persona que se aproxima a un modelo 2D:



En ésta última tenemos el modelo de un hombre sujetando una guitarra, pero este no tiene profundidad. Vemos como disminuye drásticamente la dimensión fractal respecto a los anteriores modelos de personas, acercándose a la dimensión 2, que se corresponde con un plano. Esto es muy buena señal.

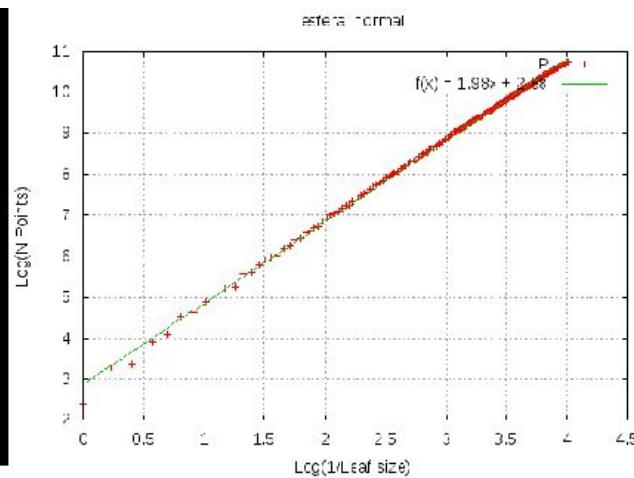
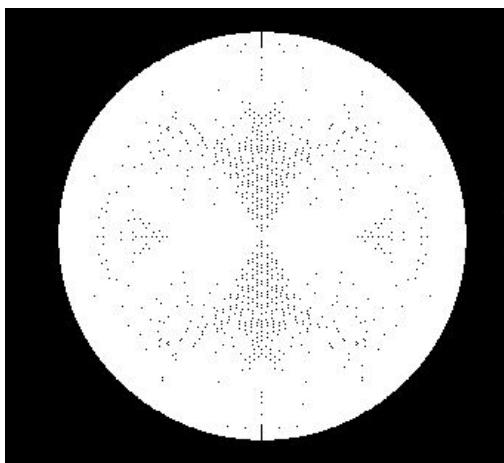
A continuación, resultados para 4 modelos distintos de plantas:





En este caso vemos que los resultados también son coherentes. Aunque visualizar modelos 3D por medio de una foto 2D puede ser un poco engañoso, si que se puede ver una clara diferencia entre los dos primeros modelos y los otros dos, y en sus correspondientes resultados.

Por último, se muestra el cálculo de dimensión sobre una esfera hueca:



Al poder crear una biyección entre la esfera hueca y el plano, son homeomorfos y tienen la misma dimensión, 2. El resultado obtenido es 1.98, aunque no exacto, muy aproximado, por tanto esto también demuestra un buen funcionamiento del algoritmo. De hecho, el error en los cálculos también podría deberse a la propia representación de la esfera, ya que esta no tiene superficie real cerrada y perfecta sino que es una nube de puntos y por tanto, la distribución de estos puntos podría afectar.

3.3.- Tiempos de ejecución

Para tener una idea de los tiempos de ejecución, se muestra a continuación una tabla con los tiempos obtenidos para varios modelos de personas.

El tiempo de ejecución incluye lectura de la nube de puntos de disco a memoria, iteraciones de conteo de cajas y cálculo de la recta de regresión:

Modelo	Tiempo de ejecución (ms)	Número de puntos
person_0089.pcd	2147.87	58.075
person_0090.pcd	2338.54	64.181
person_0091.pcd	2230.51	60.295
person_0092.pcd	1128.82	32.338
person_0094.pcd	1387.03	31.381
person_0095.pcd	1806.42	48.069
person_0096.pcd	1879.61	49.459
person_0097.pcd	1519.97	42.053
person_0098.pcd	1676.14	47.679
person_0099.pcd	2200.31	59.390
person_0100.pcd	2170.07	58.597
person_0102.pcd	1735.16	48.186
person_0103.pcd	1677.46	44.625
person_0104.pcd	2290.47	59.734
person_0105.pcd	2047.87	53.296

person_0106.pcd	1296.9	35.112
person_0107.pcd	1165.06	24.727
person_0108.pcd	1754.15	46.962

Capítulo 9: Conclusiones

Después de toda la experiencia, las pruebas y la comparación de resultados se han obtenido las siguientes conclusiones de manera global:

- Se ha conseguido programar un software capaz de obtener la dimensión fractal de objetos 3D representados mediante nubes de puntos, el cual parece obtener resultados bastante aproximados a los resultados teóricos.
- Este algoritmo ofrece tiempos de respuesta bastante razonables, siendo posible realizar una mayor optimización del mismo, además de la posibilidad de paralelizar los cálculos al existir independencia entre ellos. Realizando la optimización y paralelización de los cálculos podría incluso estudiarse la aplicación en sistemas de tiempo real.
- Realizando una investigación más extensa y con más recursos, se podría obtener una versión del algoritmo más eficaz y robusta. Existen muchas variaciones posibles, muchas pruebas y posibles experimentaciones que podrían servir para ello.
- En algunos casos, cuando se detectase cierto error en la regresión lineal, podrían utilizarse métodos de filtrado como RANSAC para mejorar los resultados, aunque el algoritmo siempre ofrece mejores resultados cuando por sí mismo es capaz de obtener un buen ajuste de los puntos a la recta.
- En cuanto a la aplicación del algoritmo en un contexto real, éste podría ser utilizado de forma genérica para nubes de puntos 3D las cuales no tienen por qué representar objetos, aunque este formato es usado mayormente para mapeo, análisis y manipulación de figuras captadas mediante cámaras 3D.
- Como se ha observado en distintas fases, parece que la densidad de las nubes de puntos puede afectar a los resultados, siendo el algoritmo poco tolerante a figuras con muy pocos vértices. Por ejemplo, el algoritmo no sería válido para calcular la dimensión de un cubo que está únicamente representado por sus vértices. Lo mismo ocurriría para cualquier objeto geométrico.
- La representación de objetos en forma de nubes de puntos 3D es una representación discontinua, por tanto en este formato no tenemos un objeto definido como un todo, y esto puede presentar algunas dificultades.

Bibliografía

La geometría fractal de la naturaleza, Benoit Mandelbrot:

<https://archive.org/stream/pdfy-lAHD8SOkMGrTE8g6/138416567-Mandelbrot-Benoit-La-Geometria-Fractal-de-La-Naturaleza#page/n5/mode/2up>

Desarrollo de la geometría fractal:

http://www.dma.fi.upm.es/recursos/aplicaciones/geometria_fractal/proyectos/movimiento_browniano/geometriafractal.htm

Fractales, conceptos, conjuntos y aplicaciones:

<https://es.wikipedia.org/wiki/Fractal>

Geometría fractal:

http://casanchi.com/mat/03_gfractal01.pdf

Teoría de fractales, aplicación en la ciencia:

<http://eprints.uanl.mx/377/1/1020114994.PDF>

Algoritmo RANSAC:

<http://slideplayer.es/slide/1737326/>

La dimensión fractal:

http://catarina.udlap.mx/u_dl_a/tales/documentos/lme/ojeda_s_r/capitulo4.pdf

Longitud y área de curvas fractales. Dimensión fractal:

<http://www.dma.ulpgc.es/profesores/personal/aph/ficheros/resolver/ficheros/fractales.pdf>

Fractales, la frontera entre el arte y las matemáticas:

http://matema.ujaen.es/jnavas/web_recursos/archivos/fractales%20datos/Modulo%205-fractales_%20jnavas.pdf

Música y análisis fractal:

http://noticiasdelaciencia.com/not/3892/estructura-fractal-oculta-en-la-musica/Informática_y_fractales

Relación con la informática:

<http://www.oni.escuelas.edu.ar/2002/neuquen/geometria/Inform%EAtica.htm>

Líneas costeras y fractales:

<http://www.madrimasd.org/blogs/universo/2007/07/07/69526>

La costa de gran breaña:

<https://vonnewmannmachine.wordpress.com/2012/11/28/la-costa-de-gran-bretana/>

Caos, fractales y cosas raras:

<http://bibliotecadigital.ilce.edu.mx/sites/ciencia/volumen3/ciencia3/150/htm/caos.htm>

Ejemplos de fractales en la naturaleza:

<http://es.gizmodo.com/diez-bellisimos-ejemplos-de-fractales-en-la-naturaleza-1677114869>

69

Sierpinski:

<http://paulbourke.net/fractals/gasket/>

Box counting:

http://www.wahl.org/fe/HTML_version/link/FE4W/c4.htm

Point Cloud:

<http://pointcloudlibrary.blogspot.com.es/2013/12/herramientas-de-la-libreria-pcl.html>

Programación con Qt4:

https://es.wikibooks.org/wiki/Programaci%C3%B3n_con_Qt4