Phraktalus

Generación de terrenos utilizando técnicas fractales

López, Gabriel Orlando
Rossi, Matías Javier
{glopez,marossi}@alumnos.exa.unicen.edu.ar

Profesor a cargo: Mg. Virginia Cifuentes

Facultad de Ciencias Exactas
Universidad Nacional del Centro de la Provincia de Buenos Aires

Resumen. En el presente trabajo resolvemos el problema de agregar mayor detalle a muestras discretas de información geoespacial utilizadas en aplicaciones de Sistemas de Información Geográfica (GIS), generando artificialmente muestras de mayor resolución debido a que se agregaron cotas artificiales a la muestra original. Básicamente, la idea es automatizar un proceso para la generación de terrenos utilizando técnicas fractales. Primeramente definimos el concepto de terreno y estructuras de datos requeridas para su representación. Luego, estudiamos las bases teóricas de los fractales y sus aplicaciones computacionales. En este punto, establecemos las pautas, fundamentos y limitaciones en el tratamiento de tales procesos. Para la producción de una herramienta capaz de realizar lo descripto, abordamos la utilización de diversos algoritmos de aplicación geométrica, detallando el análisis de eficiencia computacional, conforme avanzamos en las diferentes etapas del procesamiento de la información. Finalmente incluimos una breve descripción del sistema desarrollado, junto con demostraciones de su uso y los resultados obtenidos.

1. Conceptos generales

1.1. Terrenos

Si bien todos tenemos una noción empírica del concepto de terreno, a fines de poder discretizar el dominio y establecer las particularidades relevantes a un proceso computacional, es necesario formalizar el conjunto de variables que determinan una instancia de terreno.

En primer lugar consideramos a un terreno como un campo escalar de alturas: cada par ordenado posicional correspondiente al plano horizontal se asocia a una cota medida mediante la utilización de algún tipo de instrumento propio del estudio de la topología del planeta. A fines de representar esta información en un modelo de datos digital nos encontramos con la existencia de los comúnmente denominados modelos de elevación digital *(DEM)*. Tales campos escalares albergan los datos correspondientes a la altura (u otras variables, como temperatura,

presión, humedad, etcétera) tomados a intervalos regulares en un espacio finito y discreto. En muchos de los formatos utilizados globalmente, se incluye la información georreferencial, es decir la ubicación respecto a los orígenes de coordenadas, convencionalmente utilizados (latitud y longitud) junto con la precisión de la muestra representada.

Si bien es posible visualizar una representación fidedigna de la muestra almacenada en una de estas estructuras, es claro que como toda muestra discreta el nivel de detalle que puede obtenerse de la misma es limitado. A partir de este análisis, encontramos un interesante objetivo: generar la ilusión del incremento de la resolución de una muestra, reconstruyendo los datos no incluidos en la medición original.

Formatos de Modelos de Elevación Existen varios formatos de archivo digitales en los que se puede almacenar información de la elevación terrestre. Para la realización de este proyecto decidimos trabajar en particular con dos técnicas de almacenamiento. La primera de ellas, específica de las mediciones geoespaciales, consiste en tomar una matriz de alturas georreferenciada, es decir asociada a una región en particular del planeta y almacenar la matriz en formato binario crudo en un archivo. La segunda es una técnica general que no solo se aplica a modelos de elevación y consiste en aproximar las alturas de una superficie en función de la cantidad de luz reflejada en una fotografía.

Modelos de Elevación Digital Las agencias espaciales NASA y NGA (EE.UU.) lanzaron un proyecto denominado Misión Topográfica Radar Shuttle *(o SRTM, por sus siglas en inglés)*. La misión tiene como objetivo realizar un modelo digital de elevación de todo el globo entre los 56° S y los 60° N, desprendiéndose de aquí una gran base de datos de modelos de elevación en formatos digitales. Uno de los formatos más difundidos es el HGT *(por height, altura en inglés)*. Debido a que este formato soporta regiones cuadradas, decidimos especializar la aplicación en porciones de terrenos cuadrados, y de esta manera, evitar situaciones de posibles inconsistencias de escalas ante un uso incorrecto de sus funciones.

Uno de los problemas que acarrea la cualidad de ser georreferenciados, es la necesidad de utilizar algún tipo de proyección para poder transformar una porción de corteza esférica (suposición que no es del todo correcta, ya que la tierra es un geoide, pero dadas las condiciones es prácticamente despreciable la diferencia) en una región plana cuadrada. Al analizar diferentes opciones de proyecciones, decidimos hacer uso de una simplificación de la proyección Mercator.

En nuestro caso, decidimos simplificar el cálculo mediante el abuso de algunas características de los archivos: en primer lugar la cantidad de muestras, es decir la matriz, es perfectamente cuadrada; por otra parte, si bien son georreferenciados, la precisión de las posiciones de los archivos en el planeta es de grados enteros signados, mientras que sabemos que en la práctica, muchas muestras no han sido obtenidas exactamente al comienzo del meridiano/paralelo entero, lo que implica un posible error de $\pm$ 0° 30'.

Ante estas situaciones, decidimos optar por una solución simple y efectiva: proyectar cada grado de latitud y longitud usando como referencia la distancia

en metros, recorrida por un grado de longitud sobre el ecuador. De esta manera cada modelo HGT es transformado en un cuadrado de aproximadamente 111km de lado.

Otro aspecto importante de estas muestras es la existencia de valores desconocidos o "voids". Esto resulta un gran problema a la hora de utilizar la muestra regular, dado que los algoritmos fueron pensados para muestras regulares. En muchos software se utilizan interpolaciones bastante avanzadas a fin de corregir estos agujeros, siendo la interpolación GRASS GIS una de las más conocidas. Más adelante expondremos como resolvimos este problema.

Imágenes por luminosidad Para reconocer una imagen de dimensiones arbitrarias decidimos ofrecer la opción al usuario de estirarla, rellenarla con valores límite, o recortarla por su lado mayor. De esta manera siempre disponemos de una imagen cuadrada para hacer uso de ella como mapa de alturas.

Uno de los mayores inconvenientes al tratar con imágenes es la falta de información de referencias espaciales. Por ejemplo, es imposible determinar que altura representa el valor blanco de la imagen, y cual el negro. Además, si bien existen fotos satelitales que permiten visualizar terrenos reales, no existe una convención de nombres ni algún tipo de descriptor de archivo con detalles de las referencias terrestres, lo que impide por su parte determinar una proyección adecuada junto con un escalado proporcional a la realidad.

Para simplificar, solicitamos al usuario los parámetros relevantes, siendo el largo determinado automáticamente en 1° x 1°.

1.2. Fractales

El estudio de los fractales fue iniciado en 1975 por el matemático polaco Benoît Mandelbrot y si bien aun hoy en día, en el 2009, no se cuenta con una descripción formal precisa que cuente con aceptación general, se puede describir a un fractal como un objeto semi geométrico cuya estructura básica, fragmentada o irregular, se repite a diferentes escalas. Se le atribuyen las siguientes características:

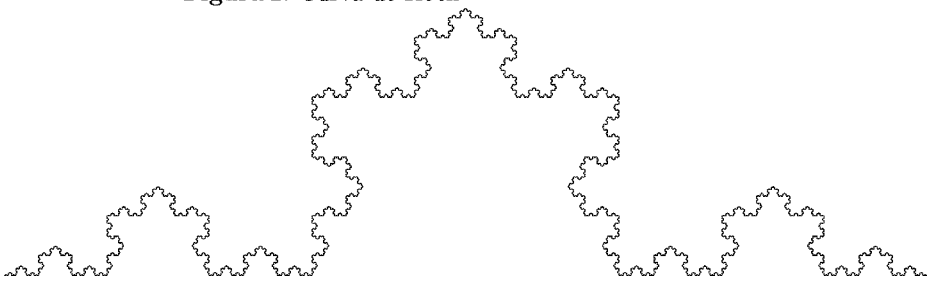
- Es demasiado irregular para ser descrito en términos geométricos tradicionales.
- Posee detalle a cualquier escala de observación.
- Posee autosimilitud (exacta, aproximada o estadística), esto es, sus partes son similares a la totalidad del objeto.
- Se pueden definir mediante un simple algoritmo recursivo.

Existen diferentes patrones fractales que fueron definidos en los últimos años y resultan relevantes a muchas disciplinas, tanto científicas como artísticas, con aplicaciones desde el estudio de la poesía a la compresión de datos. En lo relevante a nuestro proyecto, muchos elementos de la naturaleza como nubes, montañas o las líneas de costa pueden ser descritas mediante la geometría fractal.

Un ejemplo simple de geometría fractal es el caso de los *paisajes fractales*, término que fue usado por primera vez para referirse a una versión estocástica

de la *curva de Koch* que describe una línea de costa, hoy por hoy se utiliza de manera genérica para varios procedimientos fractales que pueden ser utilizados para crear datos de terreno.

Figura 1. Curva de Koch



1.3. Fundamentos de geometría y visualización computacional

Desde los estudios realizados en las ramas de *Geometría Computacional* y *Visualización de Datos*, se construyeron conceptos que es necesario remarcar, por su fuerte vinculación con este proyecto. Este es el caso de la representación mediante datos discretos de un objeto geométrico definido en el espacio tridimensional continuo, el cual concibe una abstracción del espacio real en el que estamos inmersos. Para lograr esto es necesario fracturar las figuras complejas que podemos observar a nuestro derredor, utilizando polígonos simples como componentes atómicos constituyentes del modelo de la realidad. Para materializar esto, en la práctica se utilizan figuras y cuerpos convexos denominados *símplex.*

Construcción de mallas a partir de puntos. Triangulaciones. En el caso particular de la nube de puntos, una triangulación representa una división del cierre convexo (el polígono mínimo que contiene a todos los vértices) de los puntos en símplex de forma tal que, dos símplex se intersecan a lo sumo en una arista en común y todos los vértices de cada triángulo pertenecen a la nube original. Posteriormente analizaremos diferentes técnicas de triangulación.

2. Desarrollo de los procesos

Una vez analizados los conceptos de los que partimos, estamos en condiciones de establecer con mayor precisión cuales serán los procesos requeridos y las tareas a automatizar. Nos encontramos finalmente con la existencia de modelos de elevación, los cuales describen de una forma abstracta un caso particular de terreno, sus limitaciones en lo referente a nivel de detalle, y la posibilidad de

aplicar técnicas propias de la geometría fractal a la incorporación de resolución en estos modelos. Asimismo, contamos aún con la posibilidad de construir terrenos sin información inicial, mediante la iteración de procesos aleatorios, utilizando los mismos conceptos de fractales. Teniendo en cuenta lo desarrollado previamente, en lo que refiere a la visualización de superficies, y considerando la necesidad de utilizar muestras reales de relieves, resulta conveniente contar con algún proceso de transformación de un mapa de alturas en una superficie, construida como una malla de símplex. De esta manera tanto el problema de generar terrenos, como el de incrementar el nivel de detalle quedan resumidos a la confección de mapas de alturas, los cuales serán posteriormente transformados en superficies (mallas) y finalmente renderizados en un contexto tridimensional adecuado.

2.1. Construcción de mapas de alturas

El algoritmo seleccionado para construir este tipo de terrenos se basa en la técnica de "divide y conquista". Se divide a un terreno cuadrado en cuatro subcuadrantes y se desplaza el punto central compartido mediante algún criterio. El proceso se aplica recursivamente en cada cuadrado hasta que se alcanza el nivel de detalle deseado.

El tipo de terreno generado depende principalmente del método de desplazamiento del punto medio que se utilice. En nuestro proyecto utilizamos con este fin el generador de números pseudoaleatorios provisto por la plataforma.

En la implementación realizada partimos de solo cuatro puntos localizados en las esquinas de la grilla con valores de alturas completamente aleatorios, los cuales se toman como muestra para comenzar el algoritmo. Dadas dos esquinas que compartan una arista se inserta en su punto medio un nuevo vértice con una elevación resultante del punto medio desviado de las dos esquinas. Esta misma función se aplica ahora entre el nuevo punto y los vértices originales. Continuando recursivamente este proceso se completan los laterales del cuadrado consiguiendo así, el caso base para el verdadero algoritmo.

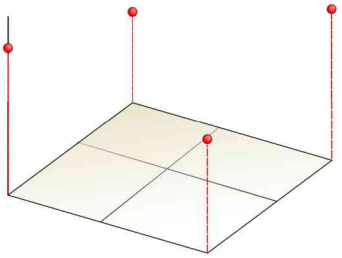
Una vez que se cuenta con un cuadrado bien definido se introduce en el centro un nuevo vértice con el valor promedio alterado de los puntos medios de cada lado. Luego se completan las líneas desde centro a la mitad de cada cara dividiendo el espacio en cuatro nuevos cuadrados, a los que se le aplicará el mismo proceso recursivamente hasta que no queden huecos en la grilla.

Cabe aclarar que este algoritmo se aplica sobre la nube de puntos de una muestra y no sobre la malla de triángulos.

A continuación se presenta el pseudocódigo del algoritmo de insersión de puntos:

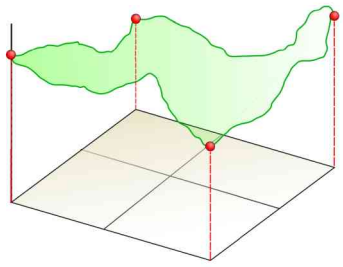
```
CompletarLinea(Grilla[N][N] G, vértice V1, vértice V2){
    vértice V = nuevo vértice (V1.altura + V2.altura )/2 + error
    Agregar V a G en posicion [(V1.posX+V2.posX)/2][(V1.posY+V2.posY)/2]
    CompletarLinea(G,V1,V)
    CompletarLinea(G,V,V2)
}

CompletarCuadrado(Grilla[N][N] G, vértice V1, vértice V2,vértice V3, vértice V4){
    vértice V12 = obtener vértice de G en [(V1.posX+V2.posX)/2][(V1.posY+V2.posY)/2]
    vértice V23= obtener vértice de G en [(V2.posX+V3.posX)/2][(V2.posY+V3.posY)/2]
```

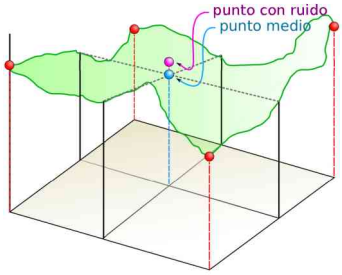
Figura 2. Explicación detallada del algorítmo de generación de terrenos



A partir de los cuatro puntos iniciales, que proyectados sobre el plano xy forman un cuadrado, se intenta dividir la región agregando el punto medio del mismo.

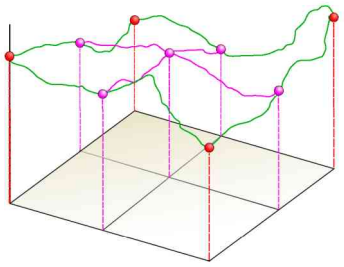
Para seleccionar el valor adecuado del punto medio, es necesario contar con una proyección de los lados del cuadrado inicial. Tomando como punto de partida los cuatro vértices iniciales, se intercalan una cantidad finita de puntos para formar las aristas del cuadrado, de modo de poder obtener el valor de altura proyectado en el punto medio.
Estos puntos, al igual que todos son insertados con ruido.





Se proyectan los puntos medios de cada lado del cuadrado, se le agrega una desviación o ruido y finalmente se almacena el valor del nuevo vértice.

A partir de este resultado se repite el proceso para cada cuadrante, hasta lograr la resolución requerida.



```
    vértice V34= obtener vértice de G en [(V3.posX+V4.posX)/2][(V3.posY+V4.posY)/2]
    vértice V41 = obtener vértice de G en [(V4.posX+V1.posX)/2][(V4.posY+V1.posY)/2]
    vértice V = nuevo vértice (V12.altura + V23.altura +V34.altura+V41.altura )/4 + error
    Agregar V a G en posicion [(V12.posX+V34.posX)/2][(V12.posY+V34.posY)/2]
    CompletarLinea(G,V12,V)
    CompletarLinea(G,V23,V)
    CompletarLinea(G,V34,V)
    CompletarLinea(G,V41,V)
}
```

Dada la simpleza de los cálculos necesarios, la complejidad temporal de este algoritmo se limita a una complejidad equivalente a $O(n \log n)$.

Inserción de detalles a una muestra El algoritmo descrito en la sección anterior nos permite generar una muestra cualquiera a partir de cuatro puntos. En esta instancia se plantea una versión más genérica que permite crear una grilla de dimensión $M \times M$ apartir de otra de $N \times N$. Dado que el objetivo es incrementar la resolución del modelo de elevación, podemos considerar sin perder generalidad que M deberá ser mayor a N. En este caso, se distribuyen los puntos de la muestra original cada intervalos regulares de M/N, y se aplica el algoritmo anterior para completar los cuadrados definidos por estos. Inspirados por la idea fractal de la autosemejanza, intentamos utilizar al modelo de elevación original como fuente de ruido para insertar error a las subregiones del mismo, sin embargo, por la necesidad de que el nivel de ruido en un punto esté limitado no es posible apreciar de manera notoria la repetición de patrones de relieve en los submodelos construidos.

2.2. Rellenado de huecos

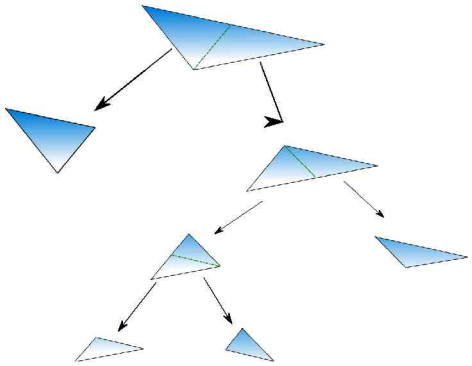
Para rellenar los huecos existentes en los modelos de elevación, utilizamos una interpolación basada en el cálculo del valor medio. Al momento de construir las estructuras de datos en memoria, se computa para cada valor nulo (void) el punto medio de sus vecinos más cercanos, utilizándose luego este valor en lugar del desconocido. Si bien es una solución que no resulta efectiva en casos patológicos o de muestras ralas, su simple implementación permite obtener modelos completos con un muy bajo costo de procesamiento.

2.3. Algoritmo de triangulación: BinTree

Tras una larga investigación, y aún una más larga discusión en las que analizamos las múltiples ventajas y desventajas de cada alternativa existente, optamos por implementar una triangulación basada en árboles binarios que consideramos la opción que mejor se adapta a nuestro proyecto. La idea básica de este algoritmo es modelar la superficie como sucesivas particiones del espacio en triángulos semejantes. Definimos así, a un nodo del árbol, como un triángulo isósceles que representa el recubrimiento convexo de un subconjunto de puntos de la grilla, que dependiendo de la diferencia de alturas existente entre los vértices contenidos se deberá dividir al subespacio delimitado por este, esto es, crear

dos nuevos nodos que serán agregados al árbol como sus hijos. Las sucesivas particiones de los triángulos se deben llevar a cabo siempre igual, incorporando una arista desde la mitad de la base hasta el vértice opuesto a esta, de esta forma los triángulos hijos resultan ser tambien isósceles, difiriendo del padre solo en la orientación. Queda claro que los nodos intermedios solo sirven para guiar el proceso y la triangulación final corresponde a la unión de las hojas del árbol, por razones que se verán a continuación, resulta necesario conocer los triángulos adyacentes a cada nodo, por lo que las hojas conforman un grafo de triángulos. Al partir de un conjunto de puntos ordenados en una grilla cuadrada el recubrimiento convexo queda definido por las cuatro esquinas de esta, al cual dividimos arbitrariamente por una de sus diagonales, obteniendo así dos triángulos isósceles que serán las raíces de nuestros dos árboles binarios.

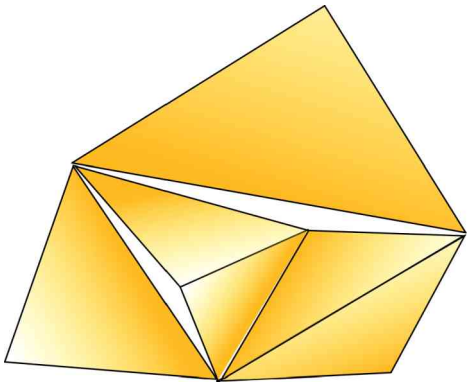
Figura 3. Ejemplo de BinTree



Si bien el árbol es una estructura naturalmente recursiva, se decidió manejar la partición de los nodos iterativamente utilizando una estructura auxiliar FIFO, no solo para evitar el overhead inherente recursión, sino que de esta forma se puede hacer fácilmente un recorrido en anchura de los nodos, lo cual implica que los triángulos más grandes son evaluados primero. Si bien quizás esto no parezca relevante a nuestro proyecto, ya que no se realiza la triangulación en tiempo real, y en definitiva se evaluarán todos los triángulos en la fila antes de empezar a mostrar el resultado, fue pensando para establecer una cota límite de triángulos a generar, así al frenar el proceso cuando se generó la máxima cantidad permitida, se minimizaría la pérdida de detalle sin necesidad de agregar un algoritmo mas complejo de selección de triángulos. Aún así, decidimos no conservar esa limitación ya que la aplicación permite al usuario seleccionar la calidad de la modelo por otros medios. Al fraccionar un triángulo para incluir un vértice más a la malla, se debe analizar si el triángulo adyacente a su base al nodo en cuestión se encuentra dividido, ya que si no es así, se generan grietas en la superficie. Para solucionar esto se debe propagar la partición de triángulos a sus vecinos base mientras sea necesario. Como consecuencia esta limitación se

desprende el hecho que en la malla, un triángulo puede diferir de sus vecinos como mucho en un nivel de profundidad del árbol.

Figura 4. Cuando no se considera la propagación de divisiones, se generan grietas.



Como criterio de división para los triángulos se debe establecer alguna métrica del error relativo a los vértices que este delimita. Con este fin, originalmente consideramos medir la diferencia de alturas de los puntos con respecto al plano definido por los vértices del triángulo, averiguamos la distancia de un punto a un plano mediante el cálculo de un determinante el cual es un cálculo bastante barato, pero surgieron complicaciones a la hora de establecer una cota a la distancia máxima permitida, ya que por uno u otro motivo, todo criterio probado resultó inapropiado para algún caso. Finalmente optamos por un método más sencillo, nos limitamos a comparar la diferencia entre las alturas de los vértices teniendo en cuenta el tamaño total del modelo, tanto en alto como en largo, tomando la decisión de partir el nodo si los puntos que este contiene exceden la variación media de altura del modelo.

Dada una grilla de $N \times N$ vértices, se puede ver como una matriz de $(N-1)^2$ celdas de cuatro puntos que en el peor de los casos, en el que se deberá hacer una triangulación completamente regular, serán a lo sumo divididas en dos triángulos, lo que equivale a una malla de $2\,(N-1)^2$ triángulos. Esto es aproximadamente el doble de triángulos que puntos, por lo que existe una relación lineal entre el número de puntos a analizar y el máximo de triángulos que se puede llegar a necesitar. Ahora bien, para generar una malla de N triángulos utilizando el algoritmo bintree es necesario analizar un total de $N \log(N)$ nodos en el proceso, la única operación relevante al análisis de complejidades realizada en cada iteración es verificar si un nodo debe dividirse o no. Para esto se debe calcular la diferencia de alturas entre los puntos contenidos dentro del mismo cumplan con un limite preestablecido. Dado que se cuenta con los vértices en una grilla (de acceso constante) organizados por sus coordenadas, el test de partición de un

nodo es proporcional a los puntos que este encierra, de aquí se desprende el hecho que la complejidad temporal del algoritmo es del orden de $O(N2 \log N)$. Por más que existan algoritmos más eficientes, sencillos, e incluso más intuitivos, optamos por esta alternativa ya que permite reducir el número de triángulos necesarios para representar la malla con una perdida de precisión acotada. Esto resulta favorable ya que por como planteamos la aplicación la triangulación se realizará una sola vez por modelo, por lo que una pérdida de performance en este proceso resulta conveniente a cambio de una reducción de los vértices a dibujar, de lo cual depende el tiempo de cálculo requerido para dar respuesta a las acciones del usuario, que se supone serán más frecuentes, al momento de interactuar con el modelo por ejemplo, rotándolo.

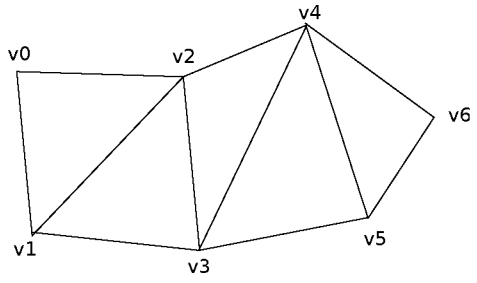
2.4. Técnicas de Renderizado

A la hora de volcar la información producida por los algoritmos en el subsistema de OpenGL, se presentaron varias alternativas, conforme a lo estudiado acerca de la interfaz de la biblioteca. En primera instancia probamos el uso de la técnica más simple de producir una salida a partir de un conjunto de polígonos: el dibujado mediante una sucesión de vértices. Este método implica transferir los datos que definen a un vértice por cada vértice de cada polígono.

En un principio esto no supuso ningún inconveniente, de hecho el sistema era completamente funcional. Sin embargo a medida que interactuamos con muestras más grandes, que se aproximaban a los tamaños de los formatos con los que deseábamos trabajar, las demoras introducidas por este proceso se volvieron inaceptables.

Para solucionar este problema optamos por utilizar el modo de representar polígonos denominado "Triangle Strip" o Tira de triángulos, que permite mediante un ordenamiento particular dibujar n triángulos utilizando $n + 2$ vértices, basado en el preconcepto de que cada triángulo comparte 2 vértices (una arista) con el último triángulo dibujado.

Figura 5. Tiras de triángulos



Entre las ventajas de esta técnica, debemos destacar, que este mecanismo permite una disminución drástica de la cantidad de datos transferidos entre las

unidades, junto con la disminución de memoria de video utilizada. Esto supone una mejora muy grande ya que en tiras muy grandes la cantidad de datos transferidos es casi tres veces menor.

Sin embargo este mecanismo tiene un costo asociado: es necesario saber en que orden dibujar los vértices, para que la biblioteca produzca los triángulos deseados. Por lo tanto es preciso contar con algún método para ordenar los triángulos de acuerdo a un criterio de vecindad, que tienda a maximizar la longitud de las tiras, y por lo tanto minimizar la cantidad de estas. Dado que encontrar una solución optima para este problema es considerado *NP-hard*, realizamos una aproximación basada en la técnica de algoritmos ávidos, ya que implementar una solución optimal implica incrementar demasiado los costos de cálculo de las tiras en lugar de favorecer la performance del proceso global, resulta en una pérdida.

3. Conclusión y Trabajos Futuros

Finalmente y para concluir, si bien logramos cumplir con los objetivos eficientemente, y estamos sumamente satisfechos con el resultado obtenido, de haber tenido mayor disponibilidad de tiempo para invertir en el proyecto, nos hubiera gustado optimizar aún más los mecanismos, aplicar diferentes criterios fractales para producir distintas variaciones de superficies, o la posibilidad de recurrir a las mismas estrategias para generar las regiones circundantes de un terreno, así como realizar un algoritmo más general para el rellenado de vacíos en las muestras.

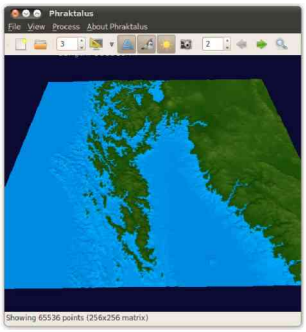
Respecto de la visualización, hubiéramos deseado poder incorporar elementos como luces más avanzadas, texturas, horizonte, así como un mejor diseño de la escena para explotar al máximo las posibilidades de la biblioteca. En lo que refiere a las texturas, hubiera sido interesante aplicar los mismas técnicas fractales para generarlas proceduralmente, y de esta manera adecuar diferentes imágenes de acuerdo a criterios como altura, pendiente, o desviación relativa al modelo.

Estas ideas quedan pendientes para una futura continuación de este proyecto.

Referencias

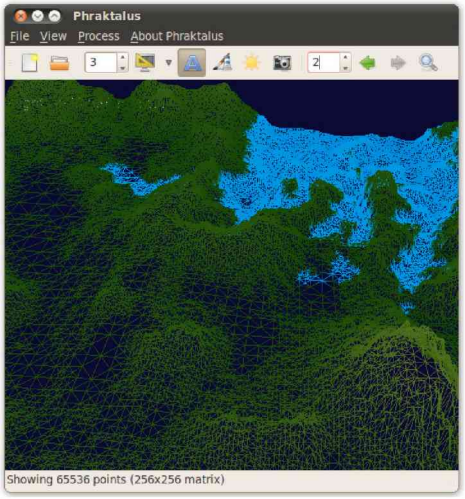
1. Cignoni, P.; C. Montani; R. Scopigno (1998). "DeWall: A fast divide and conquer Delaunay triangulation algorithm in Ed". Computer-Aided Design Volume 30 issue 5: 333–341
2. Mark de Berg. "Computational Geometry: Algorithms and Applications",3rd Edition.
3. Ismael Hidalgo Gómez. "Algoritmo para la Construcción de Grandes mallas mediante la Triangulación de Delaunay". Universidad Politécnica de Madrid, Facultad de Informática.
4. Francine Evans, Steven Skiena, and Amitabh Varshney. "Optimizing triangle strips for fast rendering". In Proc. Visualization '96, pages 319–326. IEEE Comput. Soc. Press, 1996.

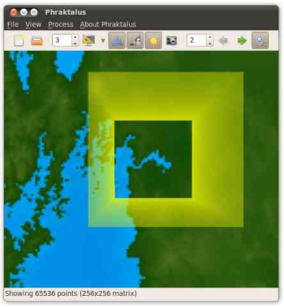
Figura 6. Resultado Final



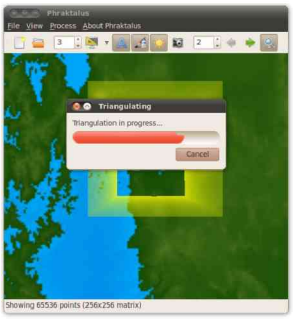
Visualización de un terreno

Visualización de la malla

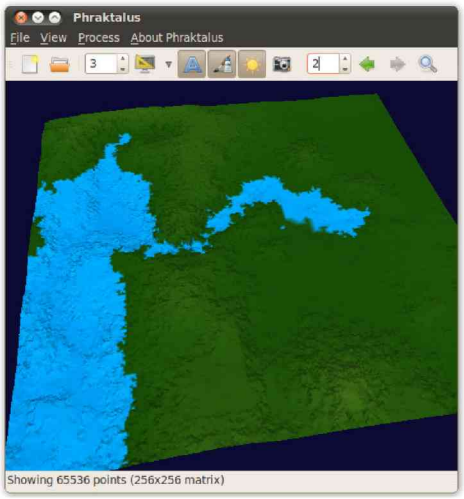




Selección de una subregión



Generación del modelo para la subregión



Terreno resultante