Universidad de las Ciencias Informáticas

Facultad 6

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Título: Modelación de capas de superficies de terrenos en tres dimensiones utilizando BDAM

Autor: José Luis Castrillón Garrido

Tutores: MSc. Gilberto Arias Naranjo

Dr. Liesner Acevedo Martínez

La Habana, Junio, 5 del 2015

Año 56 de la Revolución

Por lo general, aquello que más deseas no suele caerte encima; cae en algún sitio a tu alrededor, y tú tienes que darte cuenta de que está ahí y tienes que levantarte y que invertir el tiempo y el esfuerzo necesarios para conseguirlo. Y no es que sea así porque el universo es cruel. Las cosas funcionan así porque el universo es listo y sabe que los humanos no apreciamos las cosas que nos caen del cielo sin esfuerzo.

El método - Neil Strauss.

Declaración de Autoría

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste, firmo la presente a los ____ días del mes de _____ del año _____.

José Luis Castrillón Garrido

_____

Firma de Autor

MSc. Gilberto Arias Naranjo

_____

Firma del Tutor

Dr. Liesner Acevedo Martínez

_____

Firma del Tutor

Datos de contacto

Autor:

José Luis Castrillón Garrido

E-mail: jlcastrillon@estudiantes.uci.cu

Tutores:

MSc. Gilberto Arias Naranjo

Categoría Docente: Asistente

E-mail: gilbertoa@uci.cu

Dr. Liesner Acevedo Martínez

Categoría Docente: Profesor Auxiliar

E-mail: liesner@uci.cu

Dedicatoria

Dedico esta tesis a mis padres, a mi novia y al resto de mi familia, en especial a la memoria de mi hermano mayor Carlos Manuel.

Agradecimientos

Quisiera agradecer en primer lugar a mi madre, gracias a su apoyo y sacrificio incondicional hoy tengo la oportunidad de optar por ser ingeniero y ser el hombre que soy.

A mi novia por aportarme la musa que me ha inspirado a lo largo de este maravilloso año que llevamos juntos.

A mi padre por sus beneficiosos consejos, por compartir muchas de mis ideas y apoyarme en todo lo que hago.

A toda mi familia por el apoyo que siempre me han brindado, a mi abuela por su cariño en todo momento, a mis tíos, mis hermanos, primos y sobrinos.

A mis tutores por sus provechosas revisiones.

A todos los profesores que me apoyaron durante el desarrollo de la tesis.

A todos mis amigos.

Al movimiento de programación competitiva “Tomás López Jiménez” de la UCI.

Resumen

La visualización de terrenos en tres dimensiones juega un papel fundamental en muchas áreas de investigación científica. Un ejemplo es la visualización de simulaciones de fenómenos naturales como las crecidas de un río o el embate de un tsunami. Cuando la dimensión y resolución del mapa es grande se necesitan modelos computacionales que permitan una visualización multi-resolución del mismo, que generen la menor cantidad posible de primitivas gráficas, para hacer posible su visualización eficiente. El componente para la visualización de terrenos en tres dimensiones desarrollado en la Universidad de las Ciencias Informáticas utiliza como modelo computacional el *quadtree* restringido para la representación multi-resolución de las capas de terrenos. Sin embargo para lograr una triangulación correcta usando esta estructura, se necesita la inclusión de puntos auxiliares, lo que aumenta la cantidad de primitivas gráficas que se generan. En este trabajo se propone, el uso del modelo híbrido de Mallas Dinámicas Adaptativas por Lotes en el componente. Este modelo combina las fortalezas de los modelos jerárquicos y los modelos que generan redes irregulares de triángulos, además utiliza técnicas de simplificación de alta calidad, algoritmos eficientes para la generación de cadenas de triángulos y provee escalabilidad mediante el uso de la memoria externa para la construcción y visualización. Los resultados obtenidos evidencian una reducción significativa de la cantidad de primitivas gráficas, además de obtenerse mejoras con respecto a la calidad visual de las triangulaciones.

Palabras Claves: Cadena de Triángulos, Modelo digital de elevación, Modelo híbrido, Multi-resolución, Red Irregular de Triángulos.

Abstract

Three dimensional terrain visualization plays a key role in many areas of research such us the simulation of natural phenomena including river floods and the impact of a tsunami. Sometimes the size and resolution of a terrain may be huge, in order to achieve an efficient visualization, with the least possible number of graphic primitives, computational models that allow the multiresolution representation of the terrain are required. The component for the visualization of three dimensional terrain surfaces developed in “La Universidad de las Ciencias Informáticas” uses the restricted quadtree model for the multi-resolution representation of three dimensional terrain layers. However, this computational model needs the insertion of many auxiliary points to obtain a correct triangulation, this insertion increases the number of graphic primitives needed to represent the terrain. The present research proposes the use of the Batched Dynamic Adaptive Meshes (BDAM) model to reduce the number of generated polygons, BDAM as a hybrid model combines the benefits of hierarchical and irregular models and uses high-quality simplification and triangle strip generation algorithms, it also provides the system with scalability by using external memory management allowing the interactive representation of huge terrains. The results obtained during the present research show a significant reduction of the number of generated graphic primitives and a considerable improvement with respect to the visual quality of the generated triangle meshes.

Keywords: Triangle strip, Digital elevation model, Hybrid model, Multi-resolution, Triangulated irregular network.

Índice

Índice de Figuras

Índice de Tablas

Introducción

La visualización interactiva de varias capas de terreno en tres dimensiones (3D) de una forma eficiente y rápida es un objetivo esencial de los Sistemas de Información Geográfica (GIS). La principal ventaja de la representación 3D respecto a la representación en dos dimensiones (2D) es que se obtiene una mayor cantidad de información visual. Los GIS que incluyen la representación de terrenos en tres dimensiones han sido de gran utilidad en el apoyo a la toma de decisiones, estos abarcan desde la representación de información topográfica, hidrológica, de superficies de terreno hasta la simulación de fenómenos naturales y desastres de una manera realista.

Los Modelos Digitales de Elevación (MDEs) utilizados con el objetivo de representar los terrenos pueden alcanzar dimensiones superiores a las centenas de millones de triángulos, para poder llevar a cabo la visualización de los mismos se hace necesario filtrarlos para mostrar solamente los datos relevantes para la visualización. El enfoque más referenciado en la literatura para el filtrado consiste en la aplicación de modelos multi-resolución que permiten, en base a determinados parámetros de entrada (eje. altura y campo de visión de la cámara), adaptar la visualización del terreno de la forma más eficiente posible.

En el año 2013 se inscribe el proyecto de investigación: “Representación multicapas de información geográfica, tomando como base Modelos Digitales de Elevación, para el desarrollo y análisis de simulaciones en 3D.” suscrito al Centro de Estudios de Matemática Computacional (CEMC) de la Universidad de las Ciencias Informáticas y que se realiza en conjunto con el Centro Internacional de Métodos Numéricos en la Ingeniería (CIMNE), con sede en Barcelona, España. El objetivo fundamental del proyecto es la representación multi-resolución de varias capas de información geográfica, incluyendo aquellas que formen parte de resultados de simulaciones sobre determinado terreno. Como resultado de este proyecto se obtuvo un componente de visualización de información geográfica en tres dimensiones (en lo adelante Visor de Terrenos).

La arquitectura del Visor de Terrenos presenta tres capas (ver figura 1):

- Capa de Acceso a Datos
- Capa de Modelación
- Capa de Visualización

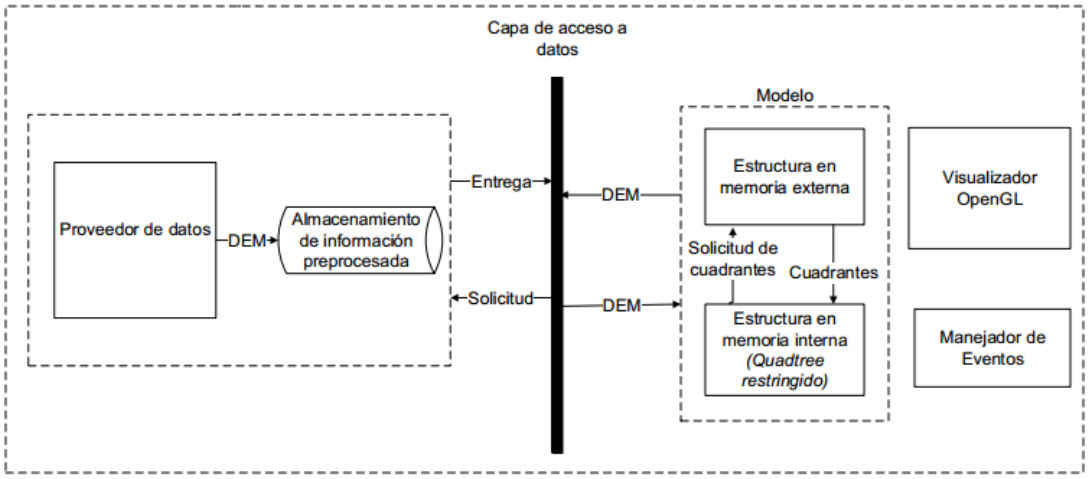

Fig. 1 Arquitectura del Visor de Terrenos (Elaboración propia)

Actualmente en la capa de modelación del Visor de Terrenos se utiliza, como estructura de datos para construir una representación multi-resolución de las distintas capas de terreno, el *quadtree* restringido. La principal desventaja en la utilización de dicha estructura es que para lograr que las triangulaciones que se construyen para la visualización sean correctas y evitar la aparición de grietas en el terreno, se necesita la inclusión de demasiados puntos auxiliares, causando esto un incremento en la cantidad de primitivas gráficas[1] a representar, lo que implica un elevado consumo de memoria por la estructura y un costo elevado en tiempo de ejecución para obtener una representación multi-resolución del terreno. En la literatura el *quadtree* restringido es clasificado como un modelo jerárquico, sin embargo existe el grupo de los modelos multi-resolución híbridos que obtienen triangulaciones correctas con menor cantidad de primitivas gráficas que los modelos jerárquicos, entre ellos se destaca el modelo Mallas Dinámicas Adaptativas por Lotes (BDAM por sus siglas en inglés).

Partiendo de lo antes expuesto se define como **problema de investigación:**

¿Cómo disminuir la cantidad de primitivas gráficas utilizadas en la representación en tres dimensiones de los MDEs en el Visor de Terrenos?

Del problema anterior se define como **objeto de estudio:** La representación gráfica multi-resolución de los MDEs en tres dimensiones.

[1] Las primitivas gráficas en un software 3D son formas geométricas simples (puntos, líneas y polígonos) que sirven como base para construir estructuras más complejas.

Se establece como **campo de acción:** Modelos multi-resolución para la representación de los MDEs en tres dimensiones.

El **objetivo** de la presente investigación es: Desarrollar funcionalidades para la representación en tres dimensiones de MDEs utilizando el modelo híbrido BDAM que permitan la disminución de la cantidad de primitivas gráficas utilizadas para la representación de los mismos en el Visor de Terrenos.

A partir del marco teórico desarrollado se formula la siguiente **hipótesis**:

Si se utiliza el modelo híbrido BDAM para la representación en tres dimensiones de MDEs en el Visor de Terrenos se reducirá la cantidad de primitivas gráficas.

Para dar cumplimiento al objetivo planteado se definieron las siguientes tareas de investigación.

- Revisión y análisis de la literatura referente a la representación multi-resolución de terrenos en 3D en tiempo real para la elaboración del marco teórico.
- Estudio de los principales modelos híbridos utilizados en la representación multi-resolución de terrenos en 3D.
- Estudio de los principales algoritmos y técnicas de optimización utilizados en el desarrollo de modelos híbridos para identificar los que más se ajusten a la presente investigación.
- Selección de las herramientas y tecnologías a utilizar en el desarrollo de la solución.
- Selección de los principios de diseño para guiar la implementación de la solución.
- Implementación de las estructuras de datos y algoritmos seleccionados para la representación multi-resolución de terrenos en 3D.
- Diseño de pruebas para la comparación del modelo híbrido con respecto al *quadtree* restringido del Visor de Terrenos.

Durante el desarrollo de la investigación fueron utilizados los siguientes métodos de la investigación científica:

- Método analítico-sintético: Se estudiaron las diferentes técnicas para la representación multi-resolución de terrenos en 3D y luego fueron sintetizadas en la construcción de un modelo multi-resolución híbrido.
- Método hipotético-deductivo: Para la formulación de la hipótesis y arribar a conclusiones.
- Método de modelado: para representar los diagramas correspondientes a las etapas de análisis, diseño e implementación del modelo.

- Análisis documental: Se realizó una revisión de la literatura sobre la representación multi-resolución de terrenos en 3D para consultar la información necesaria en el proceso de investigación.

Resultados esperados

- Incluir al Visor de Terrenos la funcionalidad de representar los MDEs en tiempo real mediante el uso de modelos multi-resolución híbridos reduciendo de esta forma la cantidad de polígonos generada.
- Aumentar la velocidad de visualización de terrenos en 3D en el Visor de Terrenos.
- Disminuir el consumo de memoria durante la visualización de los MDEs.

El presente trabajo consta de la estructura siguiente:

Capítulo 1: Fundamentación teórica

Contiene la base teórica para comprender el problema planteado. Se describen los conceptos fundamentales para el dominio del problema, así como las tecnologías a utilizar en el desarrollo del trabajo.

Capítulo 2: Análisis y Diseño

Este capítulo contiene los elementos resultantes del diseño y análisis de la solución, entre estos se encuentran el diagrama de dominio del sistema, la especificación de los requisitos, la arquitectura utilizada en la solución, el diagrama de clases, los patrones de diseño utilizados y la descripción de la presente solución.

Capítulo 3: Implementación y Pruebas

En este capítulo se expondrá la implementación de los componentes que conforman la solución así como sus principales algoritmos. Se le realizarán pruebas a la solución para validar el correcto funcionamiento de la misma. Se mostrarán además experimentos donde quede evidenciado el desempeño del modelo multi-resolución seleccionado en la representación de superficies en tiempo real.

Capítulo 1: Fundamentación Teórica

En este capítulo se muestran los principales conceptos asociados a la representación multi-resolución de terrenos en tres dimensiones así como las técnicas y algoritmos más usados con este propósito.

1.1 Conceptos asociados al dominio del problema

1.1.1 Modelo Digital de Terreno y Modelo Digital de Elevación

Un modelo digital de terreno (DTM por sus siglas en inglés) es un conjunto ordenado de puntos que representa la distribución espacial de varios tipos de información en el terreno, puede definirse también como la representación digital de la distribución espacial de información de terreno representada por localizaciones en dos dimensiones junto a una representación matemática de la información del terreno. Un modelo digital de elevación (DEM por sus siglas en inglés) es un subconjunto de DTM y se refiere a los datos de elevación organizados en forma de matriz (Li et al., 2010), a este modelo se le conoce también como Ráster.

1.1.2 Representación multi-resolución de un Modelo Digital de Elevación

El concepto de multi-resolución (Pajarola y Gobbetti, 2007) va más allá de las superficies en tres dimensiones. Un modelo multi-resolución representa varios niveles de detalle de un objeto, conocido también por sus siglas en inglés LOD (*level-of-detail*), estos modelos pueden presentar dos tipos de resolución:

- **Resolución Fija:** El terreno se representa con la misma resolución en todas sus regiones.
- **Resolución Variable:** Varias regiones del terreno presentan diferente resoluciones.

Durante la visualización de terrenos en tres dimensiones, las regiones distantes al punto de visión se proyectan en un área pequeña de la cámara, mientras que las regiones más cercanas ocupan un área mayor, esto permite la asignación de niveles de detalles a las distintas regiones del terreno tomando como base la distancia hacia el punto de vista de la cámara. La principal ventaja de estos modelos es que permiten una reducción significativa de la cantidad de polígonos para la representación del terreno lográndose de esta forma un aumento en la velocidad de visualización.

1.2 Técnicas para la representación multi-resolución de terrenos en tres dimensiones.

Los juegos de datos utilizados para la representación interactiva de terrenos en tres dimensiones pueden llegar a contener cientos de millones de puntos, lo que dificulta la visualización rápida de los mismos. Para dar solución a este problema decenas de técnicas han sido propuestas, estas pueden ser divididas en las siguientes categorías:

1- Técnicas de estructura jerárquica

2- Técnicas basadas en triangulaciones generales sin restricciones.

3- Técnicas híbridas que combinan las dos técnicas anteriores.

Los ejemplos más comunes de la primera clase son el *quadtree* restringido (Pajarola, 1998) y la jerarquía de triángulos rectángulos (Evans et al., 2001). Estos modelos tienen como idea principal la creación de una estructura jerárquica mediante simplificación o refinamiento. En el refinamiento cada nodo se divide de acuerdo a márgenes de error, incrementando de esta forma la resolución del terreno. Para la simplificación se parte de la triangulación completa del terreno y se realizan uniones de los nodos disminuyendo de esta forma la resolución del terreno.

Las técnicas de la segunda clase generan redes irregulares de triángulos (TIN por sus siglas en inglés) que no responden a una estructura jerárquica de subdivisión. Mallas Progresivas (Hoppe, 1996) es uno de los ejemplos más exitosos, donde se crean simplificaciones sucesivas del terreno usando algoritmos de simplificación de mallas triangulares.

Las redes irregulares de triángulos usan menor cantidad de polígonos para representar el terreno que las estructuras jerárquicas dado un margen de error específico, pero su implementación tiende a ser más costosa en memoria y tiempo de ejecución que las de estructuras jerárquicas (Cignoni et al., 2003). Los modelos basados en la tercera clase combinan las fortalezas de ambas técnicas mencionadas anteriormente y serán el centro de atención en la presente investigación.

1.2.1 Modelos híbridos para la representación de MDE en tres dimensiones

Los modelos híbridos han sido objeto de estudio por parte de varios autores debido a las ventajas que estos poseen con respecto al resto de las técnicas utilizadas para la visualización en tres dimensiones de superficies de terreno. A continuación se detallan los modelos híbridos que mayor relación presentan con la presente investigación.

QuadTin (Pajarola et al., 2002) recibe como entrada una TIN a partir de la cual se genera un *quadtree* restringido insertando puntos auxiliares en caso necesario. El QuadTin de Híper-Bloques (Lario et al., 2003) realiza mejoras al QuadTin en términos de costo de memoria permitiendo pre calcular distintas triangulaciones en los niveles. La principal ventaja de esta estructura radica en el uso de bloques de vértices para determinar el nivel de detalle de la triangulación a diferencia del QuadTin que utiliza los vértices individualmente.

En (Baumann et al., 1999) se introduce el concepto de árboles de aproximación que define un criterio general de subdivisión jerárquica aplicable a cualquier superficie de terreno. Cada nodo del árbol de aproximación cubre una región rectangular del terreno original y contiene una TIN que aproxima dicha región con una cantidad de triángulos predefinida. El principal reto de los árboles de aproximación radica en el tratamiento de las grietas generadas en los bordes de los nodos producto a la diferencia en los niveles de detalle.

BDAM (Cignoni et al., 2003) mantiene dos árboles estrechamente relacionados, uno para la geometría y otro para la textura. En la presente investigación solamente se abordará el árbol de la geometría dado que este es usado con el mismo propósito que el *quadtree* restringido en el Visor de Terrenos. Para la geometría se utiliza la estructura de datos espacial *bintree,* dicha estructura es un árbol binario que divide el terreno en una jerarquía de triángulos rectángulos (HRT por sus siglas en inglés). En una HRT (ver figura 2) cada nodo representa un triángulo rectángulo y los hijos son hallados al dividir el triángulo por la bisectriz correspondiente a la hipotenusa. A la unión entre dos nodos del mismo nivel por su arista más larga (hipotenusa del triángulo) se le denomina diamante (Weiss y De Floriani, 2011). Esta representación jerárquica permite la obtención de una aproximación multi-resolución de terrenos mediante refinamiento o simplificación de los nodos. El modelo BDAM tiene la particularidad de que cada nodo del bintree contiene una red irregular de triángulos. Los nodos hojas (el nivel más detallado) contienen los triángulos de la malla original. Los nodos intermedios contienen una TIN que aproxima la superficie cubierta por el nodo. Para la construcción del modelo BDAM se hace uso de algoritmos de simplificación, algoritmos para la generación de cadenas de triángulos y se utiliza la memoria externa para el procesamiento escalable de los datos. Se obtiene una aproximación multi-resolución del terreno a partir de un refinamiento *top-down* de los nodos del bintree usando como criterio el error de la proyección en la pantalla de la TIN del nodo.

En (Hao et al., 2007) se propone un modelo híbrido basado en dos niveles que divide el terreno en pequeñas regiones cuadradas llamadas parches, en cada momento se mantienen cargado en memoria una cantidad finita de parches alrededor del punto de visión de la cámara en la máxima resolución, los

parches restantes pertenecen al dominio de las redes irregulares de triángulos y representan una resolución más baja teniendo en cuenta la distancia al punto de visión. Este modelo presenta como desventaja que es sensible al movimiento rápido de la cámara provocando anomalías en los bordes de cada parche.

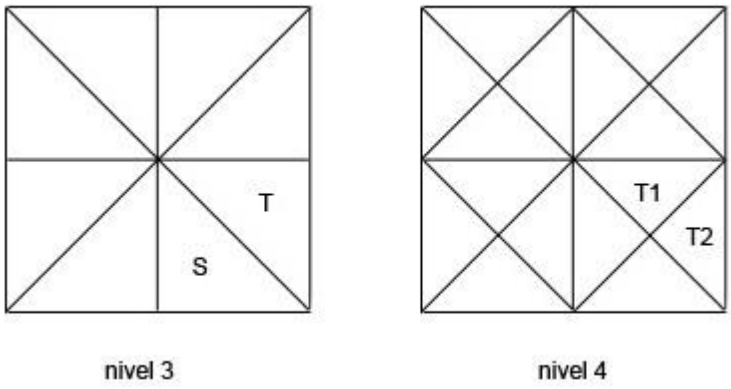

Fig. 2 Dos niveles consecutivos de una HRT: Los nodos T1 y T2 del nivel 4 son obtenidos al dividir el nodo T del nivel 3 por la bisectriz correspondiente a la hipotenusa. En el nivel 3 la unión de T y S es un diamante (Elaboración propia).

De los modelos mencionados anteriormente se escogió BDAM para realizar la modelación multi-resolución de los MDEs, dado los resultados mostrados en (Cignoni et al., 2003) este modelo presenta velocidades de visualización superiores a las cien tramas por segundo en terrenos de tamaño superior a 4k x 4k.

1.3 Algoritmos y estructuras de datos asociados al modelo BDAM.

Existe gran variedad de algoritmos y estructuras de datos utilizados durante la representación en tres dimensiones de terrenos, esta sección tiene como objetivo mostrar las técnicas más relacionadas con el modelo BDAM.

1.3.1 Algoritmos de simplificación en mallas triangulares

Las técnicas de simplificación de mallas triangulares pueden ser divididas en tres categorías según el elemento de la malla a tener en cuenta para la simplificación.

- Eliminación de vértices.
- Eliminación de aristas.

- Eliminación de triángulos.

Las técnicas correspondientes a la eliminación de vértices presentan un mejor balance entre tiempo de ejecución y calidad visual que el resto de las técnicas (Hussain, 2013), (Ng y Low, 2014) y serán el centro de atención en la presente investigación.

En el proceso de eliminación de vértices, se selecciona una de las aristas adyacentes al vértice, se colapsa y se actualiza la información resultante luego del colapsamiento. El colapsamiento tradicional crea un nuevo vértice en alguna posición que disminuya el error sobre la línea correspondiente a la arista, elimina los vértices al extremo de la arista adicionándole al nuevo vértice las aristas y triángulos correspondientes a los dos vértices eliminados.

Otra manera de realizar el colapsamiento de aristas es conocida como colapsamiento media-arista (Van Kaick y Pedrini, 2006), una media-arista es una arista dirigida que tiene un nodo origen **u** y un nodo destino **v**. Este tipo de colapsamiento tiene como ventaja que mantiene los vértices de la malla original así como las propiedades que estos contengan.

A continuación se detallan los algoritmos de simplificación de mallas triangulares más relacionados con la presente investigación. QSlim (Garland y Heckbert, 1997) es uno de los algoritmos más referenciados en la bibliografía consultada. Este algoritmo utiliza colapsamiento iterativo de las aristas manteniendo el error de aproximación de la superficie mediante matrices cuádricas. La principal desventaja de este algoritmo es el consumo de memoria requerido para almacenar las matrices para cada vértice.

El algoritmo VolSIMP (Hussain, 2013) emplea una métrica de error basada en el campo normal para la selección del vértice menos importante de la malla y colapsa la arista que menor variación de volumen genera mediante el colapsamiento media-arista. En (Ng y Low, 2014) se propone un algoritmo que utiliza las longitudes de las aristas y la distancia entre las normales de los dos triángulos adyacentes a la arista como métrica para guiar la simplificación. Tanto VolSimp como el algoritmo propuesto en (Ng y Low, 2014) basan su simplificación en la eliminación de vértices usando colapsamiento media-arista, esto les permite tener un mejor desempeño en cuanto a tiempo de ejecución que QSlim debido a que la decisión global la realizan sobre los vértices y no sobre las aristas.

La métrica más referenciada en la literatura consultada para evaluar la calidad de aproximación de una malla triangular es la distancia de Hausdorff (Cignoni et al., 1998). Esta métrica define la distancia bidireccional entre la malla original y la malla aproximada. En la presente investigación se decidió utilizar

el algoritmo VolSimp, teniendo en cuenta que el mismo presenta alta calidad con respecto a la distancia de Hausdorff (Hussain, 2013) a un bajo costo computacional.

1.3.2 Algoritmos para la generación de cadenas de triángulos

En la manera tradicional de representar mallas triangulares se necesitan **3n** vértices donde **n** es la cantidad de triángulos. Ya que las aristas son compartidas por triángulos adyacentes en una malla triangular, los vértices son enviados varias veces para ser procesados por la tarjeta gráfica.

Una cadena de triángulos es una secuencia de **n + 2** vértices que representa **n** triángulos (ver figura 3). La adyacencia de algunos grafos no permite la representación secuencial de la malla en una sola cadena y se hace necesario repetir vértices introduciendo triángulos degenerados, a esta inserción se le conoce como intercambio (Vaneek, 2005). Las cadenas de triángulos son soportadas en la mayoría de las bibliotecas gráficas (OpenGL, DirectX, etc.) y en las tarjetas de videos.

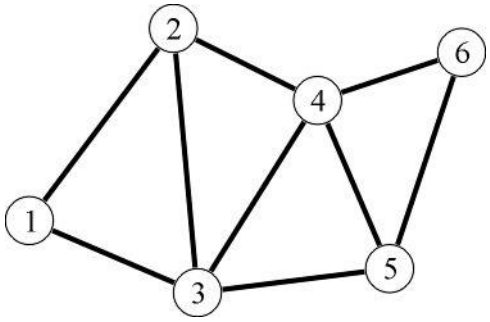

Fig. 3 Cadena de triángulos, se definen 6 vértices para representar 4 triángulos. (Elaboración propia).

Para el cálculo de la menor partición de una malla en cadenas de triángulos no existe solución polinomial (Van Kaick et al., 2004), este problema es equivalente al del camino hamiltoniano que es NP completo. Existe gran variedad de algoritmos basados en heurísticas para dar una solución aproximada a este problema.

SGI (Akeley et al., 1990) es un algoritmo ávido que construye la cadena de triángulos de la siguiente forma: sobre el grafo dual de triángulos se busca el triángulo con el menor grado de adyacencia en la red, a partir del triángulo seleccionado se busca la cadena de triángulos más larga que se puede formar comenzando en el triángulo seleccionado, esta cadena es concatenada a la cadena de salida mediante la inserción de triángulos degenerados, por su simplicidad y rapidez este algoritmo ha sido modificado y mejorado a lo largo de los años.

El algoritmo propuesto en (Xiang et al., 1999) utiliza algoritmos de búsquedas de teoría de grafos para calcular un árbol de expansión del grafo dual de la triangulación, de este árbol se obtiene un conjunto

de cadenas de triángulos mediante el uso de programación dinámica que son concatenadas luego para obtener cadenas de triángulos de mayor longitud.
(Stewart, 2001) propuso un algoritmo que utiliza un operador llamado *tunneling* para reducir la cantidad de cadenas en uno, este algoritmo produce una cantidad muy baja de cadenas pero a un alto costo en tiempo de ejecución.
En vista a los resultados mostrados en (Vaneek, 2005) se decidió utilizar SGI en la presente investigación dado que este muestra un mayor balance entre tiempo de ejecución y longitud de la cadena de triángulos generada que los demás algoritmos mencionados anteriormente.

1.3.3 Estructuras de datos en mallas triangulares

Para el manejo eficiente de la información en mallas triangulares varias estructuras de datos han sido propuestas, entre las más populares se encuentra media-arista (Botsch et al., 2002). Esta estructura crea para cada arista dos copias (media-aristas) dirigidas en direcciones opuestas, cada media-arista contiene el índice del nodo destino, el índice del triángulo a su izquierda, los índices de la siguiente media-arista y de la anterior a lo largo del borde del triángulo y el índice de la media-arista que va en dirección opuesta.

Otra estructura de datos muy común en las implementaciones de algoritmos sobre mallas triangulares es la de triángulos indexados (De Floriani et al., 2005), esta estructura a diferencia de media-arista centra su atención en los triángulos de la malla almacenando para cada uno los índices de los vértices y los índices de los triángulos adyacentes a este, cada vértice de la malla almacena solamente el índice de uno de los triángulos adyacentes, de esta forma se puede realizar un recorrido de todos los triángulos adyacentes a un nodo.
Entre las estructuras de datos consultadas en la bibliografía se decidió utilizar media-arista debido a que el algoritmo VolSimp, seleccionado para la simplificación de las mallas triangulares, realiza un gran número de operaciones vectoriales sobre las aristas adyacentes a un nodo y las estructuras de datos basadas en triángulos utilizan mayor cantidad de costo computacional que las basadas en media arista para este tipo de operaciones (Botsch et al., 2002).

1.3.4 Métricas de error y volumen delimitador.

Una de las métricas utilizadas para dirigir la extracción multi-resolución de la triangulación en los modelos multi-resolución es el error espacio-pantalla (Cohen y Manocha, 2005), esta métrica define el error de la proyección de la TIN de un nodo en la pantalla. Este error es obtenido a partir del volumen delimitador y del error espacio-objeto del nodo y se hace más pequeño en un nodo a medida que el punto de visión se aleja del volumen delimitador de este.

El error espacio-objeto en el campo de la simplificación de terrenos define el error de una región aproximada con respecto a la región original. A diferencia del error espacio-pantalla el error espacio-objeto es independiente del campo de visión de la cámara. En (Cignoni et al., 2003) se define el error espacio-objeto de un nodo como la máxima diferencia vertical entre su TIN y la malla original.

Volumen Delimitador

El volumen delimitador de un nodo define una figura geométrica que contiene en su interior el volumen de la geometría del nodo en su totalidad (Andersen y Bay, 2006), entre sus utilidades se encuentran la detección de colisiones entre objetos y la aproximación del área de la proyección de un objeto en un plano. Entre las distintas formas consultadas en la bibliografía para definir el volumen delimitador se encuentran:

- Esfera delimitadora: Representa una esfera en tres dimensiones que en su interior contiene completamente el objeto.
- Caja delimitadora orientada: Define un paralelepípedo rectangular en tres dimensiones.
- Caja delimitadora alineada con los ejes: Define un paralelepípedo rectangular en tres dimensiones con aristas paralelas a los ejes de coordenadas.

De los tres tipos de volumen delimitador mencionados se decidió utilizar esfera delimitadora debido a que presenta menor consumo de memoria que los otros tipos de volúmenes delimitadores y las operaciones realizadas tienen menor costo computacional.

1.4 Manejo de memoria

Debido al tamaño elevado de la información almacenada por las estructuras de datos asociadas a la modelación multi-resolución de terrenos, se hace necesario disminuir la cantidad de polígonos a representar así como lograr el almacenamiento de la información y el acceso a la misma haciendo uso del menor consumo posible de memoria y tiempo de ejecución. Existe gran variedad de técnicas orientadas a la optimización de la construcción de los modelos multi-resolución.

1.4.1 Curvas de relleno de espacios.

Una curva de relleno de espacios define un recorrido de todos los puntos en un espacio multidimensional (Schamberger y Wierum, 2005). La utilización de las mismas hace posible la localización eficiente de información en estructuras de datos espaciales. La forma más utilizada para obtener una curva de relleno de espacios es mediante algún criterio de subdivisión recursivo.

La curva de Hilbert define un criterio de refinamiento recursivo del espacio basado en cuadrantes, (ver figura 4). El orden de los elementos en cada cuadrante sigue el patrón de dirección en forma de U, esta dirección varía en cada nivel de la jerarquía de los cuadrantes.

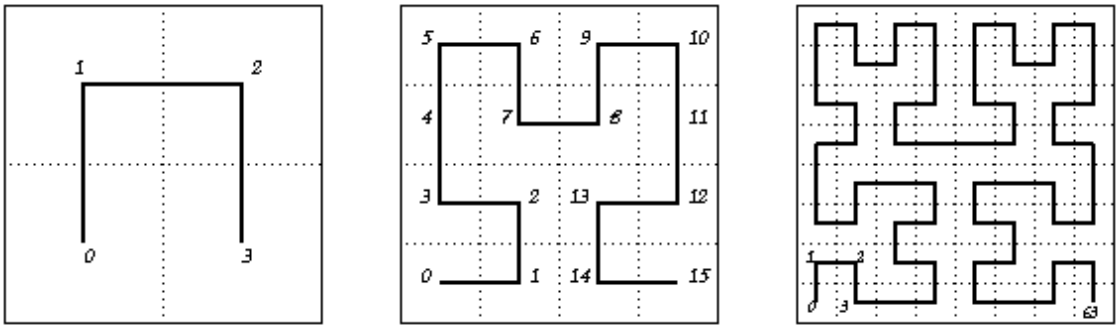

Fig. 4 Curva de relleno de espacios de Hilbert de primer, segundo y tercer orden respectivamente (Lawder y King, 2000).

La curva de Sierpinski define como criterio de refinamiento la división del espacio en triángulos rectángulos mediante la bisección de cada triángulo por el punto medio de la hipotenusa (ver figura 5).

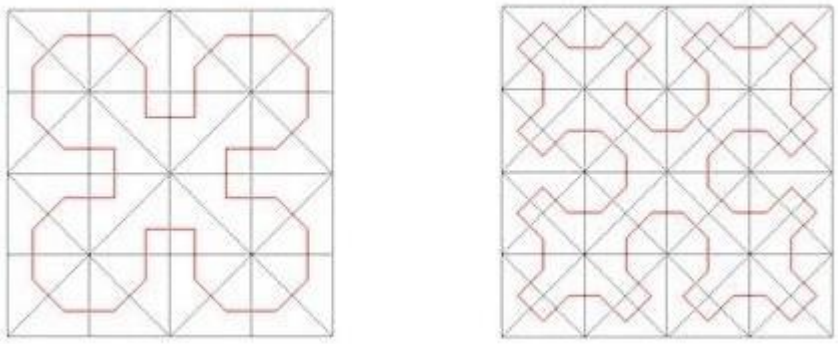

Fig. 5 Curva de relleno de espacios de Sierpinski para dos niveles consecutivos de una HRT (Elaboración propia).

De las curvas anteriormente mencionadas se decidió utilizar la curva de rellenos de espacios de Sierpinski en la presente investigación dado que los nodos del bintree y los triángulos rectángulos definidos en la curva de Sierpinski son equivalentes.

1.5 Tecnologías para el desarrollo

1.5.1 Metodología de desarrollo

Las metodologías de desarrollo se dividen en dos vertientes:

- Metodologías tradicionales o pesadas.
- Metodologías ágiles.

Las primeras centran su atención en llevar una documentación exhaustiva de todo el proyecto, son comúnmente utilizadas en grandes equipos de desarrollo, que se encuentran divididos en distintos departamentos y necesitan generar una gran cantidad de documentación para mantener controlado el proceso de desarrollo y gestionar la comunicación entre los departamentos.

Las metodologías ágiles son flexibles ante requisitos cambiantes y son utilizadas en equipos de desarrollo pequeños. Para el desarrollo de la presente solución se descartan las metodologías pesadas debido a que el equipo de desarrollo de la presente solución es pequeño, por tanto se decide utilizar una metodología ágil. Entre las metodologías ágiles más populares se encuentran XP(Lindstrom y Jeffries, 2004) y OpenUp (Balduino, 2007), se decidió implementar el modelo usando esta última metodología debido a la familiaridad que tiene el desarrollador con la misma y a que esta es la utilizada en el desarrollo del Visor de Terrenos.

Metodología de desarrollo OpenUp

Esta metodología define las fases, actividades y artefactos que se generan durante el ciclo del software. La finalidad de esta metodología de desarrollo es garantizar la eficacia mediante el cumplimiento de los requisitos iniciales y minimizar las pérdidas de tiempo en el proceso de generación del software.

Ventajas de OpenUp:

- Es extensible pues los procesos se pueden agregar o adaptar según lo vayan requiriendo los sistemas.
- Es ligero y proporciona una comprensión detallada del proyecto, beneficiando a clientes y desarrolladores sobre productos a entregar y su formalidad.
- Se centra en una arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo.
- Maneja el ciclo de vida del desarrollo de software de manera apropiada al ofrecer una buena administración de las diferentes áreas del proyecto.

1.5.2 Lenguaje de programación

El Visor de Terrenos en su totalidad está implementado en C#, este es un lenguaje de programación de alto nivel y orientado a objetos creado como parte de las tecnologías .Net de Microsoft. Entre las principales funcionalidades de este lenguaje se pueden destacar:

- **Nombres de Espacios:** C# funciona en el modelo jerárquico de nombres de espacios para mantener la organización en los programas.
- **Recopilador de Basura:** Al igual que el lenguaje de programación Java, C# realiza dinámicamente la liberación de memoria cuando un objeto termina su ciclo de vida.
- **Sintaxis:** C# hereda su sintaxis de Java y C++ tomando las potencialidades de ambos.
- **Interoperabilidad de lenguajes:** Al pertenecer a la plataforma .Net C# permite el acceso a código escrito en cualquiera de los otros lenguajes pertenecientes a .Net y puede heredar las clases escritas en estos lenguajes.
- **Métodos:** Los métodos en C# no son virtuales por defecto, lo que incrementa el rendimiento comparado con otros lenguajes como Java.

Para la implementación de la solución se utilizará C# en vista a las funcionalidades que este provee y para lograr homogeneidad con el desarrollo del Visor de Terrenos.

1.5.3 Lenguaje de Modelado

La falta de estandarización en la forma de representar gráficamente un modelo impide que los diseños realizados se puedan compartir fácilmente entre distintos diseñadores. Se necesita por tanto un lenguaje no sólo para comunicar las ideas a otros desarrolladores sino también para servir de apoyo en los procesos de análisis de un problema. Como lenguaje de modelado, se selecciona el Lenguaje Unificado de Modelado (UML por sus siglas en inglés), el cual permite especificar, visualizar y construir los artefactos de los sistemas de software. Es un sistema de notaciones destinado a los sistemas de modelado que utilizan conceptos orientados a objetos.

1.5.4 Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) es un entorno de programación que ha sido empaquetado como un programa de aplicación, este cuenta con las potencialidades de interpretar, compilar, depurar y diseñar interfaces. Con su utilización los desarrolladores agilizan la creación de programas. Un IDE puede soportar varios lenguajes de programación o estar específicamente diseñado para solo uno y puede estar integrado con alguna plataforma o no.

Se seleccionó como Entorno de Desarrollo Integrado Visual Studio 2013 para lograr compatibilidad con el desarrollo del Visor de Terrenos dado que este está siendo desarrollado haciendo uso de este IDE. Visual Studio posee las siguientes características:

- Compila aplicaciones dirigidas a plataformas de Microsoft, así como aplicaciones web móviles, otras aplicaciones web y servicios en la nube en diferentes dispositivos.

- Conecta el equipo de desarrollo, las partes interesadas y los usuarios finales a través de herramientas integradas.
- Herramientas para pruebas avanzadas garantizan la calidad a lo largo del ciclo de vida de una aplicación.
- Administra y supervisa el progreso de diferentes equipos y registros de trabajo pendiente.
- Posee herramientas de modelado para visualizar la arquitectura del software como es el caso de diagramas UML, el mapa del código y el explorador de arquitectura.

1.5.5 Herramienta CASE

Las herramientas de Ingeniería de Software Asistida por Ordenador (CASE por sus siglas en inglés) permiten organizar y controlar el desarrollo de software, especialmente en proyectos grandes y complejos, estas se utilizan para establecer un modelado de la solución que se debe desarrollar y sirven de guía para los procesos que deben implementar los desarrolladores a lo largo del ciclo de vida del software. Una de las ventajas de usar estas herramientas es conseguir la generación automática de código desde una especificación a nivel de diseño. La herramienta CASE seleccionada para realizar los artefactos pertinentes a las etapas de análisis y diseño de la solución presentada es Visual Paradigm para UML en su versión 8.0. Esta herramienta tiene las siguientes características.

- Está disponible en múltiples plataformas (Microsoft Windows y GNU\Linux).
- Diseño centrado en casos de uso y enfocado al negocio.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Diagramas de flujo de datos.
- Transformación de diagramas de Entidad-Relación en tablas de base de datos.

Conclusiones del Capítulo

En este capítulo se realizó un estudio del marco teórico de las diferentes técnicas para la representación multi-resolución de terrenos, de igual forma se abordaron los algoritmos existentes y las estructuras de datos, así como las diferentes técnicas de optimización utilizadas en el proceso de visualización de terrenos en tres dimensiones.

Para la representación de terrenos en tres dimensiones se seleccionó el modelo BDAM con el objetivo de lograr una disminución de la cantidad de primitivas generadas en la representación de MDEs. Para

la construcción del modelo BDAM se escogió el proceso de simplificación haciendo uso del algoritmo VolSimp debido a que este provee una aproximación de alta calidad con un tiempo de ejecución reducido.

Como estructura de datos para el almacenamiento y manejo eficiente de las triangulaciones durante la construcción del modelo BDAM se eligió media-arista, con el objetivo de dotar al modelo de escalabilidad se decidió hacer uso de la memoria externa durante las etapas de simplificación y extracción de la triangulación del modelo mediante el uso de ficheros, para una mejor localización de los datos la información correspondiente a cada nivel es ordenada haciendo uso de la curva de relleno de espacios de Sierpinski. Para dirigir el proceso de visualización se decidió utilizar la métrica de error espacio-pantalla definiendo una jerarquía de esferas delimitadoras.

Capítulo 2: Análisis y Diseño

2.1 Modelo del Dominio

Un modelo del dominio muestra (a los modeladores) clases conceptuales significativas en un dominio del problema, es el artefacto más importante que se crea durante el análisis orientado a objetos (Larman, 2005). En la figura 6 se muestra el modelo del dominio del Visor de Terrenos.

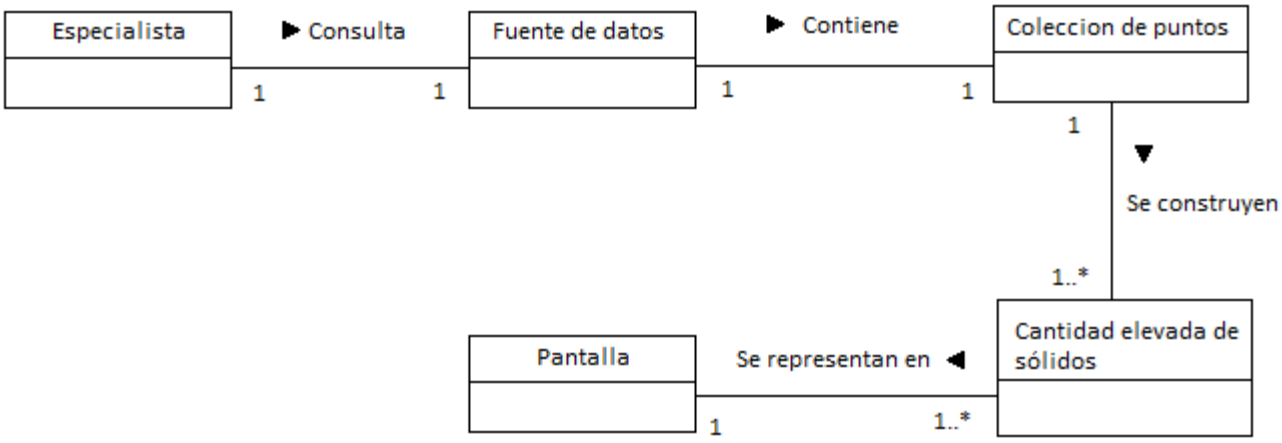

Fig. 6 Modelo del dominio del Visor de Terrenos (Elaboración propia)

Los conceptos asociados al modelo del dominio son:

- Especialista: El especialista representa al usuario que interactúa con los objetos.
- Fuente de Datos: Origen de los datos.
- Colección de puntos: Representa la información contenida en la fuente de datos.
- Cantidad elevada de sólidos: Compuesto por cientos de millones de figuras geométricas obtenidas a través de la colección de puntos.
- Pantalla: La pantalla representa el objeto donde se visualizarán los componentes representados por el motor gráfico, es el punto de comunicación con el usuario.

2.2 Especificación de los requisitos del sistema

La especificación de los requisitos del software es el proceso donde se define una descripción detallada de todos los aspectos del software antes de comenzar la construcción del mismo (Pressman, 2010). En esta se facilita el mecanismo para comprender lo que quiere el cliente, analizando necesidades, confirmando su viabilidad, negociando una solución razonable, especificando la solución sin

ambigüedad, validando la especificación y gestionando los requisitos para que se transformen en un sistema operacional.

2.2.1 Requisitos funcionales

Un requisito funcional define una funcionalidad del sistema de software o de alguno de sus componentes, este describe la forma que el sistema debe comportarse antes un conjunto de entradas y situaciones particulares.

RF1: Construir la estructura de datos multi-resolución a partir del modelo digital de elevación.

Le permite al Visor de Terrenos cargar un MDE y enviárselo al modelo BDAM para construir la estructura de datos multi-resolución.

RF2: Extraer la triangulación multi-resolución de la estructura de datos.

Este requisito permite al Visor de Terrenos obtener una representación multi-resolución del terreno a partir del Frustum.

2.2.2 Requisitos no funcionales

Un requisito no funcional responde a las exigencias de cualidades que se le imponen al proyecto. Es una característica requerida del sistema, del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que señala una restricción del mismo. Mediante estos se busca disponer de un sistema manejable y gestionable que ofrezca la funcionalidad requerida de manera fiable, ininterrumpida o con el tiempo mínimo de interrupción, incluso ante situaciones inusuales.

Eficiencia

RNF1: El tiempo de respuesta para la extracción de la triangulación multi-resolución debe ser inferior a 35 milisegundos.

Requisitos de Software

RNF2. Existencia de la plataforma .Net en su versión 4.0 o superior.

Requisitos de Hardware

RNF3. Procesador: Dual Core 2.5 GHz como mínimo.

RNF4. Memoria RAM: 2 GB como mínimo.

Requerimientos de Restricción del Diseño y la Implementación

RNF5. La solución debe ser implementada utilizando el lenguaje de programación C#.

RNF6. No deben utilizarse bibliotecas o componentes externos a la solución.

2.3 Modelado del sistema

2.3.1 Actores del sistema

Un actor es cualquier elemento con algún comportamiento, incluyendo el propio sistema que se está estudiando cuando solicita los servicios de otros sistemas (Larman, 2005).

Tabla. 1 Actores del sistema.

Actor		Descripción
Visor de Terrenos		Realiza peticiones al modelo

2.3.2 Casos de uso del sistema

Diagrama de Casos de Uso del sistema

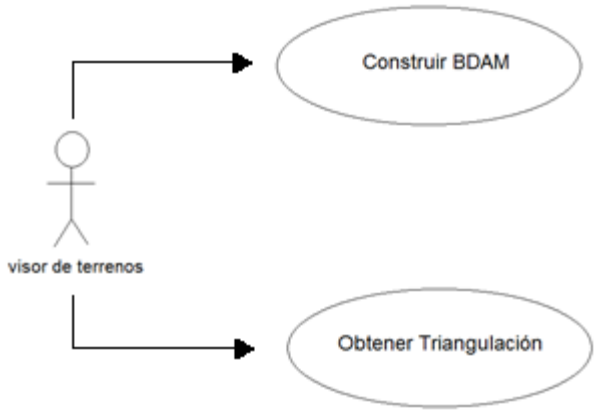

Fig. 7 Diagrama de Casos de Uso de la solución (Elaboración propia).

Descripción de los casos de uso del sistema

Tabla. 2 Descripción del caso de uso Construir BDAM.

Objetivo	Construir una representación multi-resolución de un Modelo Digital de Elevación

Actores	Visor de Terrenos (inicia)	
Resumen	Se inicia cuando el Visor de Terrenos solicita la creación del modelo multi-resolución BDAM sobre el Modelo Digital de Elevación, se construye el modelo multi-resolución BDAM, de esta forma se finaliza el caso de uso.	
Complejidad	Alta.	
Prioridad	Crítica.	
Referencias	RF1.	
Precondiciones	Existencia del Modelo Digital de Elevación	
Poscondiciones	Se genera el fichero con la información del bintree de la BDAM	
Flujo de eventos		
Flujo básico <construir BDAM>		
	Actor	**Sistema**
	1. Solicita la construcción del fichero con la información de la BDAM.	2. Construye el bintree para representar el terreno mediante un recorrido de los puntos. 3. Genera fichero con los datos del bintree.
Flujos alternos		
No existen flujos alternos.		
Relaciones	**CU Incluidos**	No incluye otros casos de uso.
	CU Extendidos	No extiende otros casos de uso.
Requisitos no funcionales	RNF1.	
Asuntos pendientes		

Tabla. 3 Descripción del caso de uso Obtener Triangulación

Objetivo	Extraer una aproximación multi-resolución del terreno basada en el campo de visión de la cámara usando el bintree de la BDAM.

Actores	Visor de Terrenos (inicia)	
Resumen	Se inicia cuando el Visor de Terrenos solicita obtener una aproximación multi-resolución del terreno basada en el frustum actual a partir del fichero del bintree de la BDAM generado en la etapa de construcción, esta es obtenida mediante un recorrido top-down del bintree, se envía al visor de terrenos finalizando de esta forma el caso de uso.	
Complejidad	Alta.	
Prioridad	Crítica.	
Referencias	RF2.	
Precondiciones	Existencia del fichero del bintree generado en la etapa de construcción y el frustum de la cámara.	
Poscondiciones	Se genera una aproximación multi-resolución del terreno basada en el frustum	
Flujo de eventos		
Flujo básico <Obtener Triangulación>		
	Actor	**Sistema**
	1. Solicita la obtención de una aproximación multi-resolución del terreno a partir del frustum	2. Recibe el frustum del Visor de Terrenos
		3. Carga fichero del bintree
		4. Realiza recorrido sobre el bintree y obtiene la aproximación multi-resolución del terreno.
		5. Envía la aproximación multi-resolución del terreno al Visor de Terrenos.
Flujos alternos		
No existen flujos alternos.		
Relaciones	**CU Incluidos**	No incluye otros casos de uso.
	CU Extendidos	No extiende otros casos de uso.

Requisitos no funcionales	RNF1.
Asuntos pendientes	

2.3.3 Arquitectura del sistema.

La arquitectura usada por el Visor de Terreno usa el patrón arquitectónico n-capas, el Visor de Terrenos está compuesto por las siguientes 3 capas.

Capa de Acceso a Datos: Realiza el acceso a los datos externos que usa el Visor de Terrenos.

Capa de modelación: En esta capa están alojadas las estructuras de datos responsables de la triangulación del modelo y la gestión de los elementos que se van a visualizar.

Capa de visualización: contiene todos los componentes responsables de realizar la representación gráfica.

La solución implementada radica en la capa de modelación, este interactúa con los elementos de la capa de acceso a datos para obtener la información referente a los modelos digitales de elevación.

2.3.4 Diagrama de Clases

El diagrama de clases muestra todas las clases del sistema con sus atributos, métodos y las relaciones existentes con otras clases, mediante este se definen las características de los objetos del sistema correspondientes a la programación orientada a objetos. Todas las clases a implementar radican en la capa de modelación del Visor de Terrenos. El Diagrama de clases se observa en la figura 8.

BDAM es la clase principal que define la interfaz presentada al Visor de Terrenos, en esta se realizan todas las operaciones referentes al modelo, esta hace uso de la instancia *Singleton* de la clase MemoryManager que a su vez es la encargada de realizar las operaciones referentes a la memoria externa.

La clase Patch define la estructura de datos que almacena las redes irregulares de triángulos de los nodos del bintree y realiza las operaciones referentes a dicha estructura, esta clase está compuesta por instancias de las clases PVertex, PHEdge y PFace que representan los nodos, las medias-aristas y los triángulos de la red irregular de triángulos respectivamente.

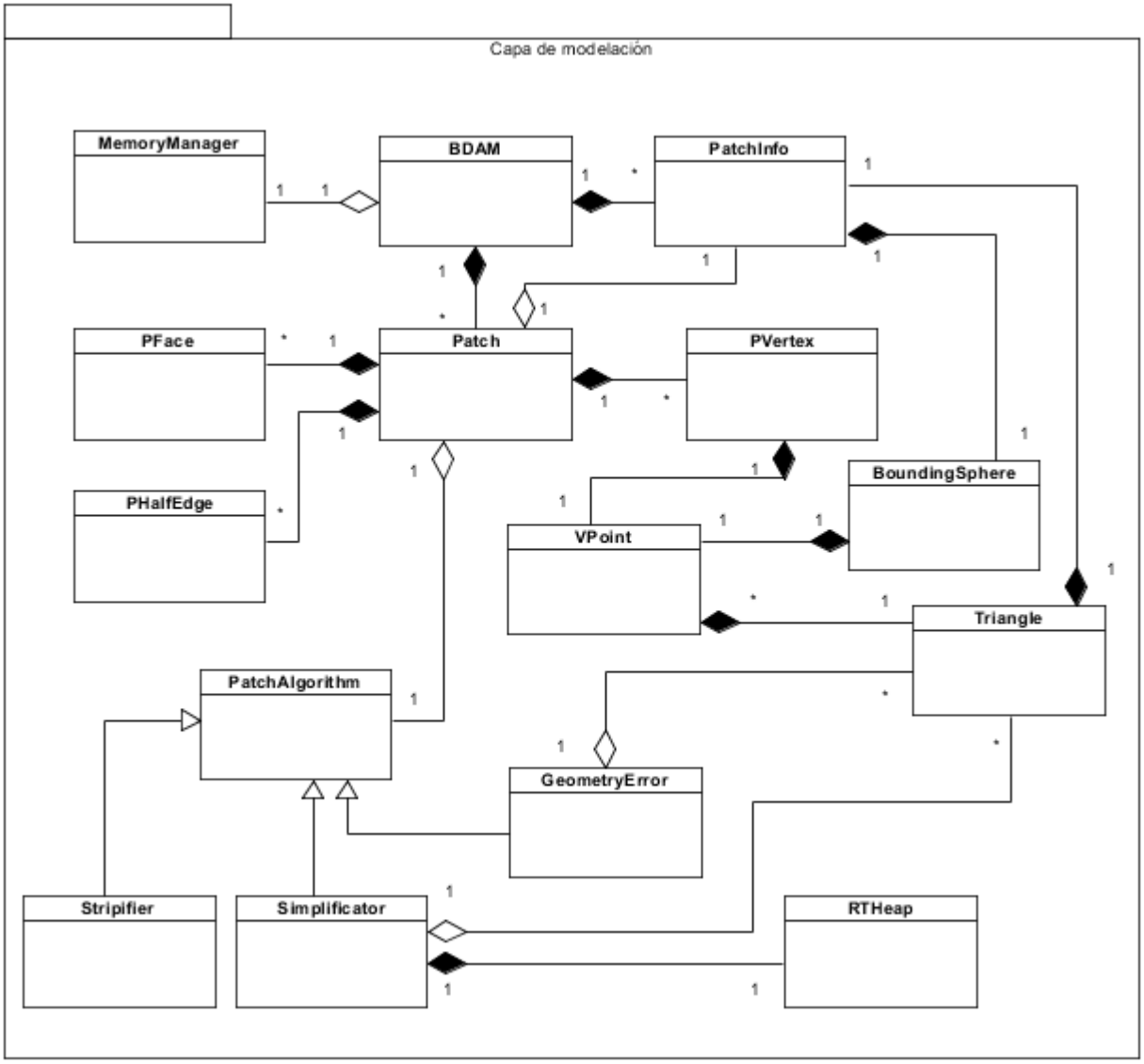

Fig. 8 Diagrama de clases (Elaboración propia).

La clase abstracta PatchAlgorithm define una interfaz para los algoritmos ejecutados sobre la estructuras de datos referentes a las instancias de la clase Patch. Entre los algoritmos utilizados para las instancias de la clase Patch se encuentran:

- **Simplificator:** Clase encargada de realizar la simplificación de las instancias de la clase Patch, esta hace uso de una instancia de la clase RTHeap que representa una cola con prioridad en la cual se pueden actualizar los valores dinámicamente.

- **Stripifier:** Las instancias de esta clase son utilizadas para obtener mediante algoritmos basados en heurísticas las cadenas de triángulos.
- **GeometryError:** Esta clase representa el algoritmo utilizado para obtener el error de aproximación de las redes irregulares de triángulos con respecto al MDE original.

2.3.5 Patrones de diseño

(Larman, 2005) define un patrón como un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos. Los patrones son utilizados en el desarrollo del software para establecer una manera de organizar y estructurar el desarrollo. Estos son una guía para realizar algunas acciones dentro del proceso de desarrollo, especificando un conjunto de subsistemas predefinidos y las funcionalidades de cada uno. Además facilitan la comprensión de la situación en que se encuentre algún problema que se quiera solucionar.

Los Patrones de Principios Generales para Asignar Responsabilidades (GRASP por sus siglas en inglés) Describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones (Larman, 2005). De los patrones pertenecientes a GRASP se utilizaron en la presente solución los siguientes:

Patrón Experto:

Se le asigna la responsabilidad al experto en información que es la clase que tiene la información necesaria para cumplir con dicha responsabilidad (Larman, 2005). Los siguientes son ejemplos del uso de este patrón en la solución propuesta.

- La clase *Simplificator* tiene la responsabilidad realizar todas las tareas de simplificación en los nodos del bintree.
- La clase TriangleStrip que es la encargada de realizar todas las tareas referentes a la obtención de cadenas de triángulos.
- La clase MemoryManager es la encargada de almacenar y recuperar la información a ser procesada durante la construcción del modelo.

Patrón Creador:

Se le asigna la responsabilidad a una clase de crear instancias de otras clases. En la presente solución este patrón se evidencia en la clase *Patch*, donde los objetos que instancian esta clase tienen la responsabilidad de crear las instancias de la clase *PVertex* y *PHalfEdge*, ver figura 9.

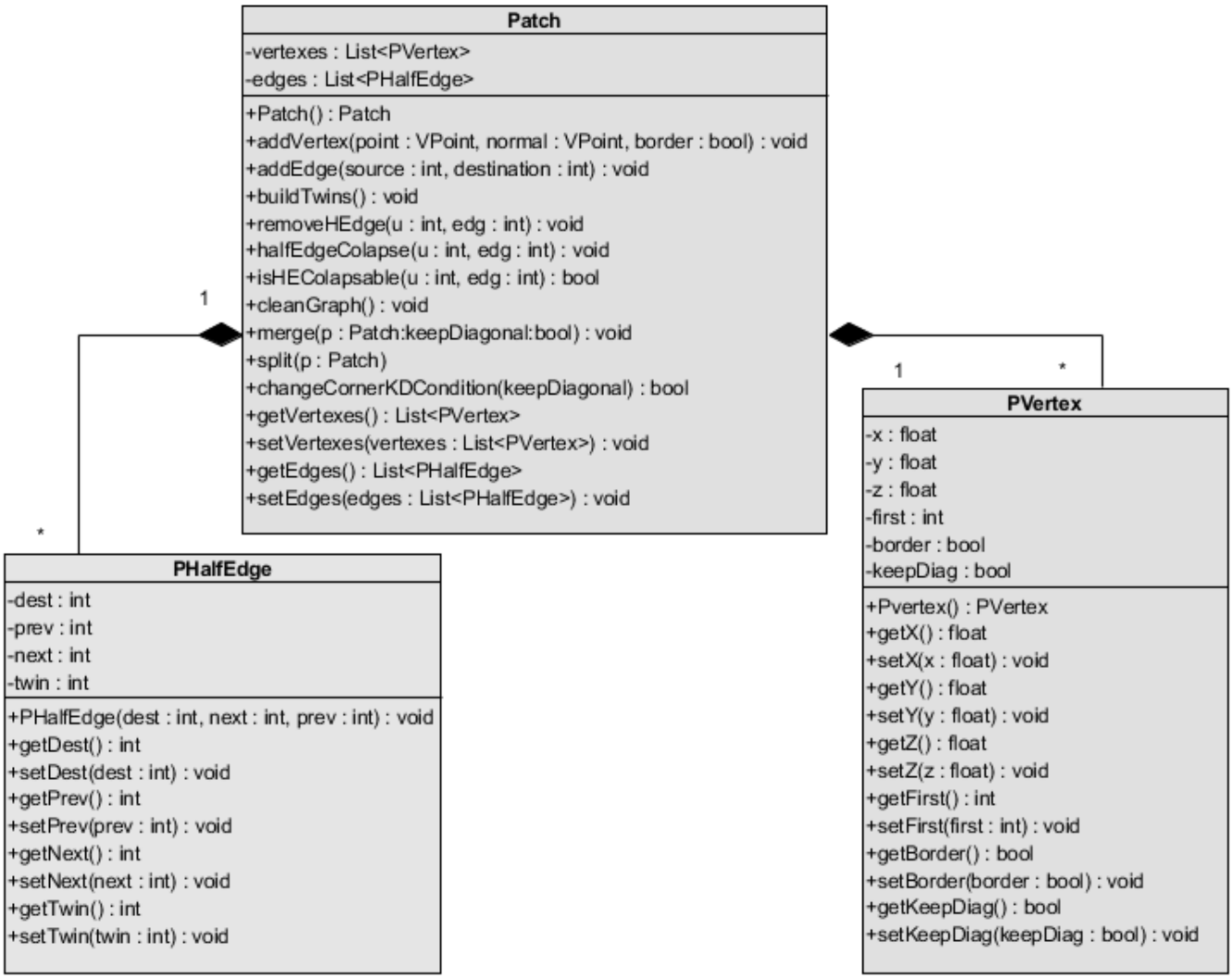

Fig. 9 Ejemplo de patrón creador (Elaboración propia).

Patrón Bajo Acoplamiento:

(Larman, 2005) define el acoplamiento como una medida de la fuerza con que un elemento está conectado a, tiene conocimiento de, confía en, otros elementos. Este patrón fue utilizado para diseñar todas las clases del modelo.

Patrón Alta Cohesión:

En cuanto al diseño de objetos, la cohesión es una medida de la fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento. Un elemento con responsabilidades altamente relacionadas, y que no hace una gran cantidad de trabajo, tiene alta cohesión. Las clases con alta cohesión son muy útiles porque simplifican el mantenimiento, mejoran la claridad y son más fáciles de reutilizar. El uso de este patrón se evidencia en todas las clases del modelo principalmente en la clase BDAM que divide las responsabilidades entre las clases *MemoryManager, PatchInfo* y *Patch*.

Grupo de cuatro [2](Vlissides et al., 1995) define otro gran grupo de patrones que se evidencian en la construcción de software, de este grupo se utilizaron en la solución los siguientes patrones:

Patrón Singleton: Asegura que una clase tenga una única instancia y solamente exista un punto de acceso a la misma (Vlissides et al., 1995). En la presente solución solamente se necesita una instancia de la clase MemoryManager para el manejo de la memoria del modelo, esta es accedida por las demás clases mediante el método estático *getInstance()* de la clase *MemoryManager* debido a que el constructor de la misma es privado.

Patrón Estrategia: Este patrón se encarga de definir una familia de algoritmos, encapsular cada uno y hacerlos intercambiables (Vlissides et al., 1995), el patrón Estrategia permite que el usuario del objeto escoja los algoritmos que este debe usar en cada contexto específico. La clase *SimplificationAlgorithm* define la familia de algoritmos referentes a la simplificación de terrenos, la clase *Patch* hace uso de una instancia de esta clase para reducir el número de primitivas en su geometría, se utiliza en la presente solución la clase *Simplificator* que define un algoritmo de simplificación de terrenos basado en colapsamiento media-arista, esto brinda la posibilidad de cambiarlo en tiempo de desarrollo y en tiempo de ejecución por algún otro algoritmo.

Patrón peso ligero[3]: Según se define en (Vlissides et al., 1995), un peso ligero es un objeto que puede ser usado en múltiples contextos, este posee estados intrínsecos y estados extrínsecos. Los estados intrínsecos consisten en la información independiente del contexto, esta es almacenada en el peso ligero. Los estados extrínsecos varían con el contexto y son obtenidos a través de los restantes objetos. Este patrón es usado cuando se dan las siguientes circunstancias.

- Una aplicación usa una cantidad elevada de objetos.
- El costo de almacenamiento es alto producto a la cantidad de objetos.

[2] Del Inglés *Gang of Four*
[3] Del Inglés *FlyWeight*

- La mayoría de los estados pueden ser extrínsecos.
- Muchos grupos de objetos pueden ser reemplazados por unos pocos objetos una vez eliminados los estados extrínsecos.

En la solución propuesta se evidencia este patrón siendo las instancias de la clase *Patch* el peso ligero, el costo de almacenar toda la información referente a los elementos de tipo *Patch* puede llegar sobrepasar la memoria interna del sistema, de esta forma toda la información extrínseca de los objetos de tipo *Patch* es almacenada en la memoria externa por la instancia Singleton de la clase *MemoryManager*, esta a su vez se encarga de recuperar los datos extrínsecos cuando estos son necesarios, la clase BDAM funciona como fábrica en la creación de las instancias de los pesos ligeros, de esta forma son cargadas simultáneamente en memoria unas pocas instancias de la clase *Patch* que son compartidas para todos los objetos de este tipo.

2.4 Proceso propuesto para la modelación multi-resolución de terrenos en 3D usando BDAM.

El modelo BDAM consta de dos etapas para la representación multi-resolución de terrenos.

- Etapa de construcción:
- Etapa de extracción de la triangulación.

El esquema representado en la figura 10 describe la interacción en cada una de las etapas entre el Visor de Terrenos y el Modelo BDAM, esta interacción comienza cuando el Visor de Terrenos solicita la representación multi-resolución de un modelo digital de elevación, este es enviado al modelo BDAM donde se da inicio a la etapa de construcción.

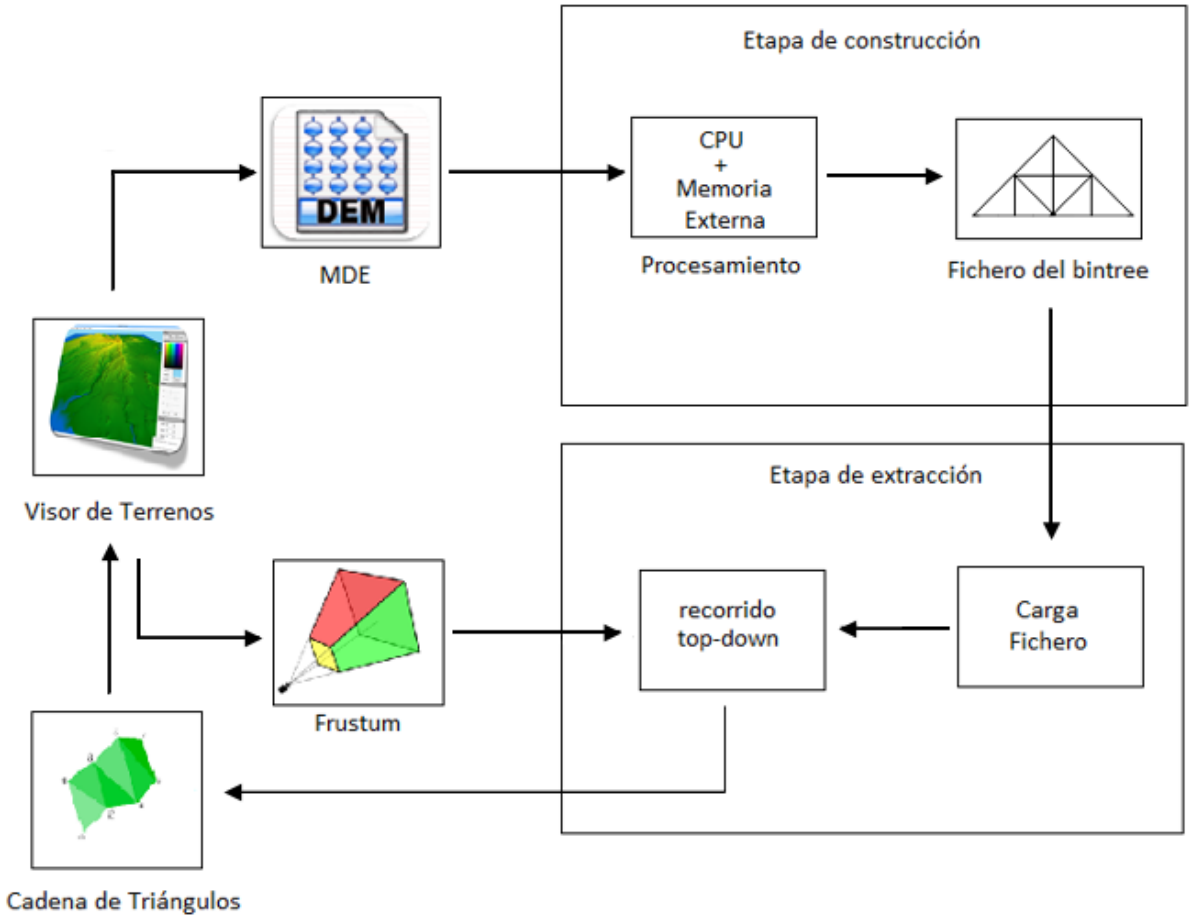

Fig. 10 Esquema de la interacción entre el modelo BDAM y el Visor de Terrenos (Elaboración propia).

Durante la etapa de construcción del modelo BDAM se utilizan dos ficheros, un fichero temporal utilizado para almacenar los vértices, aristas y las normales correspondientes a la TIN de cada nodo procesado y el fichero del *bintree* que contiene la información necesaria para visualizar la TIN de cada nodo.

El procesamiento se realiza partiendo desde los diamantes del nivel con mayor resolución hasta los diamantes del nivel de menor resolución, durante la construcción de los mismos se siguen los siguientes pasos:

1. Se cargan del fichero temporal los cuatro nodos del nivel anterior correspondientes al diamante, estos son unidos por los bordes actualizando la información de los mismos.
2. Se ejecuta el algoritmo de simplificación VolSimp en el diamante para reducir la cantidad de triángulos a la mitad.
3. Se divide el diamante en los dos nodos correspondientes al nivel actual.

4. Se evalúa el error objeto-espacio resultante de la simplificación en cada uno de los nodos del diamante.
5. Se obtiene la cadena de triángulos mediante el algoritmo SGI en cada uno de los nodos del diamante.
6. Se almacenan los nodos en el fichero temporal y en el fichero del *bintree.*

Para mejor localización de los datos se ordena la información por niveles, en cada nivel los nodos son ordenados usando la curva de relleno de espacios de Sierpinski.

Una vez terminada la etapa de construcción del modelo BDAM comienza la etapa de extracción de la triangulación, esta etapa recibe como entrada el Frustum de la cámara y devuelve como salida una cadena de triángulos que es enviada al Visor de Terrenos para su visualización, en cada momento puede obtenerse una aproximación del terreno para ser visualizada, esta representación se obtiene mediante un refinamiento top-down sobre el *bintree*. Para dirigir el refinamiento se utiliza la métrica del error espacio-pantalla, una cuota superior de este error es obtenido al medir el tamaño aparente en la pantalla de una esfera con centro en el punto más cercano del volumen delimitador del nodo al punto de visión de la cámara y con radio igual al error del nodo con respecto a la malla original.

En cada momento se verifica usando la métrica de error espacio-pantalla si un nodo necesita ser refinado, en caso de necesitarlo se reemplaza al nodo por sus dos hijos aumentando de esta forma la resolución del área cubierta por el nodo, en caso contrario se obtiene la cadena de triángulos correspondiente a la TIN del nodo y se concatena con la cadena de triángulos de salida.

Conclusiones del Capítulo

Durante la especificación de los requisitos se obtuvieron seis requisitos no funcionales y dos requisitos funcionales, a partir de estos últimos se generaron dos casos de uso, garantizando de esta forma que la presente solución resuelva la problemática planteada. Tanto la arquitectura propuesta como los patrones de diseños utilizados permitieron dotar a la solución de extensibilidad, obteniéndose de esta forma un diseño entendible para otros desarrolladores. Se realizó el diseño de las clases mediante el cual se establecieron las relaciones entre las mismas.

Al final del capítulo se dio una descripción detallada de la solución propuesta y se definieron los pasos a seguir para la realización de cada una de las etapas del modelo híbrido BDAM. La presente solución combina técnicas de optimización, algoritmos para la simplificación de mallas triangulares, algoritmos

para la obtención de cadenas de triángulos, estructuras de datos espaciales y presenta escalabilidad mediante el uso de la memoria externa del sistema para la realización de las operaciones.

Capítulo 3: Implementación y Pruebas

3.1 Estándares de codificación

Un estándar de codificación define la forma y los estilos utilizados en el código fuente de un programa, lo que permite a los desarrolladores de un proyecto generar código fácil de entender para otros desarrolladores.

La presente solución sigue el estándar de codificación de .NET detallado en (Hunt, 2007).

3.1.1 Terminologías y conceptos

Estilo Camel: Una cadena con la primera letra minúscula y la primera letra de cada palaba subsiguiente mayúscula.

Estilo Pascal: Una cadena con la primera letra mayúscula y la primera letra de cada palaba subsiguiente mayúscula.

Identificador: Una cadena utilizada para identificar de forma única un objeto o una instancia de un objeto.

Modificador de acceso: Definen la accesibilidad de los miembros de las clases y métodos. Las palabras claves de C# utilizadas con este propósito son **`public`**, **`protected`**, **`internal`**, y **`private`**.

3.1.2 Uso y sintaxis de nombres

La tabla 4 define la forma en que los nombres serán utilizados en la presente solución.

Tabla. 4 Uso y sintaxis de los nombres en el formato identificador/convención de nombre.

Identificador	Convención de nombre

Clase (*Class*) o Estructura (*Struct*)	Estilo Pascal.
	Se usa un sustantivo o frase sustantiva para el nombre de la clase.
	Ejemplo:
	public class MemoryManager
	{
	}

Interfaz	**Estilo Pascal.**
	Siempre se adiciona la letra mayúscula ´I´.
	Ejemplo:
	InterfaceI Simplificator
	{
	}

Método	**Estilo Pascal.**
	El método viene en el formato verbo o verbo-sustantivo.
	Ejemplo:
	public void AddVertex()
	{
	}

Propiedad (*Property*)	**Estilo Pascal.**
	Ejemplo:
	public float NumberOfVertexes
	{
	get{return _vertexCounter;}
	set{_vertexCounter = value;}
	}

Atributo (*public*, *protected*, *internal*)	**Estilo Pascal.** Usar propiedades en vez de atributos públicos. Ejemplo: protected totalArea;
Atributo (*private*)	**Estilo Camel:** Se le concatena como prefijo un guion bajo (_). Ejemplo: private _triangleCounter;
Variable (local)	**Estilo Camel.** Ejemplo: int auxiliaryVariable;

Parámetro	**Estilo Camel.**
	Ejemplo:
	public void simplifyTo(int triangleCount)
	{
	}

3.1.3 Estilo de codificación

La organización del código es fundamental para la generación de código entendible. Los elementos a tener en cuenta con respecto al estilo de codificación fueron tomados de (Hunt, 2007).

1. Siempre iniciar las llaves en una nueva línea.
2. En las sentencias condicionales siempre usar corchetes.
3. Siempre usar margen de tamaño cuatro.
4. El orden a seguir para los elementos en una clase es el siguiente:
    a) Atributos
    b) Constructores y Finalizadores.
    c) Enumeradores, Estructuras y Clases.
    d) Propiedades
    e) Métodos
5. En cada uno de los grupos anteriores se ordenan los elementos de acorde al siguiente orden del modificador de acceso.
    a) public
    b) protected
    c) internal
    d) private
6. Cada variable se declara independiente, no en la misma sentencia.
7. Todos los comentarios se escriben en el mismo idioma sin errores gramaticales.

3.2 Implementación de la etapa de construcción del modelo BDAM

En esta sección se abordarán los principales detalles de implementación de la etapa de construcción del modelo BDAM mencionada en el capítulo 2.

3.2.1 Ficheros utilizados durante la construcción

Durante la etapa de construcción del modelo se generan dos ficheros binarios para el trabajo con la memoria externa del sistema, uno temporal que es eliminado al finalizar la etapa de construcción y el fichero del *bintree* que es la entrada recibida por la etapa de extracción de la triangulación.

La información del *bintree* referente a la construcción es almacenada en el fichero temporal con el fin de realizar el procesamiento escalable de los datos. La ubicación de los nodos en este se realiza de forma lineal siguiendo el orden de la curva de relleno de espacios de Sierpinski, para cada nodo se almacenan los vértices y aristas correspondientes a su TIN y las normales. La esfera delimitadora, el error espacio-

objeto y el triángulo delimitador de cada nodo se almacenan en memoria interna para aliviar la cantidad de operaciones de lectura y escritura realizadas.

El fichero del *bintree* contiene la información necesaria para la extracción multi-resolución de las triangulaciones en el modelo. La estructura de este fichero se presenta en la tabla 5.

Tabla. 5 Estructura del fichero del *bintree*

Descripción	Tipo de datos	Cantidad de bytes
Cantidad de nodos del bintree (N)	Entero de 32 bits	4
Error espacio-objeto de cada nodo	Decimal de 32 bits	4N
Esfera delimitadora de cada nodo	Decimal de 32 bits	16N
Posición en el fichero de los datos de cada nodo	Entero de 64 bits	8N
Datos	Decimal de 32 bits Entero de 16 bits	$\sum_{i=1}^{N} \text{BytesDelNodo}(i)$

Las cuatro primeras filas de la tabla representan la cabecera del fichero del bintree, el orden de los datos de cada nodo se presenta en la tabla 6.

Tabla. 6 Orden de los datos de un nodo en el fichero del *bintree*

Descripción	Tipo de datos	Cantidad de bytes
Cantidad de vértices de la TIN (V)	Entero de 16 bits	2
Coordenadas x,y,z de cada vértice	Decimal de 32 bits	12V
Valores x,y,z de las normales de cada vértice	Decimal de 32 bits	12V
Cantidad de índices de la cadena de Triángulos (P)	Entero de 16 bits	2
Índices correspondientes a la cadena de triángulos	Entero de 16 bits	2P

En la figura 11 se muestra la interacción de la etapa de construcción del modelo BDAM con los dos ficheros creados.

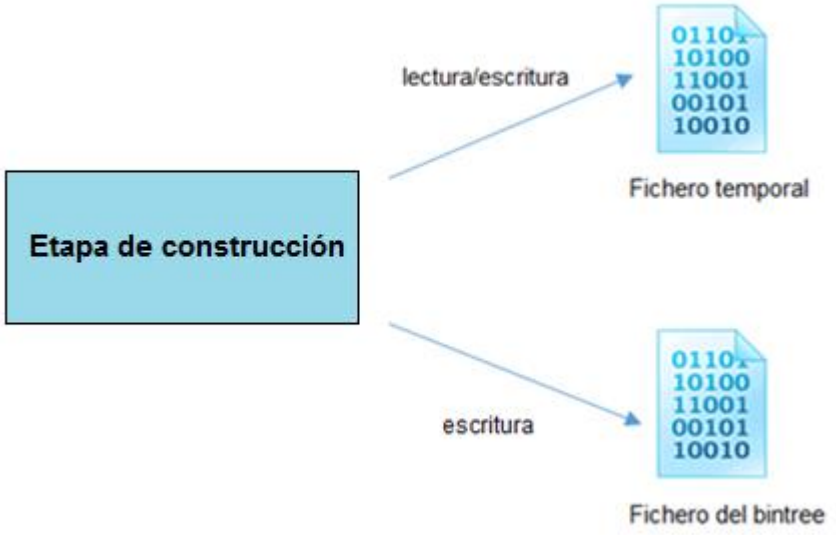

Fig. 11 Operaciones sobre ficheros en la etapa de construcción (Elaboración propia)

3.2.2 Obtención del Modelo Digital de Elevación

Con el propósito de obtener una representación multi-resolución de un MDE el modelo necesita tener conocimiento de la siguiente información:

```
1    ncols = cantidad de columnas de la matriz de alturas.
2    nrows = cantidad de filas de la matriz de alturas.
3    xdim = distancia entre los puntos en el eje X.
4    ydim = distancia entre los puntos en el eje Y.
5    nodata_value = representa la ausencia de valor en ese punto.
```

Estos valores son estándares y aparecen en todos los formatos de MDE. Para realizar el proceso de construcción del modelo BDAM los valores *nrows* y *ncols* tienen que ser iguales y estar en la forma $2^k + 1$ para realizar las divisiones correspondientes a la jerarquía de triángulos rectángulos, con este objetivo se halla el menor valor de k tal que $2^k + 1$ >= nrows y $2^k + 1$ >= ncols, luego se completa con valores nulos las celdas nuevas al tomar *ncols* y *nrows* valor igual a $2^k + 1$. Esta transformación de la matriz original es la entrada que recibe el modelo BDAM para realizar su construcción.

3.2.3 Construcción del bintree

Los MDEs al ser transformados poseen dimensiones cuadradas lo que imposibilita el uso de un solo *bintree* para la representación de los mismos, con el objetivo de resolver este problema se crea un nodo raíz artificial para cubrir el área total del terreno, sus hijos son obtenidos al dividir el cuadrado por la

diagonal. La cantidad de triángulos en la TIN de cada nodo del bintree es constante para todos los nodos sin importar el nivel. Mientras mayor sea la cantidad de triángulos por nodo del bintree mayor será la cantidad de niveles y como resultado mayor será la cantidad de resoluciones utilizadas para representar el terreno, una cantidad elevada de niveles le dificulta el trabajo a los algoritmos de simplificación debido a que hay mayor probabilidad de que rasgos del terreno se pierdan al ser la cantidad de triángulos por nodo un valor pequeño.

En la implementación actual del modelo se definió la cantidad de triángulos por nodo igual a 1024 (1k) con el fin de encontrar un balance entre cantidad de niveles y cantidad de triángulos por nodo. Este valor permite que el tipo de datos utilizado para almacenar los índices de las aristas y vértices en la TIN de cada nodo del bintree sea el entero de 16 bits (tipo de datos *short* en C#).

Una vez decidida la cantidad de triángulos en la TIN correspondiente a cada nodo del *bintree*, se define su jerarquía en un proceso de construcción top-down haciendo uso del algoritmo de búsqueda primero a lo ancho (BFS por sus siglas en inglés) de teoría de grafos. En el siguiente pseudocódigo se detalla la creación del *bintree*.

```
1    Q = Cola de enteros
2    rootNode1 = Hijo raíz 1
3    rootNode2 = Hijo raíz 2
4    Q.enqueue(rootNode1)
5    Q.enqueue(rootNode2)
6    while(Q.Count != 0)
7        nodoActual = Q.dequeue()
8        patchList.add(nodoActual)
9        if(nodoActual.level == LEAVE_LEVEL)
10           ProcessLeaveNode(nodoActual)
11       else
12           hijoIzquierdo = nodoActual.CreateLeftChild()
13           hijoDerecho = nodoActual.CreateRightChild()
14           Q.enqueue(hijoIzquierdo)
15           Q.enqueue(hijoDerecho)
```

El método *ProcessLeaveNode()* de la línea 10 almacena en los ficheros la información correspondiente a los nodos hojas del árbol. Los métodos *CreateLeftChild()* y *CreateRightChild()* obtienen los hijos izquierdo y derecho respectivamente del nodo, el hijo izquierdo del nodo tiene que estar antes en el orden de la curva de Sierpinski.

La información de los nodos intermedios del árbol se obtiene a partir de un recorrido *bottom-up* de los diamantes del *bintree*. El proceso es detallado en el siguiente pseudocódigo:

```
for n = niveles(bintree) − 1 to 1
    foreach d ∈ diamantes(n)
        CargarNodosHijosEnMemoriaInterna()
        diamante = UnirNodosHijos()
        VolSimp.SimplificarMitad(diamante)
        CalcularErrorEspacioObjeto(diamante)
        CalcularVolumenDelimitador(diamante)
        Diamante.DividirPorLaDiagonal(nodo1,nodo2)
        CalcularCadenaDeTriángulos(nodo1)
        CalcularCadenaDeTriángulos(nodo2)
        AlmacenarEnMemoriaExterna(nodo1)
        AlmacenarEnMemoriaExterna(nodo2)
```

3.2.4 Estructura de datos para la representación de la TIN de un nodo.

La clase *Patch* es la responsable de almacenar los datos correspondientes a la TIN de cada nodo. Esta consta de una lista de vértices y una lista de media-aristas. A continuación se muestran los detalles de implementación de los elementos de la TIN.

Estructura de una media-arista:

```
PHalfEdge{
    Next : Entero
    Prev : Entero
    Dest : Entero
    Twin : Entero
}
```

Estructura de un vértice:

```
PVertex{
    X : Punto Flotante
    Y : Punto Flotante
    Z : Punto Flotante
    First : Entero
}
```

En la media-arista, *Next* y *Prev* representan la siguiente y la anterior media-arista en el sentido de las manecillas del reloj, *Dest* es el nodo destino y *Twin* la media-arista en sentido contrario con origen en *Dest*. De esta forma solo se necesita almacenar para cada vértice una sola arista (*First*). Esta estructura de datos permite la realización de operaciones complejas como por ejemplo:

- ***a)*** Recorrido de todos los triángulos adyacentes a un nodo ***A*:**

```
aristaActual = vértices(A).First;
do
    aristaSiguiente = aristas(aristaActual).Next;
    B = aristas(aristaActual).Dest;
    C = aristas(aristaSiguiente).Dest;
    triangulo = Triangle(vertices(A), vertices(B), vertices(C));
    aristaActual = aristas(aristaActual).Next;
while (aristaActual != vértices(A).First);
```

3.2.5 Algoritmo VolSimp para la simplificación de los nodos.

El método *SimplifyHalf* de la clase *Simplificator* disminuye la cantidad de triángulos de una instancia de la clase *Patch* a la mitad obteniéndose una aproximación de la malla haciendo uso del algoritmo de simplificación de mallas triangulares *VolSimp*. Este algoritmo basa la simplificación de la malla en la eliminación de vértices mediante un algoritmo ávido que consta de dos fases, la primera fase consiste en la selección del vértice *v* menos importante de la malla usando la variación en el campo normal (HUSSAIN ,2013) definida como $VC(v)$.

$$VC(v) = \sum_{ti} \Delta_{ti} - \left\| \sum_{ti} \Delta_{ti} n_{ti} \right\| \tag{1}$$

Donde ti es un triángulo adyacente a v, Δ_{ti} representa el área del triángulo y n_{ti} es un vector unitario equivalente a la normal del plano definido por ti.

La segunda fase consiste en encontrar la media-arista con origen en *v* que menor pérdida de volumen genere al ser colapsada mediante la técnica media-arista. La pérdida de volumen para una media-arista hi se calcula mediante la función $HC(hi)$:

$$HC(hi) = \sum_{tj} TC(j) \tag{2}$$

$$TC(t) = \frac{1}{6}[(v_1 - v) \times (v_2 - v) \cdot (v_i - v)]^2 \tag{3}$$

El siguiente pseudocódigo muestra los detalles del algoritmo VolSimp:

```
Entrada:
M = Malla Triangular (Patch)
cantidad = cantidad de triángulos a simplificar
Salida:
M' = Malla Triangular aproximada
```

```
6
7    heap = Heap de pares (Vértice, Punto Flotante)
8    foreach v ∈ vértices(M)
9        heap.Insertar(Par(v,VC(v)))
10   while(M.CantidadTriangulos() > cantidad)
11       mejorVertice = heap.ObtenerMejor()
12       arista = AristaMenorPerdidaVolumen(mejorVertice)
13       M.colapsarMediaArista(arista)
14       heap.EliminarMejor()
```

En la línea 12 la arista hi de menor pérdida de volumen se escoge teniendo en cuenta $HC(hi)$.

3.2.6 Algoritmo SGI para la obtención de cadenas de triángulos.

El método *Stripify* de la clase *Stripifier* obtiene una cadena de triángulos a partir de una instancia de la clase *Patch* haciendo uso del algoritmo ávido SGI.

Algoritmo SGI:

```
1    Entrada:
2    M = Malla Triangular (Patch)
3    Salida:
4    Cadena de Triángulos
5
6    heap = Heap de pares (Triángulo, Entero)
7    foreach t ∈ triángulos(M)
8        heap.Insertar(t, t.CantidadAdyacentes())
9    cadenaSalida = Cadena de Triángulos
10   while(heap.ContieneElementos() == false)
11       mejorTriangulo = heap.ObtenerMejor()
12       mejorCadena = vacio
13       foreach t ∈ TriangulosAdyacentesA(mejorTriángulo)
14           cadena = Extender(t)
15           if(cadena.Longitud > mejorCadena.Longitud)
16               mejorCadena = cadena
17       foreach t ∈ mejorCadena()
18           heap.Eliminar(t)
19       Concatenar(cadenaSalida, mejorCadena)
```

El *heap* definido en la línea 6 mantiene un orden de los triángulos acorde a la cantidad de triángulos adyacentes. En cada momento se extrae del *heap* el triángulo con la menor adyacencia y se realiza una búsqueda con el método *Extender* (línea 14) de la mayor cadena de Triángulos que se puede obtener en un camino con *mejorTriangulo* como nodo inicial. Luego de seleccionada la cadena más larga esta es concatenada a la cadena de triángulos de salida y se actualiza la información del heap eliminando los triángulos pertenecientes a la cadena y sus adyacencias.

3.3 Implementación de la etapa de extracción de la triangulación del modelo BDAM

La etapa de extracción de la triangulación tiene su inicio luego de la construcción del bintree, la entrada recibida por esta etapa es el campo de visión de la cámara (frustum) y el fichero del *bintree*, la salida resultante es una cadena de triángulos que es representada por el Visor de Terrenos. En esta sección se muestran los principales detalles de implementación de esta etapa.

3.3.2 Recorrido top-down del bintree del modelo BDAM

El recorrido del bintree parte desde el nodo raíz, en cada iteración se decide si un nodo puede ser representado usando la métrica de error espacio-pantalla o si necesita aumentar la resolución al ser sustituido por la unión de sus dos hijos. Una cuota superior de este error es obtenido al medir el tamaño de una esfera con centro en el punto más cercano del volumen delimitador del nodo al punto de visión de la cámara y con radio igual al error espacio-objeto del nodo. La condición para decidir si un nodo necesita refinamiento es:

*radioEsfera > umbralDeError * distancia*

Donde *radioEsfera* es el error espacio-objeto del nodo, *distancia* es la distancia entre el punto de visión de la cámara y el volumen delimitador del nodo y *umbralDeError* es la tolerancia en cantidad de pixeles del error de la proyección del nodo en la pantalla. El pseudocódigo mostrado a continuación describe el proceso de refinamiento *top-down* del *bintree*.

```
1    Entrada:
2    frustum = Frustum
3    ficheroBintree = Fichero del bintree
4    Salida:
5    Cadena de Triángulos
6
7    cadenaSalida = Cadena De Triángulos
8    Q = Cola de Eenteros
9    rootNode1 = Hijo raíz 1
10   rootNode2 = Hijo raíz 2
11   Q.enqueue(rootNode1)
12   Q.enqueue(rootNode2)
13   while(Q.Count != 0)
14       nodoActual = Q.dequeue()
15       if(frustum.Intersecta(nodoActual) == false)
16           continue;
17       if(PuedeVisualizar(nodoActual))
18           cadena = ficheroBintree.ObtenerCadenaDeTriangulos(nodoActual)
19           cadenaSalida.Concatenar(cadena)
```

```
20      else
21          hijoIzquierdo = nodoActual.GetLeftChild()
22          hijoDerecho = nodoActual.GetRightChild()
23          Q.enqueue(hijoIzquierdo)
24          Q.enqueue(hijoDerecho)
```

El recorrido se implementa como una búsqueda a lo ancho sobre el *bintree*. En las líneas 21 y 22 hijoIzquierdo es procesado antes de hijoDerecho siguiendo el orden de la curva de relleno de espacios de Sierpinski. El método PuedeVisualizar devuelve un valor lógico que indica si un nodo del *bintree* cumple con la métrica de error espacio-pantalla teniendo en cuenta un umbral de error predefinido. El método Intersecta de la clase Frustum (línea 15) permite descartar los nodos que están fuera del campo de visión de la cámara, esta técnica se le conoce como recorte de *frustum*.

3.4 Diagrama de Componentes

Un diagrama de componentes modela cómo un sistema de software es dividido en distintos componentes que se relacionan entre sí para darle cumplimiento a las funcionalidades propuestas durante la etapa de diseño de un sistema informático. Los paquetes generados (ver figura 12) en la presente solución son los siguientes:

Estructura de datos: Contiene todos los componentes referente a la estructura de datos para almacenar las mallas triangulares, a continuación se detallan dichos componentes.

- Patch: Contiene los vértices y aristas correspondientes a la TIN de un nodo del bintree.
- PHalfEdge: Contiene la información asociada a una media-arista en la TIN
- PVertex: Este componente almacena los datos de cada vértice de una TIN.
- PFace: Contiene la información correspondiente a un triángulo (en función de nodo) de la TIN y provee funcionalidades para el recorrido de los triángulos adyacentes.
- RTHeap: Este componente funciona como una cola con prioridad, permite la actualización dinámica de los elementos.

Algoritmos: Contiene los componentes referentes a los algoritmos utilizados en el modelo BDAM.

- PatchAlgorithm: Este componente define una plantilla para los algoritmos utilizados en las instancias de la clase Patch.

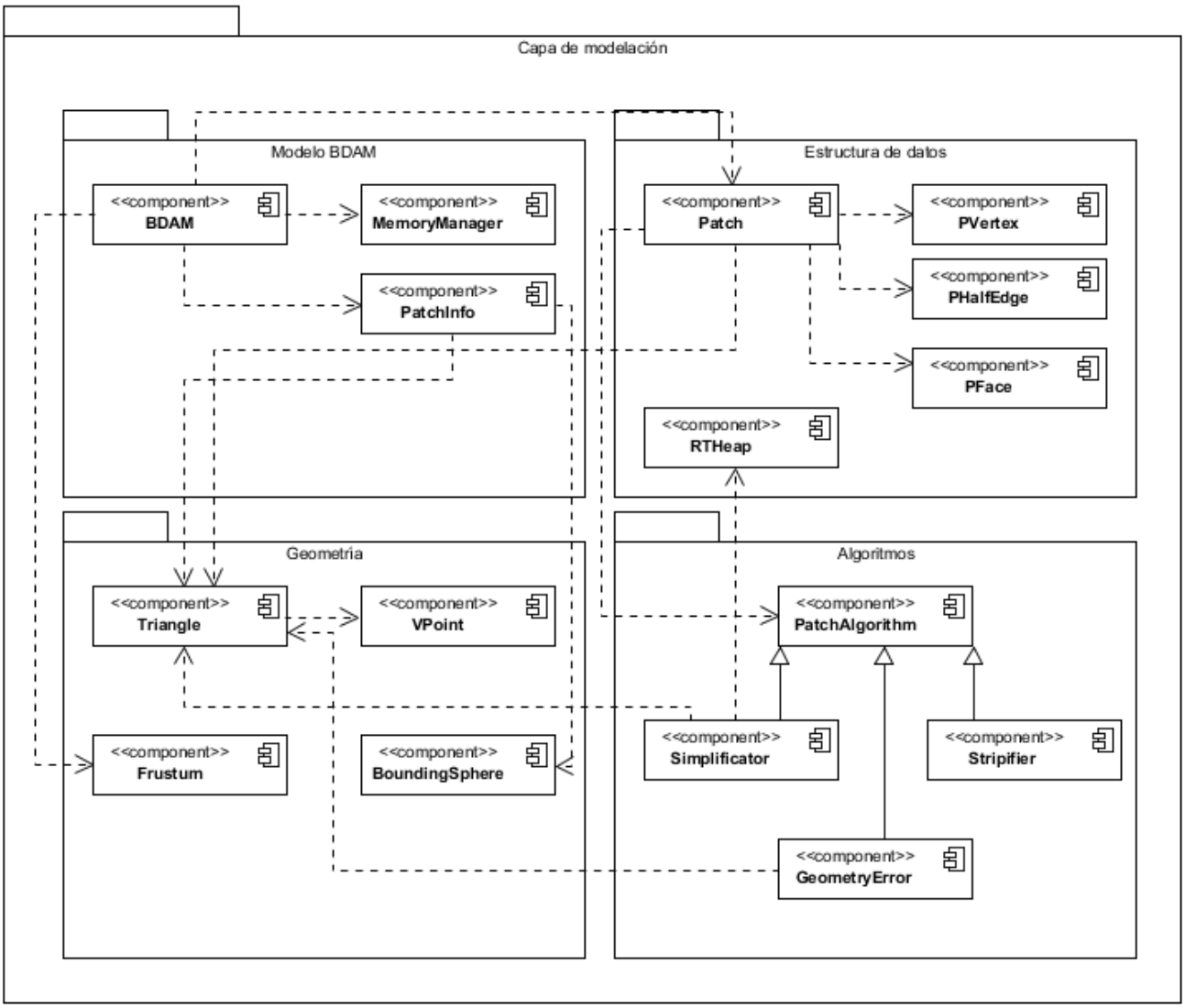

Fig. 12 Diagrama de componentes del sistema (Elaboración propia).

- Stripifier: Componente que implementa el algoritmo SGI para la generación de cadenas de triángulos.
- Simplificator: Componente utilizado para simplificar la cantidad de triángulos de la TIN de un nodo del bintree usando el algoritmo VolSimp.
- GeometryError: Mediante este componente se obtiene el error geométrico luego de la simplificación de un Patch.

Modelo: Este paquete contiene los componentes responsables de realizar las etapas del modelo BDAM, a continuación se muestran sus componentes.

- BDAM: Realiza todas las operaciones referentes al modelo BDAM.
- MemoryManager: Hace posible el uso de la memoria externa durante las operaciones del modelo BDAM.
- PatchInfo: Contiene la información correspondiente a cada nodo del bintree.

Geometría: En este paquete radican los componentes que gestionan la información referente a la geometría del modelo.

- Triangle: Componente que gestiona las operaciones referentes a los triángulos en tres dimensiones.
- VPoint: Representa un punto en tres dimensiones y permite al uso de operaciones vectoriales sobre este.
- Frustum: Representa el campo de visión de la cámara mediante una figura geométrica en forma de pirámide truncada, se utiliza durante la etapa de extracción de la triangulación para eliminar los objetos que están fuera del campo de visión, este componente proviene del Visor de Terrenos.
- BoundingSphere: Este componente representa una esfera delimitadora, es utilizado para la detección de colisiones y para la proyección de errores en la pantalla.

3.5 Pruebas realizadas al modelo

Las pruebas en un sistema informático tienen dos objetivos fundamentales:

1. Demostrar al desarrollador y al cliente que el software cumple sus requerimientos.
2. Descubrir situaciones en las cuales el software se comporta de manera incorrecta, no deseada o no cumple con sus especificaciones.

(Pressman, 2010) define cuatro niveles de abstracción para las pruebas realizadas al software durante el desarrollo del mismo.

1. Pruebas unitarias, donde unidades individuales del programa o clases de objetos son probadas.
2. Pruebas de Integración, donde varias unidades son integradas para crear componentes compuestos, esta se centra en el diseño y en la arquitectura del sistema.
3. Pruebas de validación, donde los requisitos establecidos durante el diseño son validados.
4. Pruebas de sistema, en esta el software y los otros elementos del sistema se prueban como un todo.

Entre los métodos de pruebas más utilizados se encuentran las pruebas de caja blanca, estas son una filosofía para el diseño de juegos de datos centrada en los detalles procedimentales del código, los juegos de datos para este tipo de prueba son diseñados con el objetivo de examinar cada uno de los posibles flujos de ejecución del programa. Entre las técnicas utilizadas para las pruebas de caja blanca se encuentra la técnica del camino básico, esta permite obtener una medida de la complejidad lógica del diseño de los procedimientos y utilizar esta medida como guía para derivar un conjunto de caminos de ejecución, los juegos de datos diseñados con el propósito de ejecutar dichos caminos garantizan la ejecución de cada sentencia en el programa al menos una vez (Pressman, 2010).

En la presente investigación se realizaron pruebas unitarias utilizando el método de pruebas de caja blanca y pruebas de integración utilizando el método de pruebas de integración basada en uso descrito en (Pressman, 2010).

3.5.1 Aplicación de las pruebas unitarias

El modelo fue separado en unidades para ser probadas cada una individualmente, a cada clase se le asignó una unidad. El método privado *preprocessMatrix*, asociado al requisito funcional "Construir BDAM" de la clase BDAM, fue escogido para mostrar el uso de la técnica del camino básico durante la etapa de pruebas unitarias. Este método transforma las dimensiones del modelo digital de elevación de entrada agregando valores nulos para completar en caso necesario. En el siguiente código se muestran la asignación de los nodos del grafo de flujo a las sentencias.

```
1    private List<float> preprocessMatrix(List<float> M,   int nrows,   int ncols, float
2    noData)
3    {
4        _tileSize = 1;
5        while (_tileSize + 1 < Math.Max(ncols, nrows)) ------------→(1)
6        {
7             _tileSize <<= 1; ------------------------------------→(2)
8        }
9        _tileSize++;
10       List<float> mat = new List<float>();
11       for (int cell = 0; cell < _tileSize * _tileSize; cell++) -->(3)
12            {
13               int i = cell / _tileSize;
14               int j = cell % _tileSize;
15               if (i < nrows && j < ncols) ----------------------→(4)(5)
16               {
17                   mat.Add(M[i * _tileSize + j]); ----------------→(6)
18               }
19               else
20               {
21                   mat.Add(noData); ------------------------------→(7)
```

```
22              }
23          }
24      return mat; ----------------------------------------------->(8)
25  }
```

La figura 13 muestra el grafo de flujo correspondiente al método *preprocessMatrix* mencionado anteriormente. La cantidad de juegos de datos necesarios para ejecutar todas las sentencias posibles es al menos la complejidad ciclomática del grafo del flujo. El valor de la complejidad ciclomática $V(G)$ del grafo de flujo G se obtiene mediante la ecuación:

$$V(G) = E - N + 2 \tag{4}$$

donde E y N definen la cantidad de aristas y la cantidad de nodos respectivamente de G.

Utilizando la ecuación 4 se obtuvo como resultado que la complejidad ciclomática del grafo de flujo del método *preprocessMatrix* es cinco. Con este propósito se diseñaron cinco juegos de datos para cubrir todos los caminos posibles en el grafo.

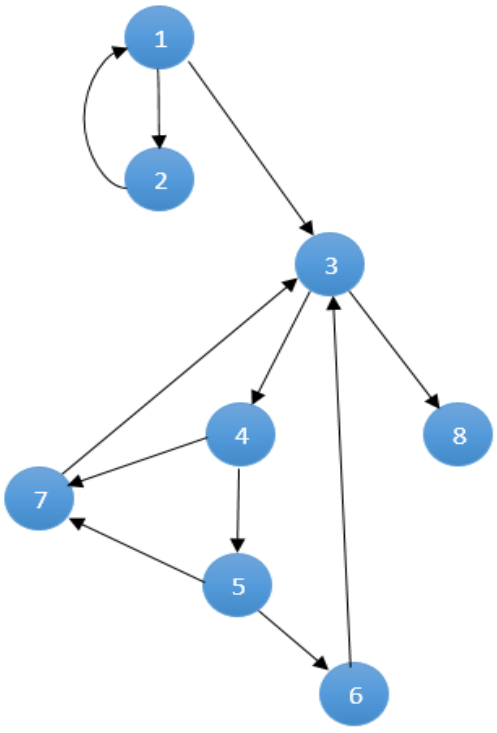

Fig. 13 Grafo de flujo del método preprocessMatrix (Elaboración propia).

En la tabla 7 se reflejan los 5 caminos básicos posibles dado la complejidad ciclomática obtenida.

Tabla. 7 Caminos del grafo del flujo

No.	Camino
1	1-3-8
2	1-2-1-3-8
3	1-3-4-7-3-8
4	1-3-4-5-6-3-8
5	1-3-4-5-7-3-8

Luego de la obtención de la cantidad de caminos básicos del grafo de flujo se procedió a la confección de los juegos de datos. La tabla 8 muestra el juego de datos correspondiente al cuarto camino básico. El resto de los casos de pruebas puede consultarse en los anexos.

Tabla. 8 Caso de prueba para el camino básico 4.

Camino	Descripción	Entrada	Resultado esperado
1-3-4-5-6-3-8	La matriz tiene como cantidad de filas y cantidad de columnas valores de la forma $2^k + 1$, de esta forma la condición 4 y 5 siempre se cumplen.	nrows = 129 ncols = 129 nodata = -9999 M = arreglo con todos los elementos iguales a 1	La matrix se mantiene igual, no se le agregan valores nulos.

Se ejecutaron cuatro iteraciones de las pruebas unitarias, en la figura 14 se muestran los resultados de las mismas en cada una de las etapas del modelo BDAM.

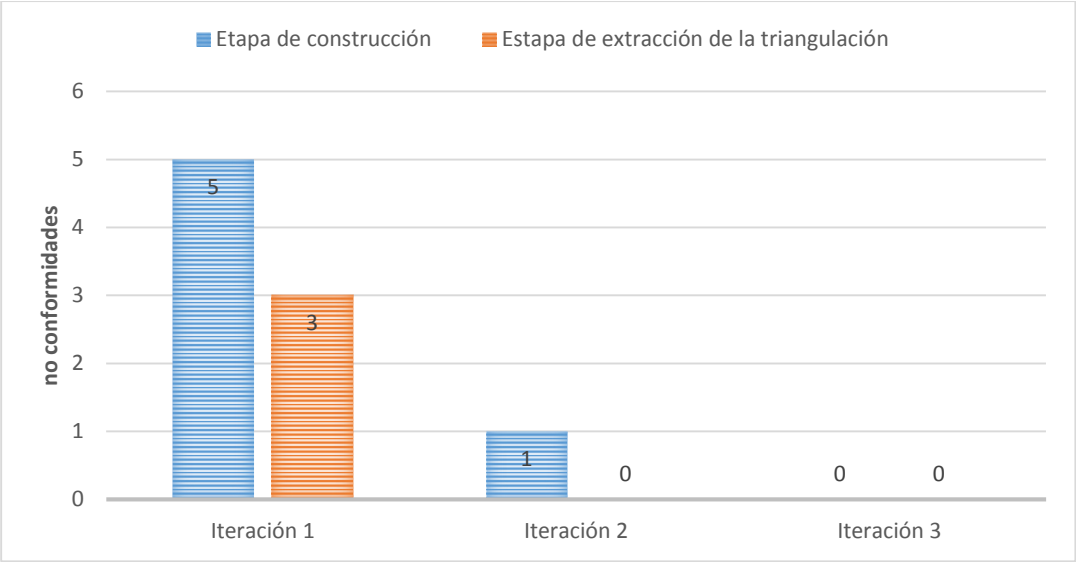

Fig. 14 Resultado de las pruebas unitarias (Elaboración propia)

Al finalizar cada iteración, se procedió a la corrección de las no conformidades, en la etapa de construcción fueron detectadas cinco no conformidades durante la primera iteración, de estas fueron resueltas correctamente el 80% al finalizar la primera iteración y el 100% al finalizar la segunda iteración. En la etapa de extracción de la triangulación fueron detectadas tres no conformidades durante la primera iteración, de estas todas fueron resueltas correctamente.

3.5.2 Aplicación de las pruebas de integración

Las pruebas de integración basadas en uso (Pressman, 2010) se definen en el contexto de la programación orientada a objetos, estas tienen comienzo con la realización de pruebas a las clases que tienen muy poca o ninguna dependencia, una vez probadas estas clases se le realizan pruebas a las clases que dependen de estas y así sucesivamente, esta secuencia de capas resultantes se define hasta que el sistema completo haya sido testeado.

Las clases que representan la primera capa sometida a pruebas de integración son MemoryManager, VPoint, PFace, PHalfEdge, Stripifier, RTHeap y BoundingSphere, estas clases no utilizan dependencias de ninguna de las otras clases.

La segunda capa contiene solamente las clases Triangle y PVertex, ambas dependientes de la clase VPoint.

La tercera capa contiene las siguientes clases:

- PatchInfo, depende de las clases Triangle y PatchAlgorithm
- Simplificator, depende de las clases Triangle y RTheap.
- GeometryError, depende de la clase Triangle.

La última capa contiene solamente la clase BDAM, esta depende de las clases PatchInfo, MemoryManager y Patch.

Las pruebas de integración fueron realizadas a cada una de las relaciones de las clases, con este objetivo fueron creados *logs* en cada momento que una clase utilizaba instancias de sus dependencias, el objetivo principal de estas pruebas fue la revisión de las relaciones entre las distintas clases y el uso correcto de las mismas. Se realizaron dos iteraciones de las pruebas de integración, en la primera iteración se detectó una no conformidad en la clase Simplificator que no inicializaba correctamente la instancia de la clase RTHeap, esta fue corregida agregando el operador *new* para invocar el constructor de la misma. La segunda iteración no arrojó ninguna no conformidad, de esta forma se concluyó con las pruebas de integración.

3.6 Validación del Modelo

Con el objetivo de realizar una comparación con respecto a la cantidad de primitivas gráficas generadas entre el Visor de Terrenos usando *quadtree* restringido y el modelo BDAM se realizaron un conjunto de experimentos. La implementación actual del *quadtree* restringido utiliza la métrica de error espacio-objeto para la selección del nivel de detalle, debido a esto fue necesario utilizar esta misma métrica en el modelo BDAM para lograr equidad en las comparaciones. En la tabla 9 se muestran el nombre y las dimensiones[4] de los juegos de datos utilizados, estos contienen información variada de los relieves de distintas porciones de la superficie terrestre y se encuentran disponibles para la comunidad científica en el enlace web http://vterrain.org/BT/. La variable resolución define la tolerancia con respecto al error espacio-objeto en metros. Se definieron además las variables “Millones de triángulos por trama” (MTPT) y “Millones de índices por trama” (MIPT) para medir la cantidad de triángulos generados y la cantidad de índices necesarios para la visualización de los triángulos en el Visor de Terrenos respectivamente. Los índices en la presente solución representan las cadenas de triángulos a diferencia de la implementación del *quadtree* restringido en el Visor de Terrenos que utiliza abanicos de triángulos (Shreiner et al., 2013) con el mismo propósito.

[4] En las dimensiones 1K representa 1024 unidades

Tabla. 9 Juegos de datos utilizados en los experimentos

Juego de datos	Dimensiones	Resolución
Big Island	4k x 4k	20
Kauai	4k x 4k	30
Costa Rica	2k x 2k	30
Baia Mare	2k x 2k	20

En las gráficas de la figuras 15 y 16 se presentan a modo de comparación la cantidad de primitivas gráficas generadas por el *quadtree* restringido y el modelo BDAM para distintos juegos de datos y resoluciones. Los resultados mostrados anteriormente evidencian la disminución de la cantidad de primitivas gráficas del modelo BDAM con respecto al *quadtree* restringido para una misma resolución. En la figura 17 se puede observar la diferencia visual entre la cantidad de triángulos generada por ambos modelos para un mismo umbral de error.

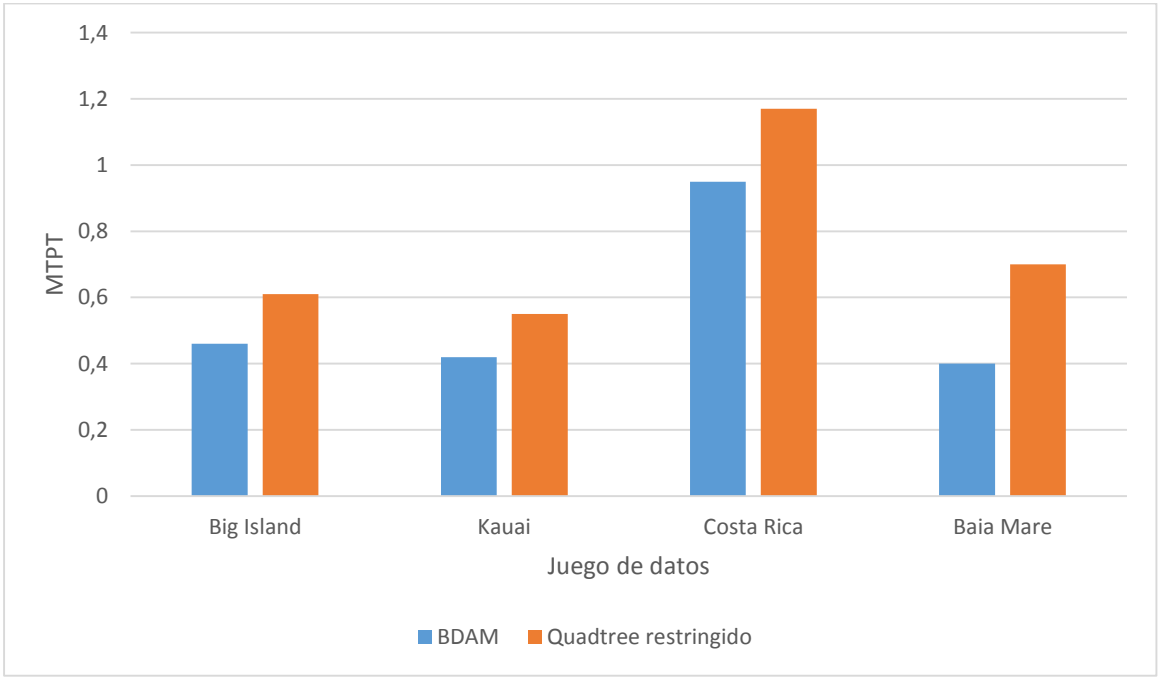

Fig. 15 Comparación de la cantidad de millones de triángulos por trama (MTPT) generadas por ambos modelos en el Visor de Terrenos (elaboración propia).

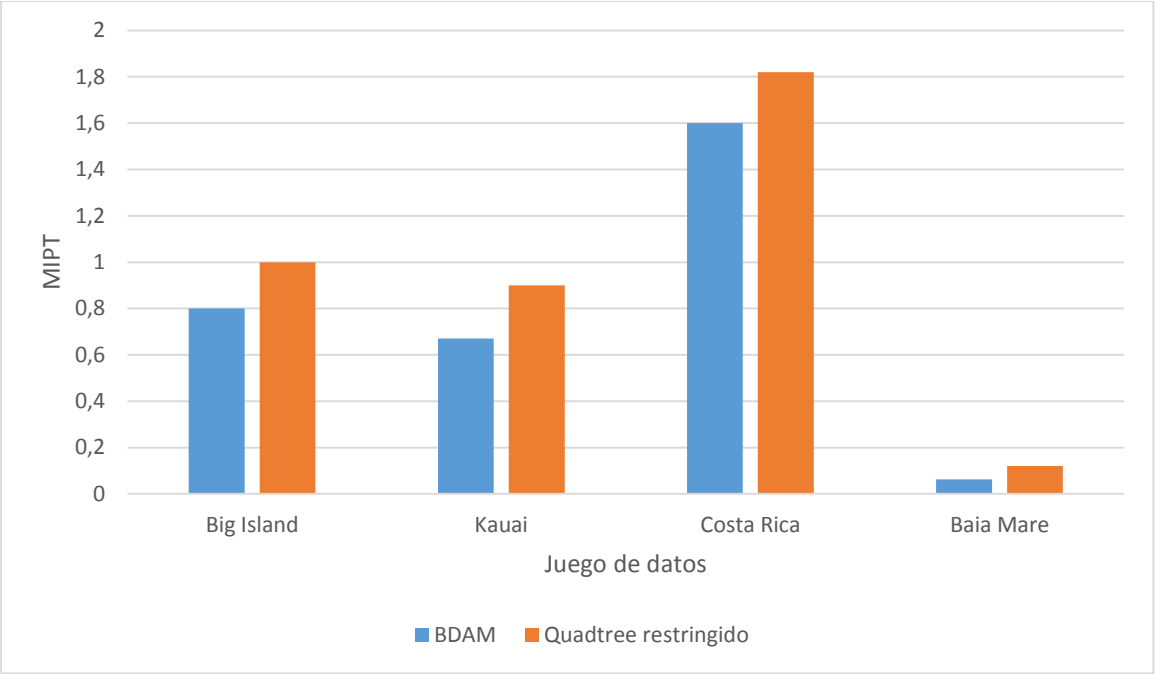

Fig. 16 Comparación de la cantidad de millones de índices por trama (MIPT) generada para la representación de los triángulos por ambos modelos en el Visor de Terrenos (Elaboración propia).

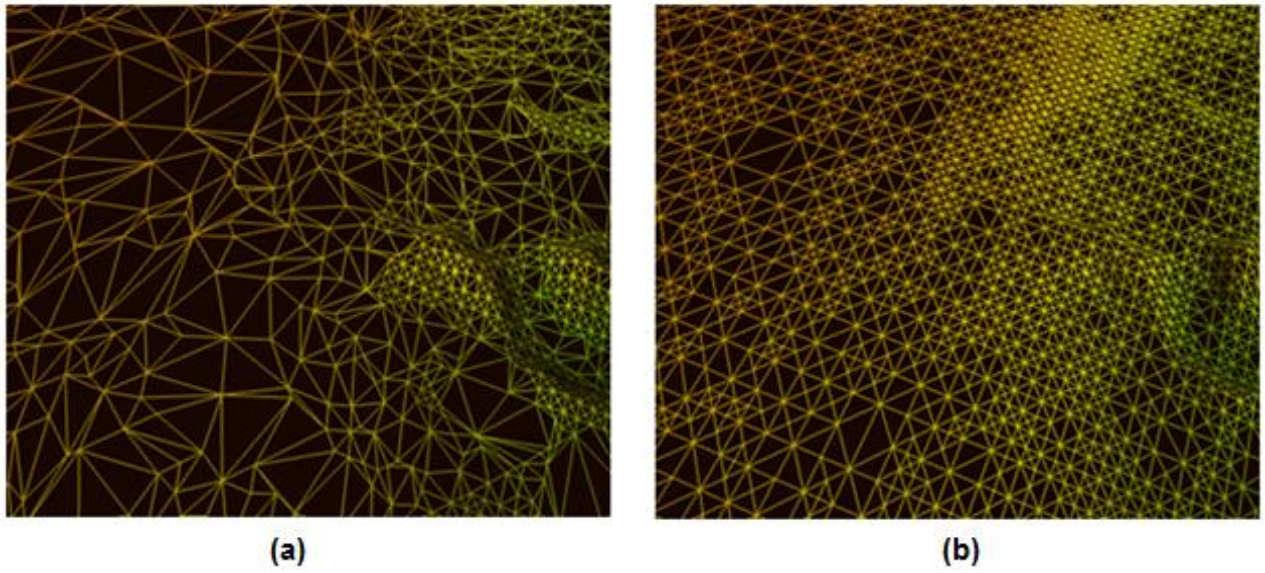

Fig. 17 Comparación visual de la cantidad de triángulos generadas por el modelo BDAM (a) y el *quadtree* restringido (b) en el juego de datos Big Island para un umbral de error espacio-objeto de 20 metros (Elaboración propia).

3.6.1 Aplicación de la métrica de error espacio-pantalla para la selección del nivel de detalle

Una de las nuevas funcionalidades agregadas al Visor de Terrenos como resultado de la presente investigación es el uso de la métrica de error espacio-pantalla para la selección del nivel de detalle durante la etapa de extracción de la triangulación del modelo BDAM. Con el objetivo de medir el

rendimiento del modelo implementado en la presente investigación se realizó un recorrido sobre el terreno representado por el juego de datos Big Island con dimensiones de 4k x 4k puntos y una tolerancia con respecto al error espacio-pantalla de 1 píxel. Se definieron las variables “Tramas por segundo” (TPS) y “Millones de triángulos por segundo” (MTPS) para medir la velocidad de visualización del modelo. El hardware utilizado para la realización del experimento presenta las siguientes prestaciones:

Procesador: DualCore Intel Core i3, 3400Mhz.

Memoria RAM: 4GB DDR3.

Tarjeta gráfica: Intel HD Graphic.

La duración del recorrido fue de aproximadamente 20 segundos, los resultados del mismo se muestran en la gráfica de la figura 18. La velocidad promedio con respecto a las tramas por segundo alcanzada durante la visualización fue aproximadamente 29.6 TPS con valores cercanos a las 35 TPS, teniendo en cuenta que la mínima cantidad de tramas por segundos necesarias para una animación fluida es de 30 TPS (Microsoft, 2003) se puede concluir que el modelo implementado presenta un buen desempeño en cuanto a la velocidad de visualización.

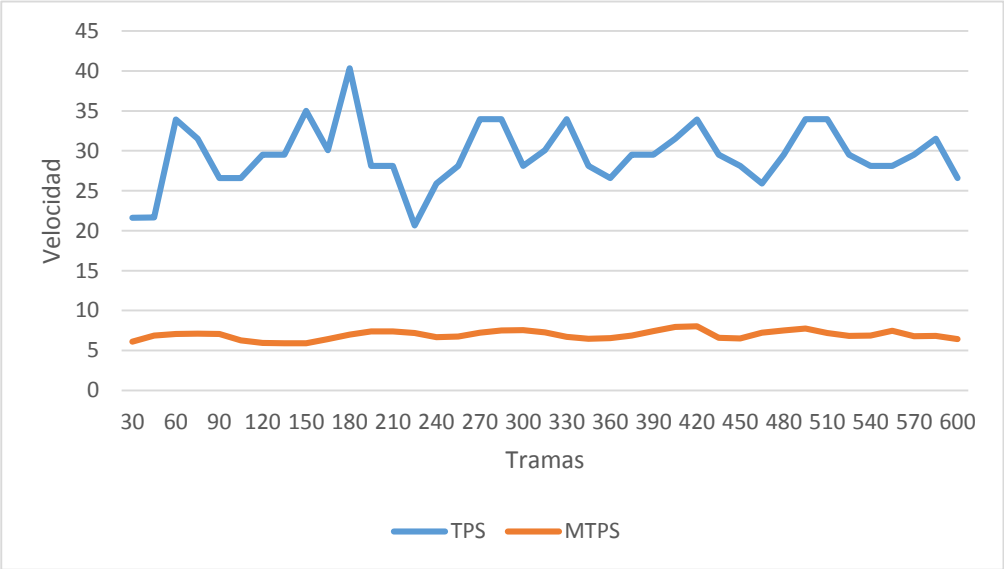

Fig. 18 Rendimiento del modelo BDAM para el juego de datos Big Island con umbral de error espacio-pantalla de 1 píxel (Elaboración propia).

3.6.2 Comparación de la calidad visual entre ambos modelos en el Visor de Terrenos

Con el fin de comparar la calidad visual de las triangulaciones generadas por la presente solución con respecto a la implementación del *quadtree* restringido en el Visor de Terrenos, se representó el modelo

digital de elevación correspondiente al juego de datos Kauai con una misma cantidad de triángulos usando ambos modelos (ver figura 19). En la representación del modelo BDAM (figura 19a) el uso de la métrica de error espacio-pantalla permitió conservar mayor cantidad de detalles del terreno que la representación del *quadtree* restringido (ver figura 19b) con la métrica de error espacio-objeto.

El *quadtree* restringido divide el terreno en bloques para lograr escalabilidad durante la construcción y visualización de los MDEs, cada uno de estos bloques es procesado individualmente durante la etapa de visualización donde se realiza el cálculo de los vectores normales para determinar la iluminación del terreno, en los bordes de dichos bloques se evidencia una diferencia notable en la iluminación debido al procesamiento individual de los mismos. En la figura 19b se muestra el fenómeno antes mencionado en el Visor de Terrenos, como resultado adicional de la presente investigación se le dio solución a este problema mediante el cálculo de los vectores normales durante la etapa de construcción del modelo BDAM.

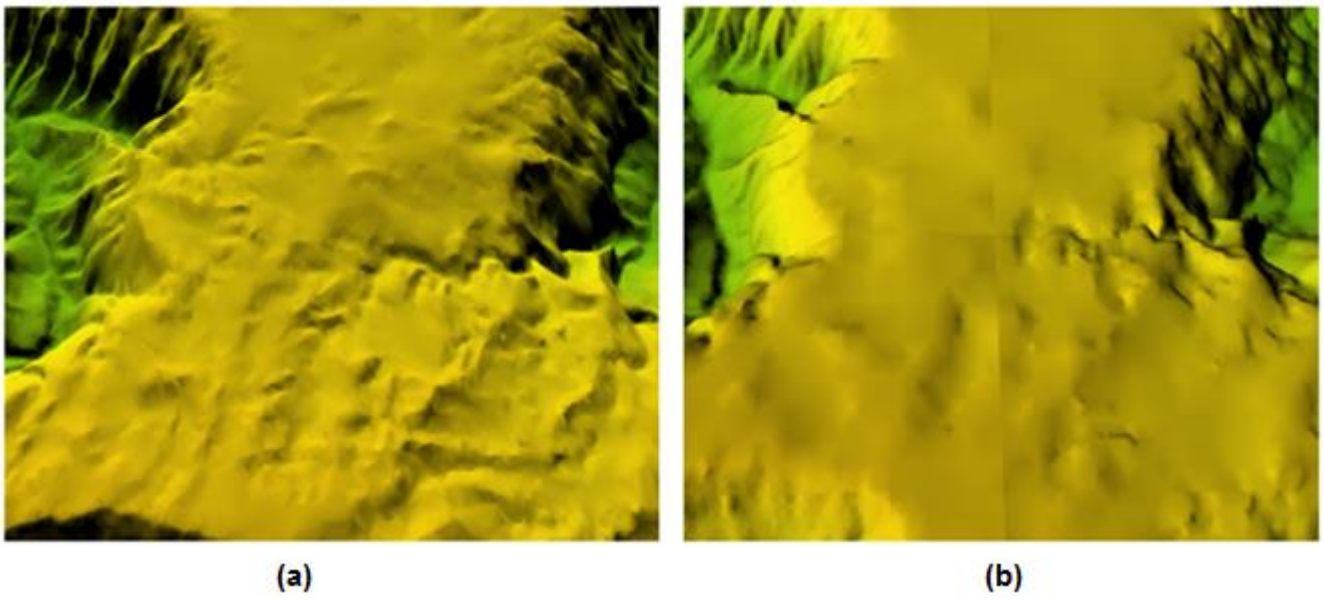

Fig. 19 Comparación de la calidad visual entre el modelo BDAM (a) y el *quadtree* restringido (b) con una cantidad de triángulos igual a 500K usando el juego de datos Kauai (Elaboración propia)

Conclusiones del Capítulo

Durante el desarrollo del presente capítulo se definieron los estándares de codificación utilizados en la solución propuesta mediante los cuales se logró homogeneidad con la plataforma .Net. Fueron mostrados los principales detalles de implementación de cada una de las etapas definidas en el modelo BDAM así como la relación entre los distintos componentes definidos para llevar a cabo su

implementación. Se realizaron diversos experimentos donde se comparó la cantidad de primitivas gráficas generada por el *quadtree* restringido con la cantidad generada por el modelo implementado, durante el desarrollo de los mismos se evidenció una reducción de la cantidad de primitivas gráficas por parte del modelo BDAM así como un incremento en la calidad de visualización. Además se le realizaron pruebas unitarias y pruebas de integración a la implementación del modelo, durante la ejecución de las mismas se corrigieron todas las no conformidades que surgieron.

Conclusiones

Se implementó un modelo híbrido para la representación multi-resolución de modelos digitales de elevación en tres dimensiones que fue integrado al Visor de Terrenos donde se evidenciaron los siguientes resultados:

- El estudio de la literatura relacionada a los modelos multi-resolución para la visualización de terrenos permitió la selección del modelo BDAM para la representación de MDEs en el Visor de Terrenos.
- A partir del análisis y diseño del modelo BDAM se obtuvieron un conjunto de artefactos que facilitaron el desarrollo del mismo.
- El modelo BDAM redujo la cantidad de primitivas gráficas con respecto al *quadtree* restringido para un mismo umbral de error.
- El algoritmo VolSimp seleccionado para la simplificación de las mallas triangulares presentó un buen desempeño con respecto a la calidad de las aproximaciones.
- El algoritmo SGI utilizado para la obtención de cadenas de triángulos redujo la cantidad de información a enviar al Visor de Terrenos para la visualización de las triangulaciones.
- Utilizando la memoria externa el sistema es dotado de escalabilidad logrando procesar terrenos de dimensiones superiores a 4k X 4k.
- La métrica de error espacio-pantalla permitió dotar de menos detalle las regiones más alejadas al punto de visión de la cámara, lográndose de esta forma representar los terrenos con menor cantidad de triángulos.

Recomendaciones

Se recomienda para un trabajo futuro:

- Utilizar paginado en demanda para el manejo de la memoria mediante el uso de técnicas de mapeo de memoria similares a las de los sistemas operativos que permitan un acceso más eficiente a los datos almacenados en la memoria externa.
- Utilizar técnicas de programación en paralelo para optimizar la construcción del modelo con el fin de reducir el tiempo de preprocesamiento.
- Utilizar técnicas de compresión para disminuir la cantidad de información utilizada para el almacenamiento en la memoria externa.

Referencias Bibliográficas

AKELEY, K.; HAEBERLI, P.; BURNS, D. The tomesh. C: C Program on SGI Developer's Toolbox, 1990.

ANDERSEN, Kasper Amstrup; BAY, Christian. A survey of algorithms for construction of optimal Heterogeneous Bounding Volume Hierarchies. Saatavilla pdf-muodossa http://image. diku. dk/projects/media/christian. bay. kasper. andersen. B, 2006, vol. 6.

BALDUINO, Ricardo. Introduction to OpenUP (Open Unified Process). [en línea] www.eclipse.org, 2007 [Consultado el: 2 de junio del 2015]. Disponible en: http://www.eclipse.org/epf/general/OpenUP.pdf

BAUMANN, Konstantin, et al. A hybrid, hierarchical data structure for real-time terrain visualization. En Computer Graphics International, 1999. Proceedings. IEEE, 1999. p. 85-92.

BOTSCH, Mario, et al. Openmesh-a generic and efficient polygon mesh data structure. 2002.

CIGNONI, Paolo; ROCCHINI, Claudio; SCOPIGNO, Roberto. Metro: measuring error on simplified surfaces. En Computer Graphics Forum. Blackwell Publishers, 1998. p. 167-174.

CIGNONI, Paolo, et al. BDAM—Batched Dynamic Adaptive Meshes for high performance terrain visualization. En *Computer Graphics Forum*. Blackwell Publishing, Inc, 2003. p. 505-514.

COHEN, Jonathan D.; MANOCHA, Dinesh. Model simplification. Visualization Handbook, Elsevier Butterworth-Heinemann, Oxford, 2005, p. 393-411.

DE FLORIANI, Leila; KOBBELT, Leif; PUPPO, Enrico. A survey on data structures for level-of-detail models. En *Advances in multiresolution for geometric modelling*. Springer Berlin Heidelberg, 2005. p. 49-74.

EVANS, William; KIRKPATRICK, David; TOWNSEND, Gregg. Right-triangulated irregular networks. Algorithmica, 2001, vol. 30, no 2, p. 264-286.

GARLAND, Michael; HECKBERT, Paul S. Surface simplification using quadric error metrics. En *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997. p. 209-216.

GOBBETTI, Enrico, et al. C-BDAM–Compressed Batched Dynamic Adaptive Meshes for Terrain Rendering. En Computer Graphics Forum. Blackwell Publishing, Inc, 2006. p. 333-342.

HAO, Aimin; TANG, Shaopeng; JIA, Lintao. A method for terrain rendering real-time based on two-level model. En Digital Media and its Application in Museum & Heritages, Second Workshop on. IEEE, 2007. p. 189-194.

HUNT, Lance. *Coding Standards for .NET.* 2007.

LARMAN, Craig. Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development, 3/e. Pearson Education India, 2005.

LARIO, Roberto; PAJAROLA, Renato; TIRADO, Francisco. Hyperblock-quadtin: Hyper-block quadtree based triangulated irregular networks. En Proceedings IASTED Invernational Conference on Visualization, Imaging and Image Processing (VIIP). 2003. p. 733-738.

LAWDER, Jonathan K.; KING, Peter JH. Using space-filling curves for multi-dimensional indexing. En Advances in Databases. Springer Berlin Heidelberg, 2000. p. 20-35.

LI, Zhilin; ZHU, Christopher; GOLD, Chris. Digital terrain modeling: principles and methodology. CRC press, 2010.

LINDSTROM, Lowell; JEFFRIES, Ron. Extreme programming and agile software development methodologies. Information systems management, 2004, vol. 21, no 3, p. 41-52.

MICROSOFT. Understanding Frames Per Second (FPS). [en línea]. 2003 [Consultado el: 1 de junio del 2015]. Disponible en: https://support.microsoft.com/en-us/kb/269068.

NG, Kok-Why; LOW, Zhi-Wen. Simplification of 3D Triangular Mesh for Level of Detail Computation. En *Computer Graphics, Imaging and Visualization (CGIV), 2014 11th International Conference on*. IEEE, 2014. p. 11-16.

PAJAROLA, Renato. Large scale terrain visualization using the restricted quadtree triangulation. En *Visualization'98. Proceedings*. IEEE, 1998. p. 19-26.

PAJAROLA, Renato; ANTONIJUAN, Marc; LARIO, Roberto. Quadtin: Quadtree based triangulated irregular networks. En *Proceedings of the conference on Visualization'02*. IEEE Computer Society, 2002. p. 395-402.

PAJAROLA, Renato; GOBBETTI, Enrico. Survey of semi-regular multiresolution models for interactive terrain rendering. The Visual Computer, 2007, vol. 23, no 8, p. 583-605.

PRESSMAN, Roger S. Software engineering: a practitioner's approach 7/e. Mc Graw Hill, 2010.

HOPPE, Hugues. Progressive meshes. En *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996. p. 99-108.

HUSSAIN, Muhammad. Volume and Normal Field Based Simplification of Polygonal Models. *J. Inf. Sci. Eng.*, 2013, vol. 29, no 2, p. 267-279.

SCHAMBERGER, Stefan; WIERUM, Jens-Michael. Partitioning finite element meshes using space-filling curves. Future Generation Computer Systems, 2005, vol. 21, no 5, p. 759-766.

SHREINER, Dave, et al. OpenGL programming guide: The Official guide to learning OpenGL, version 4.3. Addison-Wesley, 2013.

STEWART J. Tunneling for Triangle Strips in Continuous Level-of-Detail Meshes. In Graphics Interface, pages 91–100, 2001.

VAN KAICK, Oliver Matias; DA SILVA, Murilo Vicente Gonçalves; PEDRINI, Hélio. Efficient generation of triangle strips from triangulated meshes. 2004.

VAN KAICK, Oliver Matias; PEDRINI, Hélio. A comparative evaluation of metrics for fast mesh simplification. En *Computer Graphics Forum*. Blackwell Publishing Ltd, 2006. p. 197-210.

VANEEK Petr. Triangle Strips for Fast Rendering. Doctoral Thesis in Computer Science. Faculty of Applied Sciences, University of West Bohemia. Pilsen, 2005.

VLISSIDES, John, et al. Design patterns: Elements of reusable object-oriented software. Reading: Addison-Wesley, 1995, vol. 49, p. 120.

WEISS, Kenneth; DE FLORIANI, Leila. Simplex and diamond hierarchies: Models and applications. En *Computer Graphics Forum*. Blackwell Publishing Ltd, 2011. p. 2127-2155.

XIANG, Xinyu; HELD, Martin; MITCHELL, Joseph SB. Fast and effective stripification of polygonal surface models. En Proceedings of the 1999 symposium on Interactive 3D graphics. ACM, 1999. p. 71-78.

Glosario de Términos

Frustum: El frustum es una porción de una figura geométrica (usualmente un cono o una pirámide) comprendida entre dos planos paralelos. En el campo de los gráficos por computadora el frustum se usa como representación del volumen de visualización de una cámara. En este caso se usa un frustum piramidal con seis planos, los cuales son:

- Plano de recorte lejano (far).
- Plano de recorte cercano ($near$).
- Plano superior (top).
- Plano inferior ($bottom$).
- Plano izquierdo ($left$).
- Plano derecho ($right$).

TIN: Red Irregular de Triángulos como indica su traducción al español (del inglés Triangulated Irregular Network) es una estructura de datos espacial que refleja las relaciones existentes entre los vértices, aristas y triángulos de una malla triangular.

GIS: Un sistema de información geográfica (GIS por sus siglas en inglés) es una integración organizada de hardware, software y datos geográficos diseñado para capturar, almacenar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada con el fin de resolver problemas complejos de planificación y gestión geográfica.

Anexos

Anexo A: Juego de datos del método preprocessMatrix

Tabla. 10 Juegos de datos para el método preprocessMatrix.

Camino	Descripción	Entrada	Resultado esperado
1-3-8	Este camino nunca se ejecuta dado que 3 siempre se cumple.	No hay entrada para este caso de prueba.	No se espera ningún resultado para este caso.
1-3-4-5-7-3-8	Ambos valores cantidad de filas y cantidad de columnas son distintos y no están en la forma $2^k + 1$, cumpliéndose las condiciones 4 y 5 cuando *i* y *j* no están en el rango de la matrix original.	nrows = 100 ncols = 110 nodata = -9999 M = arreglo con todos los elementos iguales a 100	La matrix agrega nuevas filas y nuevas columnas con valores nulos.
1-2-1-3-8	Este camino nunca se ejecuta dado que 3 siempre se cumple.	No hay entrada para este caso de prueba.	No se espera ningún resultado para este caso.
1-3-4-5-6-3-8	La matriz tiene como cantidad de filas y cantidad de columnas valores de la forma $2^k + 1$, de esta forma la condición 4 y 5 siempre se cumplen.	nrows = 129 ncols = 129 nodata = -9999 M = arreglo con todos los elementos iguales a 1	La matrix se mantiene igual, no se le agregan valores nulos.
1-3-4-7-3-8	La matriz tiene solamente como cantidad de columnas un valor de la forma $2^k + 1$, de esta forma la condición 4 se cumple y la 5 nunca se llega a cumplir.	nrows = 120 ncols = 129 nodata = -9999 M = arreglo con todos los elementos iguales a 50	La matrix agrega nuevas filas con valores nulos.

Anexo B: Vistas del modelo BDAM

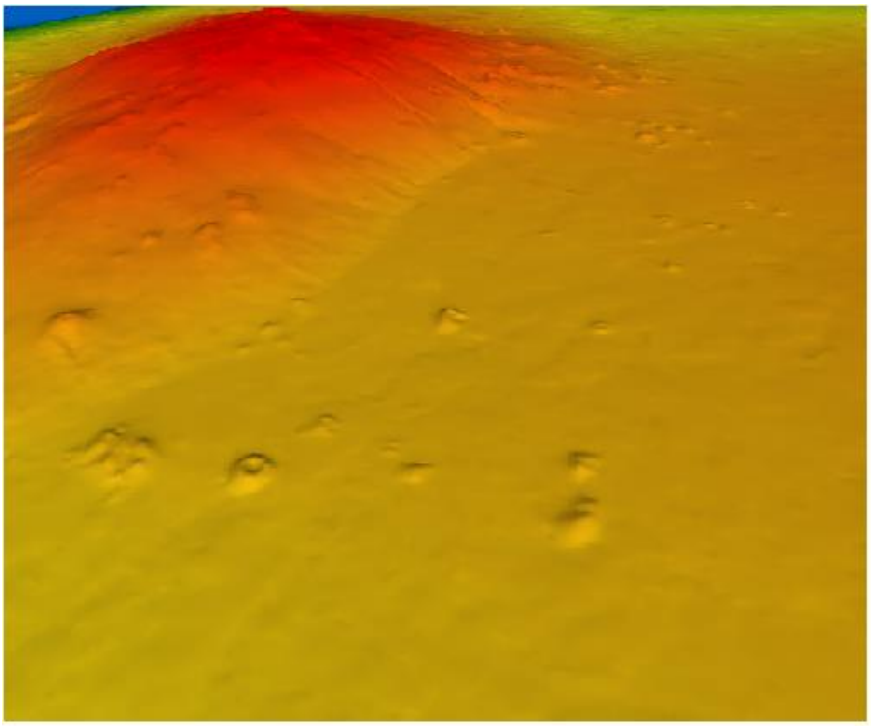

Fig. 20 Juego de datos Big Island 4k x 4k, con una tolerancia de 2 píxeles usando BDAM (Elaboración propia).

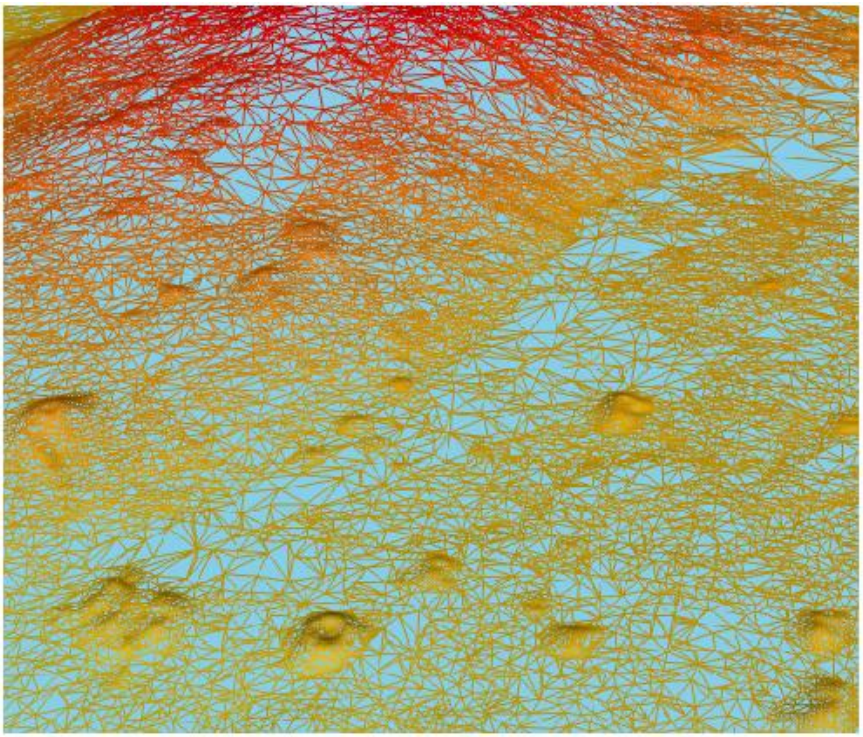

Fig. 21 Malla de Big Island 4k x 4k, con una tolerancia de 2 píxeles usando BDAM (Elaboración propia).