UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Departamento de Electrónica e Computación

PhD Thesis

REPRESENTATION TECHNIQUES FOR REAL-TIME VISUALIZATION OF HYBRID TERRAIN MODELS

Presented by:

Enrique González Paredes

PhD Advisors:

Montserrat Bóo Cepeda

Margarita Amor López

Santiago de Compostela, November 2013

Montserrat Bóo Cepeda, Profesora Titular de Universidad del Área de Arquitectura y Tecnología de Computadores de la Universidad de Santiago de Compostela

Margarita Amor López, Profesora Titular de Universidad del Área de Arquitectura y Tecnología de Computadores de la Universidad de A Coruña

Javier Díaz Bruguera, Profesor Catedrático de Universidad del Área de Arquitectura y Tecnología de Computadores de la Universidad de Santiago de Compostela

HACEN CONSTAR:

Que la memoria titulada **REPRESENTATION TECHNIQUES FOR REAL-TIME VISUALIZATION OF HYBRID TERRAIN MODELS** ha sido realizada por **D. Enrique González Paredes** bajo nuestra dirección en el Departamento de Electrónica e Computación de la Universidad de Santiago de Compostela, y constituye la Tesis que presenta para optar al título de Doctor.

Santiago de Compostela, 29 de noviembre de 2013

Montserrat Bóo Cepeda
Codirectora de la tesis

Margarita Amor López
Codirectora de la tesis

Javier Díaz Bruguera
Tutor de la tesis

Enrique González Paredes
Autor de la tesis

Resumen

El uso de modelos digitales del terreno y de bases de datos geográficas ha aumentado de forma considerable en los últimos años. Este crecimiento, debido principalmente al abaratamiento de los sensores y los avances en las tecnologías de adquisición de datos geográficos tales como LiDAR, ha motivado, a su vez, nuevos dominios de aplicación para la información geográfica. Herramientas de planificación del territorio, de gestión de infraestructuras, cartografía, sistemas de navegación global o aplicaciones de realidad virtual son algunos de los ejemplos más destacados.

Un modelo digital del terreno es una colección compleja y generalmente extensa de datos geográficos que almacena información topográfica y geométrica de un área de la superficie terrestre. Debido a sus grandes dimensiones y a la precisión utilizada, los modelos del terreno a menudo sobrepasan las capacidades de cálculo y almacenamiento de equipos convencionales. Por ello, es necesario usar técnicas eficientes y estructuras de datos avanzadas para su procesado, almacenamiento y visualización.

Las dos representaciones más utilizadas para almacenar los datos de elevación de un modelo digital del terreno son las mallas estructuradas cuadradas, o mallas regulares, y las mallas triangulares irregulares. Los modelos basados en mallas regulares son más sencillos de procesar. Sin embargo, requieren un número de muestras mucho mayor que los modelos irregulares ya que utilizan una densidad de muestreo fija en todo el modelo, lo cual es especialmente ineficiente para ciertas características del terreno, como áreas planas o superficies regulares. Para reducir su coste espacial, normalmente se almacena en cada muestra el valor de elevación del relieve y las coordenadas espaciales se deducen de su posición en la estructura de datos. Ello impide que se pueden representar algunas características geográficas peculiares como cuevas y acantilados o salientes en voladizo. Los modelos basados en mallas triangulares (*Triangulated Irregular Network* o TIN), al contrario, suelen ocupan menos espacio de almacenamiento

porque la frecuencia de muestreo y el tamaño de los triángulos se adapta a la irregularidad del terreno, permitiendo representar grandes áreas planas con muy pocas muestras. Sin embargo, esta mayor flexibilidad en la representación de las muestras requiere de estructuras de datos más complejas que penalizan la eficiencia de las operaciones de procesado o visualización.

Las técnicas de representación multirresolución se utilizan principalmente para conseguir visualizar en tiempo real modelos 3D de gran tamaño. Se basan en la generación de diferentes niveles de detalle para representar los modelos según las necesidades del proceso. Además, los diferentes niveles de detalle no se aplican de un modo global uniforme sino que se pueden asignar de forma independiente a cada zona del modelo, para refinar su representación en mayor o menor medida según los requisitos específicos de la operación. Durante la visualización interactiva, por ejemplo, el modelo se refina con mayor detalle en las zonas con mayor influencia en la apariencia final del modelo, normalmente las más cercanas al punto de vista, para obtener una imagen final sin una pérdida significativa de calidad en la apariencia visual de la escena. Los métodos más recientes y eficientes utilizan mallas regulares por la simplicidad y regularidad de sus estructuras de datos.

Una estrategia complementaria para optimizar la representación de grandes modelos del terreno consiste en utilizar modelos híbridos que combinen mallas regulares cuadradas y mallas irregulares de triángulos. Un tipo de modelo híbrido especialmente interesante es el que combina una gran malla base regular junto con mallas irregulares de alta resolución para las zonas que requieran de mayor detalle. La malla base sirve para representar una gran extensión del terreno a un nivel de detalle medio mientras que las mallas irregulares modelan zonas del terreno con accidentes geográficos complejos o estructuras artificiales como edificios y otras construcciones. De esta forma se produce una descripción geométrica más precisa de las características morfológicas más relevante del terreno, sin tener que aumentar la resolución general de toda la malla base.

Pese a sus ventajas teóricas, el uso de modelos híbridos no ha sido apenas estudiado hasta muy recientemente y apenas existen trabajos de investigación al respecto en el campo de la visualización interactiva de modelos del terreno. El algoritmo de *Mallado Híbrido* (HM) es una de las pocas propuestas recientes que exploran este camino, aunque es un trabajo principalmente teórico sin aplicación real.

El objetivo de la investigación realizada en esta tesis ha sido desarrollar métodos mejorados para la visualización en tiempo real de modelos híbridos del terreno. Las técnicas desarrolladas permiten integrar datos geográficos provenientes de fuentes heterogéneas y combinar

los beneficios propios de los modelos regulares y de los modelos irregulares. Todo ello sin necesidad de realizar un proceso de combinación intensivo que forzosamente modificaría los datos originales para integrar ambas mallas en un nuevo modelo. Estas técnicas son también compatibles con la aplicación de métodos de visualización con multirresolución, algo esencial para lograr una visualización interactiva tratándose de modelos del terreno de gran extensión.

Así pues, esta tesis presenta en primer lugar un detallado análisis del algoritmo HM, originalmente desarrollado por miembros del grupo de investigación y que sirve como punto de partida para este trabajo. El algoritmo HM genera un modelo híbrido mediante la combinación ya mencionada de un modelo base regular con una malla TIN de alta resolución que agrega detalles de alta resolución en áreas específicas del modelo. Esta combinación se obtiene mediante un proceso de triangulación adaptativa a nivel local entre los límites de las mallas regulares e irregulares. Los triángulos introducidos en el proceso cubren el área limítrofe entre ambas mallas de tal forma que el modelo híbrido resultante está perfectamente definido y no contiene huecos ni espacios vacíos en su interior.

Para obtener este modelo continuo es necesario, no sólo generar la malla de triangulación que une los bordes de las mallas regulares e irregulares entre sí, sino que también hace falta limitar el área de visualización de las mismas en las zonas en que se superponen. Es decir, la malla base regular, de menor resolución, no debe visualizarse en las zonas en las que la malla TIN de alta resolución está disponible. Para ello, las celdas de la malla regular se clasifican en tres tipos diferentes según su relación con la malla irregular: celdas *no cubiertas* (NC) por el TIN, *parcialmente cubiertas* (PC) y *completamente cubiertas* (CC). Las celdas NC no interactuan para nada con la malla irregular así que se visualizan normalmente. Las celdas CC, en cambio, se eliminarán del resultado final de la visualización ya que son sustituidas por la malla irregular de mayor calidad. Finalmente, las celdas PC son las celdas en las que se genera la triangulación adaptativa entre los modelos ya que contienen los vértices y aristas exteriores del TIN, que deben ser conectados con la malla regular. Para ello se genera la malla de triangulación entre el borde del TIN y las esquinas no cubiertas de la celda en cuestión.

Este proceso de triangulación adaptativa que une los bordes de ambas mallas está parcialmente precalculado, puesto que utiliza información generada en una fase previa a la visualización. Durante la visualización se accede a las estructuras de datos precalculadas, que codifican de forma sencilla y eficiente la información necesaria para generar la parte más compleja de la triangulación: la transformación del borde del TIN en un polígono localmente convexo dentro de cada celda regular. A partir de esa información y de unas reglas sencillas para completar-

la, se generan los triángulos necesarios para el nivel de detalle activo en la malla base en el momento de visualizarla. El proceso de triangulación es local porque se realiza en el interior de cada celda de la malla regular que se cruza con el borde del TIN, y es adaptativo ya que se amolda a los bordes de la malla regular que varían según el nivel de detalle activo en la malla base. Así pues, la información precomputada y codificada en las estructuras de datos del algoritmo se basa únicamente en el borde del TIN, ya que no puede depender de que el borde de la malla regular no varíe según las condiciones de la escena visualizada. Gracias a este enfoque adaptativo local, el algoritmo HM puede combinarse con diferentes procedimientos de triangulación y algoritmos de multirresolución, lo que lo convierte en un método potente y versátil.

El resultado final que se obtiene con este método es un modelo de alta calidad, ya que utiliza en todo momento la representación del terreno con mayor detalle disponible, perfectamente integrada con el resto del modelo. La importancia de los resultados conseguidos con este método es aún mayor considerando que no se han encontrado otros métodos de visualización similares que obtengan modelos híbridos de forma interactiva.

Tras el análisis detallado del algoritmo HM, se presentan a continuación dos propuestas diferentes para la visualización interactiva de modelos híbridos del terreno, basadas en la misma estrategia del algoritmo HM. Dicha estrategia, un proceso de triangulación local adaptativa entre los bordes de los modelos con diferente representación, es utilizada por las dos propuestas para generar un modelo híbrido sin discontinuidades, tal y como hace el algoritmo HM.

La primera propuesta utiliza un algoritmo estándar de triangulación de polígonos complejos, en este caso el algoritmo de triangulación incremental aleatorio de Seidel [74], para generar los triángulos de la malla de triangulación adaptativa entre los modelos de una forma directa y sencilla. Esta propuesta aporta una ventaja importante en cuanto a la simplicidad del procedimiento, especialmente en la etapa de preprocesado, que es mínima, ya que no utiliza ninguna estructura de datos precomputada para codificar información previa sobre la convexificación del borde del TIN. Durante la fase de visualización se utiliza el algoritmo de Seidel para generar la triangulación del polígono formado por los bordes de las mallas en el interior de la celda PC. Así, el polígono a triangular está formado por las esquinas de la celda PC, que son vértices de la malla regular, y los vértices del borde del TIN. Esta propuesta, aunque es sencilla y produce resultados de calidad, es menos eficiente en términos de rendimiento, ya que la malla de triangulación se genera completamente durante la visualización.

La segunda propuesta es una adaptación del algoritmo HM original a un *pipeline gráfico* con una arquitectura hardware convencional. El algoritmo original propone una nueva unidad hardware para descodificar fácilmente en la propia tarjeta gráfica la información codificada durante el preprocesado. Como esta propuesta teórica no ha llegado a implementarse en hardware, en esta tesis se ha evaluado la posibilidad de realizar una implementación software del algoritmo con una arquitectura convencional. Así pues, esta segunda propuesta implementa el algoritmo HM sustituyendo la unidad hardware propuesta por una etapa de descodificación de la información en la CPU. Los resultados obtenidos con esta segunda propuesta son similares en términos de calidad a los obtenidos con la propuesta anterior con el algoritmo de Seidel.Además, en un caso general la calidad es similar y el rendimiento de esta segunda propuesta es muy superior a la anterior, tal y como se muestra en la sección de resultados del capítulo 3.

A continuación, se presenta el método GPU-HM, un nuevo algoritmo paralelo de visualización de modelos híbridos. Este método utiliza el alto grado de paralelismo existente en los procesadores gráficos actuales o GPUs (*Graphics Processing Unit*) para mejorar el rendimiento de la visualización interactiva. En este caso, la CPU se libera del proceso de descodificación y generación de la malla de triangulación adaptativa de las celdas PC, que se realiza en la GPU. Así, la CPU se encarga de empaquetar los datos precomputados en estructuras de datos fácilmente accesibles desde los *shaders* de la GPU, y de enviar los identificadores de las celdas activas en cada momento para que sean teseladas en la tarjeta gráfica. Como la triangulación adaptativa se genera a nivel local para cada celda PC, el número de nuevos triángulos generados por celda no es demasiado elevado, así que todo el proceso de generación de geometría se puede realizar utilizando el *geometry shader*. Esta etapa de la GPU permite generar pequeñas cantidades de primitivas gráficas de una forma mucho más versátil que la más moderna unidad de teselado que incorporan actualmente los procesadores gráficos.

Esta propuesta basada en el *geometry shader* tiene varias consecuencias destacables. En primer lugar, al utilizar una unidad estándar del *pipeline gráfico* en lugar de usar técnicas de programación general en la GPU (GPGPU), es fácil de integrar con motores de visualización y sistemas gráficos que sean compatibles con la especificación de procesadores gráficos *Shader Model 4*. En segundo lugar, al no requerir las nuevas capacidades específicas de teselado introducidas en *Shader Model 5*, esta técnica es compatible con una gama más amplia de dispositivos, incluyendo procesadores antiguos o de bajo rendimiento.

Por otra parte, el núcleo del algoritmo de triangulación empleado en la fase de visualización para unir la malla regular al TIN presenta varias optimizaciones importantes respecto a la propuesta HM original. Se han fusionado parcialmente el procedimiento para reconstruir la convexificación local en el interior de las celdas con la generación de los triángulos complementarios entre la malla regular y el TIN, para reducir pasos innecesarios y aumentar la eficiencia del algoritmo. Además, el método GPU-HM aplica una reordenación a los vértices de la malla TIN en el *buffer* de geometría para mejorar el acceso a los vértices del borde y eliminar un nivel de indirección en las estructuras de datos. Por último, se han aplicado técnicas convencionales para mejorar el paralelismo en la ejecución de tareas en la GPU, como desenrollar bucles o eliminar bifurcaciones en el código del *shader*.

El método GPU-HM ha sido implementado y evaluado con varios modelos de prueba con resultados satisfactorios en términos de calidad y rendimiento. La implementación consigue visualizar de forma interactiva modelos con varios millones de triángulos, sin discontinuidades geométricas o superposiciones entre las partes. El método GPU-HM es, por tanto, el primer método paralelo de visualización de modelos híbridos del terreno implementado en la GPU.

Otro nuevo método para la visualización de modelos híbridos del terreno presentado en esta tesis es el algoritmo EHM (*Extended Hybrid Meshing*). Este nuevo método consigue incorporar técnicas de visualización multirresolución en la parte de la malla TIN. Es decir, en el método EHM, tanto la malla regular como el TIN se visualizan mediante técnicas multirresolución que reducen dinámicamente el número de triángulos utilizados en el modelo según las condiciones de la escena, sin que ello afecte a la calidad de la imagen generada. Además, el algoritmo EHM no depende de ningún método de multirresolución en particular. Puede combinarse con cualquier método que funcione para mallas triangulares irregulares, siempre y cuando se pueda restringir parcialmente el orden de eliminación de los vértices del borde del TIN según unas simples reglas o precondiciones. Cumpliendo esta propiedad se puede garantizar la reconstrucción correcta de la convexificación local del borde del TIN, sin importar el nivel de simplificación en que se encuentre.

Las técnicas de multirresolución más utilizadas para mallas irregulares se basan en una estructura jerárquica en forma de árbol para almacenar tanto las operaciones de simplificación como las dependencias entre ellas. Así pues, la forma más sencilla de restringir el orden de las operaciones de simplificación en los vértices del borde consiste en alterar el proceso de construcción de esta estructura, para que cumpla las restricciones del algoritmo EHM. De

esta forma, no se introduce ningún coste adicional en el proceso durante la visualización y el rendimiento del método multirresolución no se ve afectado en absoluto.

Los beneficios de este enfoque son claros, ya que el modelo híbrido al completo se visualiza con técnicas multirresolución sin incurrir en penalizaciones de rendimiento adicionales causadas por el algoritmo de triangulación. De hecho, el coste del algoritmo de triangulación adaptativa en la fase de visualización es proporcional al número de vértices activos en el borde del TIN, por lo que su rendimiento aumenta a bajos niveles de resolución, como cualquier otra técnica multiresolución. Por tanto, el algoritmo EHM logra generar un modelo híbrido del terreno de calidad y sin discontinuidades entre las mallas pese a la geometría variable de los bordes de los modelos multirresolución.

El algoritmo EHM ha sido implementado y probado con resultados satisfactorios con modelos de prueba, tal y como se expone en la tesis. El coste de la triangulación adaptativa resulta ser mínimo respecto al coste de las técnicas multirresolución en el TIN y la malla regular, así que el rendimiento final del método depende principalmente de la eficiencia de los algoritmos multirresolución utilizados. La calidad de las mallas obtenidas, igual que en casos anteriores, es buena y pese a la elevada variabilidad de los bordes de las mallas a unir, no aparecen discontinuidades ni huecos entre ellas.

Por último, en esta tesis se presenta también un nuevo método para la visualización de modelos híbridos del terreno que, al contrario que los métodos anteriores, no está basado en el algoritmo HM original. El algoritmo EDP es una nueva propuesta que presenta varias mejoras importantes respecto a los enfoques anteriores. Se divide igualmente en dos fases, una de preprocesado en la que se precomputa y codifica la información necesaria para teselar los bordes de las mallas eficientemente, y otra de visualización en la que se genera la triangulación adaptativa y se visualiza el modelo híbrido completo.

La característica más relevante del algoritmo EDP es que puede generar la triangulación adaptativa entre de las mallas regulares e irregulares sin la inclusión de nuevos vértices adicionales en el borde del TIN. Los métodos basados en el algoritmo HM realizan la triangulación completamente a nivel de celda y por tanto requieren que el área a teselar de cada celda sea un polígono perfectamente cerrado e independiente del resto. Para conseguirlo, se introducen en el borde del TIN vértices adicionales situados en los puntos de intersección entre el TIN y la malla regular. Estos vértices únicamente dividen en dos partes una arista exterior del TIN y, por tanto, no alteran la geometría de la malla, pero simplifican mucho el proceso pues cada arista exterior está completamente contenida en una única celda de la malla regular.

Sin embargo, estos vértices adicionales también incrementan la complejidad de la triangulación, puesto que aumenta el número de triángulos y vértices que participan en el proceso. El algoritmo de EDP, en cambio, es capaz de enlazar los bordes de las mallas regulares sin requerir que las celdas parcialmente cubiertas de la malla regular formen polígonos cerrados e independientes. Para ello, utiliza como primitiva básica para codificar la información de triangulación, la arista exterior del borde del TIN. Con esta primitiva se consigue codificar la triangulación local parcial de cada celda de la malla regular y, al mismo tiempo, permite resolver los casos en que una arista atraviesa dos celdas contiguas, aplicando una serie de reglas basadas en su orientación.

Como líneas de investigación futuras, en esta tesis se proponen diversas rutas de trabajo. Por una parte, se contempla el estudio de los atributos no geométricos del modelo híbrido, para mejorar la gestión de características visuales importantes tales como la textura y el color u otros tipos de datos geofísicos asociados al terreno, como el uso del suelo o la temperatura. Por otra parte, y más a largo plazo, sería interesante abordar el problema con un enfoque completamente diferente, usando técnicas en el dominio de la imagen y no en el de la geometría de los modelos. Sería especialmente interesante en casos donde la precisión geométrica del modelo no es tan relevante como su apariencia final, o en casos en que la geometría de los modelos varía constantemente y no se puede aplicar ningún tipo de preprocesado.

A mi familia

"No man is an island,
entire of itself;
every man is a piece of the continent,
a part of the main."

Meditation XVII, John Done

"Less is a possibility."

Generation X, Douglas Coupland

"Una mañana, sin más, decidí hacerlo."

Cuentos premonitorios, Cristof Polo

Acknowledgments

I would like to express my sincere gratitude to the people and institutions which have supported me during this thesis.

Special thanks goes to my thesis advisors, Montserrat Bóo and Margarita Amor, for their guidance, patience and support. Their continuous confidence in my work has been decisive in the completion of this dissertation. My gratitude also extends to Javier Bruguera, former head of the Department of Electronics and Computer Science at the University of Santiago de Compostela, for his valuable help and motivation.

I would also like to acknowledge the Department of Electronics and Computer Science, specially to the Computer Architecture Group, and to the Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS) at the University of Santiago de Compostela, as well as the Computer Architecture Group at the University of A Coruña, for providing the physical resources and technical support required by this project.

My gratitude also goes to Prof. Dr. Jürgen Döllner and the people from the Computer Graphics Systems Group at the Hasso-Plattner-Institut, University of Potsdam, for their warm welcome and kind support during my two stages in Potsdam.

At a more personal level, I would like to thank to the many friendly people from Lab 13 at the DEC and from Lab 0.2 at the UDC, for their help, kindness and support, and to the people at CGS lab at the HPI for making me feel part of their group. I can no forget to thank to the wonderful people I met in Berlin, specially Arian, Jana, Tassilo and Marie for being my adoptive family there, and Beatriz for being such a good friend in a cold autumn. My family and close friends deserve my deepest thanks for their invaluable patience and encouragement during difficult times through all these years.

Finally, I am also thankful to the following institutions for funding and supporting this work: Xunta de Galicia under projects 08TIC001206PR, 08SIN011291PR and the Program

for Consolidation of Competitive Research Groups ref. 2010/06 and 2010/28; Ministry of Science and Technology of Spain under contract TIN 2007-67537-C03; and Ministry of Science and Innovation of Spain, cofunded by the FEDER funds of the European Union, under contracts TIN2010-16735 and TIN2010-17541.

Santiago de Compostela, November 2013

Abstract

In recent years, advances in sensing and data processing technologies have led to a continuous growing of geographic datasets. Interest about using this increasingly detailed geographic information, has consequently arisen in application domains such as urban planning, cartography, global navigation systems or virtual reality. However, real-world digital terrain models demand for efficient techniques for managing and rendering their complex and large-scale geometric data while, at the same time, may require specific features and operations. An effective terrain model should therefore maintain a good trade-off between the accuracy with which the terrain surface is modeled, storage requirements, and real-time rendering.

Multiresolution representations try to solve the problem by reducing the rendering cost without a significant loss in the visual appearance of the scene. Unnecessary detailed data is reduced by using alternative representations of the same model with different accuracy, simplifying the polygonal geometry of distant or unimportant portions of the model. A different strategy, barely studied until recently, is to use hybrid terrain models combining large regular data sets for large unimportant areas of the terrain, and high-resolution irregular meshes for topographically complex terrain features and man-made microstructures. A more precise geometry description of morphologically important terrain features is thus provided, enhancing the perceptual quality of visualized terrains models without increasing the overall resolution of the whole mesh.

In this thesis several techniques to generate and visualize 3D hybrid terrain models are proposed. These techniques can integrate geographic data from heterogeneous sources, combining the benefits of coarse-grained grid-data and fine-grained irregular data without requiring a remeshing process that would modify the original datasets. Multiresolution rendering of the obtained representations are supported in the large regular areas, and in some cases in the whole hybrid model. All these methods are based in a local tessellation approach which can

selectively rebuild the adaptive tessellation between the regular and irregular meshes of the models during the interactive visualization process. This approach was originally developed in the Hybrid Meshing algorithm developed by some members of the research group, which is the departing point of this work.

In particular, this thesis contains an analysis of the Hybrid Meshing algorithm and two different proposals of an interactive hybrid terrain model renderer based on it. Since the original algorithm proposes a theoretical hardware unit in the graphics pipeline to assist in the rendering of the hybrid terrain model, the algorithm have been adapted to a conventional graphics pipeline in two different ways. One of the methods is complete faithful to the Hybrid Meshing algorithm, while the other avoids the preprocessing phase by using a conventional tessellation algorithm. The Hybrid Meshing algorithm has been also enhanced and adapted to run in a parallel implementation using the Geometry Shader Unit of the GPU. This parallel proposal incorporates several improvements in the organization of the data lists and in the interactive visualization phase.

Besides the several extensions and improvements to the Hybrid Meshing algorithm, two new methods have been developed to overcome the limitations of the previous approaches: the EHM and the EDP proposals. The EHM algorithm supports multiresolution rendering of the whole hybrid model, both in the base regular grid as well as in the high resolution irregular meshes. Therefore, the EHM algorithm is the first method in the literature achieving a complete view-dependent rendering of a hybrid model without cracks or other visible artifacts in the visualization of the model. The EDP algorithm proposes the use of the external edges of the irregular meshes as the fundamental primitive to tessellate the boundaries of both types of meshes. This approach achieves a better parallelism in the rendering phase while simplifies the preprocessing phase by minimizing the insertion of vertices in the irregular mesh.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digital Terrain Models (DTMs) are fundamental components in the geographical information processing field. They have been used since the 1970s to model, analyze and display the surface form of the Earth [23, 49, 22]. Today, they constitute the basis of a well-established and increasing number of information systems in both the geoscience and engineering domains such as Geographic Information Systems (GIS), cartography, urban and landscape planning, natural disaster management, network planning in telecommunication and energy supply, virtual immersive environments or video games.

In many of these application domains, from the critical monitoring systems to immersive environments and entertainment, a high quality visualization of the terrain models is required by the users. However, real-time 3D visualization of large terrain models has been traditionally regarded as a challenging task. Terrain datasets typically include huge amounts of data which exceed the capabilities of hardware, requiring additional optimization techniques to provide a representation of the original dataset with a reasonable frame-rate. Even though graphics hardware performance increases continuously, the subject remains open as the availability of terrain elevation data also rises at fast pace using different technologies [81]. Moreover, the interest in digital terrains is gradually switching from large terrain natural areas to more complex human environments, often formed by a number of overlapping datasets acquired over the years with a number of manual and semiautomatic technologies. The integration of disparate datasets with different data structures, organization and resolutions is hence an interesting problem and should be considered with more attention.

In this thesis we present our work dealing with hybrid terrain models formed by the union of regular and irregular datasets. Specifically, we have developed different solutions and optimizations for the real-time rendering of hybrid terrains.

In this chapter the fundamentals of digital terrain modeling and visualization are exposed, as well as a brief introduction to the Hybrid Meshing algorithm, which is the starting point of our work. At the end of the chapter, a summary of our contributions to the field and a presentation of the structure of this thesis report are provided.

1.1 Digital terrain models

Generally speaking, a terrain model is a digital representation of the ground continuous surface. However, this term is used in the literature with different meanings and frequently mixed with related terms as Digital Elevation Model (DEM) or Digital Surface Model (DSM). Within the context of GIS, DTMs are generally considered as a digital representation of a landscape which includes at least the characterization of the relief. In addition to the geomorphological information, a DTM can also contain additional content about the terrain as aspect, material, land cover, land use and other geophysical properties [85, 11]. In [51], a classification of different modeling methods that have been developed in computer graphics and geological structures is presented.

In this thesis, the term DTM is used as a digital data set describing some attributes of the terrain surface form. Conversely, a DSM is more appropriately used to denote a representation of the upper surface of a landscape, including vegetation and buildings, and thus is a slightly different concept than a DTM.

A DEM, on the other hand, is a specific form of terrain model representing the elevation of the ground surface as a two dimensional array or matrix of height values. This kind of raster data sets can be also named as Digital Height Model (DHM) or Digital Terrain Elevation Data (DTED). A Triangular Irregular Network (TIN) is a different vector-based form of representing elevation using irregularly distributed elevation samples joined to form a mosaic of irregularly shaped triangular faces. DEMs, TINs, contour or break lines are the most important types of DTMs. For example, in the well established CityGML standard [26] terrain relief may be specified as a regular raster or grid, a TIN, by break lines or by mass points.

A DTM can be created using different approaches: by sampling the real terrain with different acquisition techniques, by the interpolation or derivation of the data from a previous

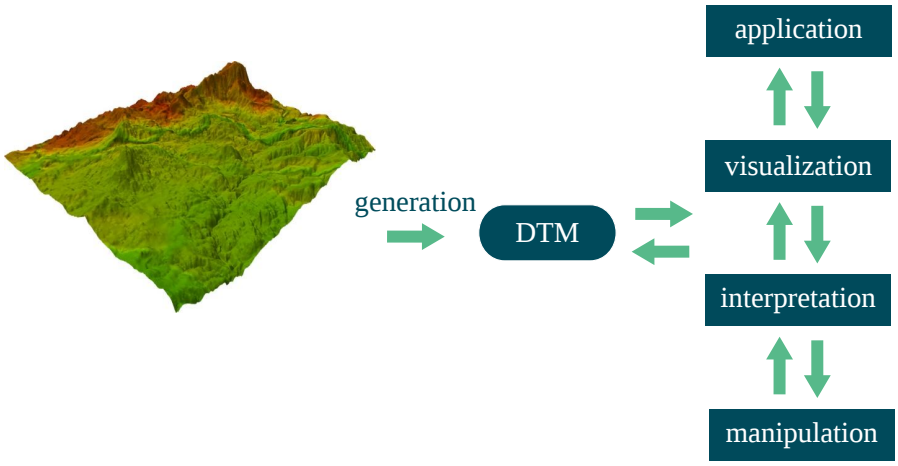


Figure 1.1: Standard workflow working with digital terrain models.

representation of the surface (i.e. cartography), or by generating synthetically the model by means of a simulation software.

Applications working with DTM deal with a broad set of tasks in addition to the generation (model construction), such as the storage and transmission, manipulation (modification, refinement and derivation), interpretation (data analysis and information extraction), visualization (graphical rendering) and domain specific operations (specific functional requirements in a given application domain). Figure 1.1 shows how these tasks are organized in a standard workflow for terrain model processing [83]. For large-scale terrain models, the storage, transmission, processing and visualization requirements require an effective encoding and rendering technique to cope with their geometric complexity, specially in real-time interactive visualization applications.

The following subsections introduce several aspects of the data acquisition for the generation of DTM and the organizational models used to store and access the model information, two important topics in the generation of DTMs.

1.1.1 Data acquisition

Data included in a DTM consist of a set of samples about relief elevation and other additional attributes of the terrain, like photographic images or material definitions. The particular technique used for the sampling operation basically depends on the efficiency, cost, technological

maturity and availability of capturing devices. Real-world terrain data sets are generated from one or several of the following data sources: manual ground surveys, photogrammetric data capture, radar or laser altimetry (*Laser Imaging Detection and Ranging* or LIDAR) and digitized cartographic data [11]. Additionally, synthetic terrain models may be created using computer simulation software.

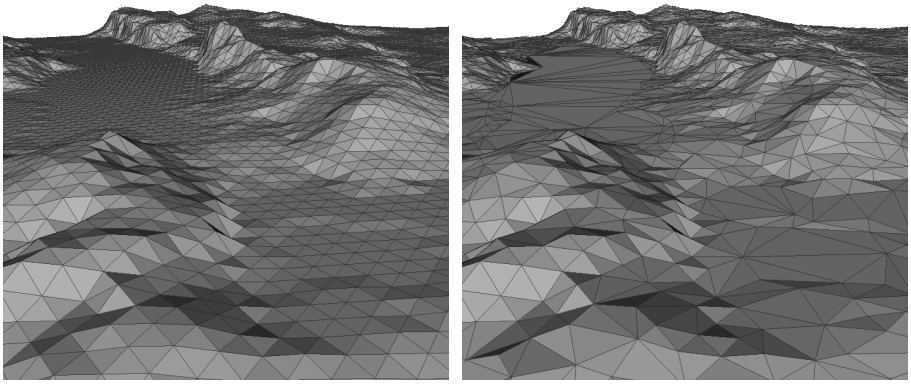
DTMs created from ground surveys tend to be very accurate since, during the acquisition, the most significant relief points are directly measured and collected through field instruments. However, this process is time consuming and error prone and thus data sets are usually limited to specific small areas of interest.

Photogrammetric data acquisition is based on the sampling of aerial and satellite photographs. The DTM is automatically extracted from the imagery using regular or progressive sampling techniques and digital stereo image correlation. Similar regular sampling methods are used with elevation data from altimetry instruments. Since photogrammetric as well as altimetry data capturing techniques use remote sensing, they can be fully automated to minimize the collection effort. Therefore, these techniques are frequently used for large acquisition projects and public nationwide data. However, when regular sampling is used, an excessive number of points is collected in homogeneous regions and, at the same time, too few points are captured in rugged areas. This problem can be partially solved by using a progressive sampling scheme, where the sampling resolution is increased wherever the accuracy of the original sampled is not satisfactory.

A lower cost terrain data source for DTM creation is the digitalization of existing analogue cartographic documents, such as contour maps and profiles. This data collection technique has been specially used in the past due to its reduced cost. However, it presents several accuracy problems: contour lines need to be oversampled while flat areas between contours remain largely empty, precision errors are introduced during the raster scanning and vectorization of the data, and, moreover, most of the additional information contained in the map is lost.

1.1.2 Data organization models

Once the data acquisition process has finished, the collected data elements are structured for the construction of a comprehensive DTM. During this construction process the topological relations between the captured data elements are defined, as well as the interpolation model to approximate the terrain data in areas where no samples were collected. The data structure of a DTM is the way the collected data is organized so as to data could be accessed and processed



(a) (b)

Figure 1.2: Digital terrain models representing the same surface: (a) grid; (b) TIN.

efficiently. This structure should be selected according to the nature of the collected data and the purpose of the DTM. For example, a DTM data may be tightly related to a numerical computation model which needs to be preserved.

A poorly structured model will cause performance and coherence problems during the processing. Several data structures have been used in the past for the representation of DTMs. Nevertheless, the majority of the models used today for real-time visualization are structured as a DEM, i.e. a rectangular elevation grid (see Figure 1.2(a)), or as a TIN, i.e. a network of irregular triangles with different size, shape and orientation (see Figure 1.2(b)).

Data grids present a matrix structure where the elevation values are arranged in rows and columns, according to their x and y coordinates. Thus, the grid can be stored as a two-dimensional array of elevations values –one for each sampled elevation position– as the other coordinates are easily determined from their relative positions on the data grid. It is easy to see that data grids, also called height fields, are just discrete parameterizations of a scalar function defined over the sampled area domain. Thus, topological relations between data points are implicitly stored and the neighbors of any element can be found with simple index arithmetics. The main advantage of grids is that their processing and visualization tend to be relatively straightforward and fast, as working with matrices of data in modern computers and graphic hardware is a highly optimized operation. Additionally, since this data organization

closely resembles a multilayered 2D image, several image processing algorithms can be easily applied to height fields as processing filters.

On the contrary, one disadvantage of grids is their inability to accommodate the point density to the complexity of the terrain relief. As a result, gridded data occupies a large storage space with highly redundant data in flat areas, while abrupt changes in the relief of rough terrain areas may be under sampled. In this case, the only solution to represent the terrain for a required level of accuracy is to increase the overall sampling density, which may generate an excessive number of data points. Another important drawback of using grid terrain models is that some topographic features forming 3D structures (e.g. overhangs, cliffs, caves, riverbeds [4], wave-cut platforms or surface-rupturing landforms generated by earthquakes [84, 87]) cannot be represented by the model. Since these structures involve several elevation values for the same point in the base plane, it is impossible to characterize them using height fields.

TIN data structures constitute another common strategy for representing DTMs. In this case, terrain data is stored as a triangular mesh generated by the tessellation of the sampled points –usually following an irregular distribution– in the actual terrain. TIN based models are able to represent any terrain feature without problems, as they are real 3D surfaces. Consequently, adaptive sampling is also possible by allocating more vertices in the abrupter regions of the model and a fewer number in the flat homogeneous areas. This ability to store irregularly sampled terrains contribute to a bigger efficiency in the number of points needed to faithfully represent the relief. However, it also implies an increased complexity of the data processing since topological relations have to be computed and recorded explicitly.

The advantages and drawbacks of both structures have been extensively discussed in the literature [83], concluding that none of them is clearly superior for every kind of processing tasks involving DTMs. Hence, a real-world digital terrain modeling system needs to work with both classes of data structures at the same time. For instance, the terrain model defined in the CityGML standard [26] allows the combination and nesting of different terrain models delimited by region of interest markers. This type of scenario, where several DTMs from different data sets are available for the same territory, is not uncommon.The combination of large scale regional DEMs with finer resolution TINs representing specific geologic structures or man-made constructions, is a fairly typical case. Hybrid DTMs represent [4, 88, 87], in this context, an alternative and convenient approach for enhancing large-scale DEMs, without increasing the grid resolution or transforming the original dataset into a TIN-based represen-

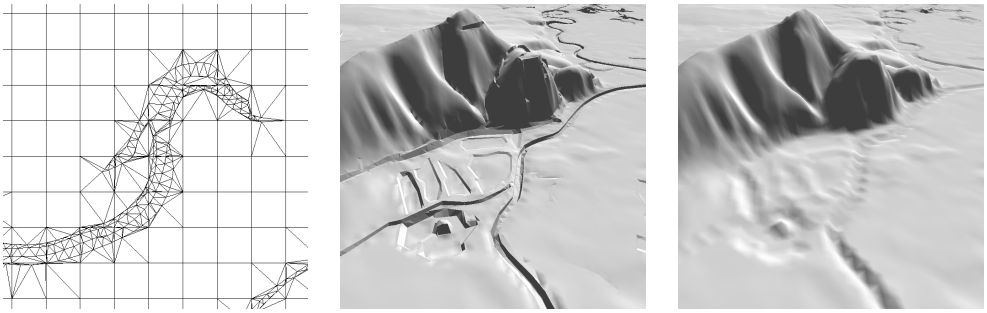


Figure 1.3: Example of a hybrid terrain model formed by a base grid and highly detailed TINs extracted from [4]. A wireframe view of the terrain model is shown at left, a render of the integrated model in the middle and the same view but without using the detailed TINs at right.

tation. Hybrid data model can thus represent really complex terrain models by integrating information from different sources to detail specific regions of interest. As an example see Figure 1.3 where the grid cells describes the DEM and the TIN is included for topographically complex or interesting terrain parts like the riverbed shown in this case.

1.2 Interactive 3D visualization

The visualization or rendering of three-dimensional (3D) models consists in the generation of a recognizable visual representation – usually a photorealistic image – of the digital data stored by the model. If this visual representation is generated within strict time constraints since any of the parameters of the visual representation change (i.e. lightning, model attributes or viewpoint position), the process is labeled as *real-time* visualization. The process becomes interactive if the main aspects of the visual representation can be directly controlled by the user process.

The core component of real-time visualization systems is the *graphics rendering pipeline* [1]. The *pipeline* is the standard organization of the processes involved in the generation of images from a 3D scene description containing models, lights, and virtual camera definitions. The resulting image produced by the pipeline is determined by the geometry and the material

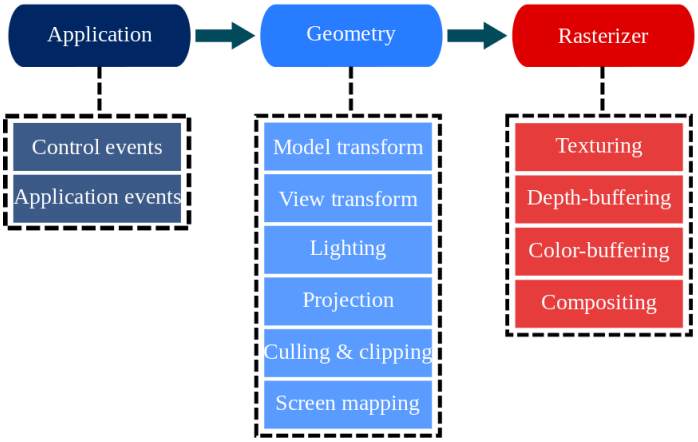


Figure 1.4: Conceptual rendering pipeline.

properties of the models, the lightning definition, and the placement of the virtual camera in the scene. This topic is covered in more detail in the next sections.

1.2.1 Graphics rendering pipeline

The concept of a rendering pipeline may be implemented in many different forms. Conceptually, however, the pipeline is composed by three main stages: application, geometry, and rasterizer. This structure is depicted in Figure 1.4. Those three stages are also composed by several substages forming a pipeline in itself, which may be implemented by software components or by a hardware graphics accelerator. Ideally, this pipeline architecture is able to execute pipeline stages in parallel, producing a noticeable increase in performance only limited by the pace of the slowest stage.

The application stage represents the rendering application program, running on the CPU and feeding commands to the graphics subsystem. The geometry stage performs per-vertex and per-primitive operations such as coordinate transformations, lighting, texture coordinate generation, and clipping. An example of the operations included in this stage is depicted in Figure 1.5. Finally, the rasterizer subsystem performs per-pixel operations, like writing the final color values into the framebuffer, or other complex operations like depth buffering, alpha blending, and texture mapping. Nowadays, the geometry and rasterizer stages are usually

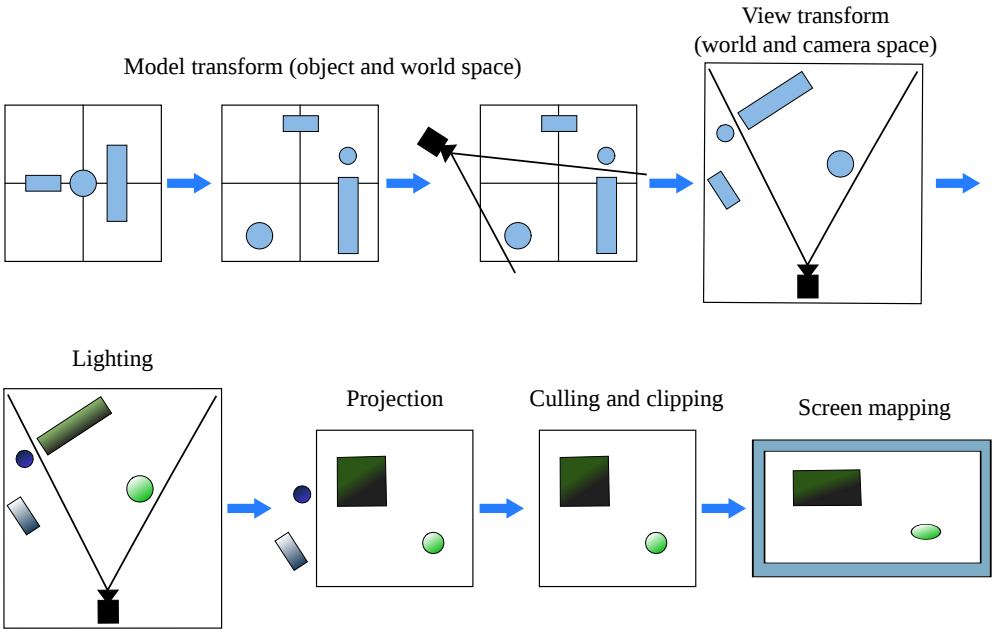


Figure 1.5: Geometry processing pipeline.

performed by specific hardware, denominated Graphic Processing Unit (GPU), that contains many programmable cores instead of the traditional fixed-operation hardware.

1.2.2 3D object representation

A 3D object is a collection of geometric primitives representing an entity in the scene considered for rendering. Consequently, a scene is the collection of all the models in the environment to be rendered, also including other attributes as material descriptions, lighting, and the view definition. 3D objects can be described in terms of the volume they occupy (volumetric representation) or by the external surface of the model which marks the separation between the object and the surrounding space (surface representation). Volumetric representations are not used for terrain models, since terrain models already represent an actual surface.

In computer graphics, polygonal meshes are widely used for the representation of 3D models since they present a convenient mathematical simplicity which leads to simple and regular

algorithms that embed well in hardware. Thus, they are frequently used as a sort of lowest common denominator for any model representation. Polygon meshes approximate the surface shape of an object by specifying a set of points in space, which define the position of vertices of polygonal faces, and the adjacency relations between them. In practice, triangle meshes are almost exclusively used, since triangles are simple and convenient convex polygons and all their vertices reside in a single plane in a 3D space. Furthermore, any polygonal mesh can be transformed into a triangle mesh since every polygon with n vertices that does not intersect with itself can be triangulated using $n-2$ triangles. Several algorithms [40, 74, 75, 55] have been developed for the triangulation of a point set with different time complexities and properties in the results obtained. For instance, some triangulating algorithms are less prone to produce thin or sliver triangles than others, which is a desirable property since the area of sliver triangles is almost null and they may pose numeric problems in computations or introduce artifacts during visualization. Hence, the mesh geometry is stored by the mesh vertices, and the topology is represented by the connectivity between the mesh points, that is, by the edges and faces of the mesh. Different edge-based and triangle-based [1] strategies exist to store the connectivity, such as triangles list, triangle fans, triangle strips, shared vertex representation or half-edges. Nevertheless, current graphics hardware is only capable of dealing with triangle-based ones.

An important issue for avoiding rendering artifacts related to polygonal meshes is the *two-manifold* property. A polygonal mesh is said to be two-manifold if every point in the mesh has a neighborhood topologically equivalent to an open disk of E^2 (Euclidean space). Intuitively, two-manifolds are non-selfintersecting, closed surfaces. Consequently, a mesh is a *two-manifold with boundary* if every point has a neighborhood topologically equivalent to an open disk of the two-dimensional euclidean space E^2, except points on the edge of an open surface patch. Intuitively speaking again, two-manifolds are non-selfintersecting, open surfaces. Figure 1.6 shows some examples of non-manifold meshes.

Rendering applications are constrained by a tradeoff between representation detail and performance. Note that the accuracy obtained modeling a surface with a triangle mesh model is directly related to the storage cost and computation time required to process it. Even though hardware computing power grows up quite fast, specially in graphics related hardware, the complexity of the rendered 3D models has continuously increased due to technological improvements of the sensing technology. Naturally, a model capturing very fine surface details is desirable for ensuring the best quality results in numerical processing, but the storage and

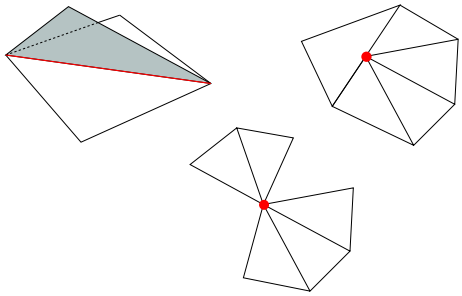


Figure 1.6: Examples of non-manifold meshes.

transmission costs for high-quality models rises dramatically for finer sampling resolution. Furthermore, when the model is rendered from a distant viewpoint the detail of the entire object is highly redundant, unnecessary and wasteful. Since many applications require far less detail, specially those with time constraints as interactive visualization, it is often desirable use a simpler approximation of the original model to achieve interactive rendering and meaningful results at the same time. This conflict between rendering quality and speed is handled with adaptive rendering techniques, to cope with the wide range of viewing conditions in a consistent manner.

1.3 Adaptive multiresolution rendering

The main objective of this dissertation is rendering high quality hybrid models without cracks or holes. Our second objective is achieved interactive rendering of these models. Therefore, a balance of the competing requirements of realism and frames per second is chosen. In this section, we describe briefly the most common multiresolution approaches for terrain rendering. A more detailed description can be read in [57].

Realistic representations of geometrically complex scenes may easily use as far as millions of polygons, specially in the case of terrain models. In these context, occlusion and view frustum culling techniques [1] are not sufficient for meeting the performance requirements of interactive visualization systems and high quality final images.Adaptive multiresolution techniques use different representations of a dataset during the rendering process constrained by user imposed time or quality requirements. The strategy of using several representations of a geometric object with different levels of accuracy and complexity is denominated *Level-*

Of-Detail (LOD) modeling. This technique, first exposed in [15], manages to reduce the rendering cost of a scene without significant loss in the visual appearance of the rendered image. Even though the most common applications of LOD modeling use polygonal meshes, the essential concepts may be applied to other model representations as well.

The simplest implementation of a LOD representation system consists in using a collection of meshes of different sizes, each one of them representing the object at a given resolution. LOD meshes can be built easily by the repeated application of any simplification algorithm with different quality targets. This basic approach, called *discrete LOD*, is similar to the original concept proposed by [15] and it is easy to implement, easy to generate and generally quite fast. However, this simple LOD approach present some practical limitations. First, the number of levels is limited by storage constraints since each mesh is stored independently. Second, the transitions from one level to a different one are abrupt, hence causing unpleasant *popping* effects during the visualization. The third main drawback of discrete LODs is that the resolution of each level mesh is uniform, meaning that there is no possibility of tuning the resolution of very large objects spanning several ranges of distance from the viewpoint within the same view. To overcome the first two problems mentioned above *continuous LOD* approaches were developed. The *view-dependent* or *anisotropic* LOD methods were developed to solve the third one.

A continuous LOD model is generated creating a range of simplified representations of the original model, in order to capture a continuous spectrum of levels of detail that could be reconstructed on demand. An usual approach to create these multiresolution structures is to store a hierarchical sequence of coarser and coarser versions of the model, automatically generated by surface simplification methods. Note that although surface simplification and multiresolution model generation algorithms are related in some aspects, their aims are completely different. In polygonal surface simplification the goal is to generate a simplified model from the input polygonal mode. On the contrary, the multiresolution model generation process has a more complex goal: to build a new representation of the original model which allows quick generation of adaptive approximations of the original model.

Using a multiresolution data structure allows the real-time extraction of finely tuned representations during rendering to fulfill the requirements of the visualization system. The basic workflow is followed by these techniques during rendering is depicted in Figure 1.7. At the beginning of each frame generation the multiresolution structure is queried to obtain the cus-

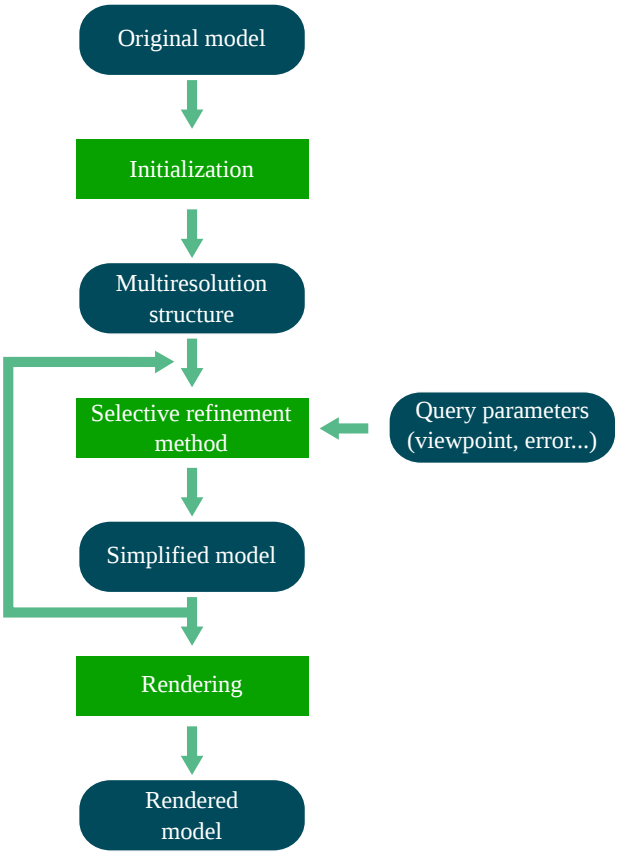


Figure 1.7: General workflow of a multiresolution rendering method.

tomized representation of the model that will be used in the frame according to the active view parameters.

Several publications [58, 46, 42, 43, 67, 68] have characterized the key properties to obtain a valid multiresolution scheme. Despite some minor differences depending on the author's point of view, they can be summarized in the following points:

– *Efficient information processing*. The algorithm for retrieving data from the model should be capable of extracting any of the different LODs existing in the structure in a short period of time. Hence, information should be stored according to an optimized strategy for the extraction of data on request.

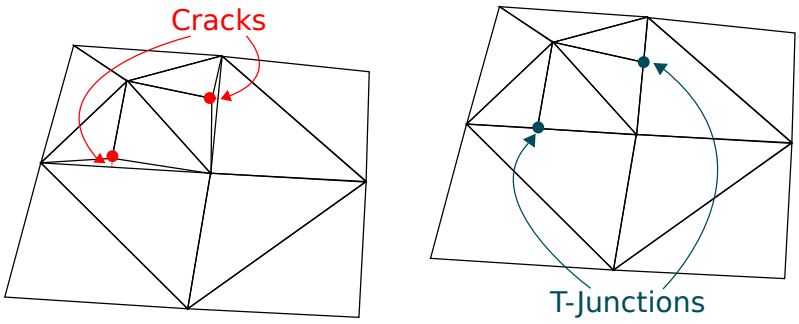


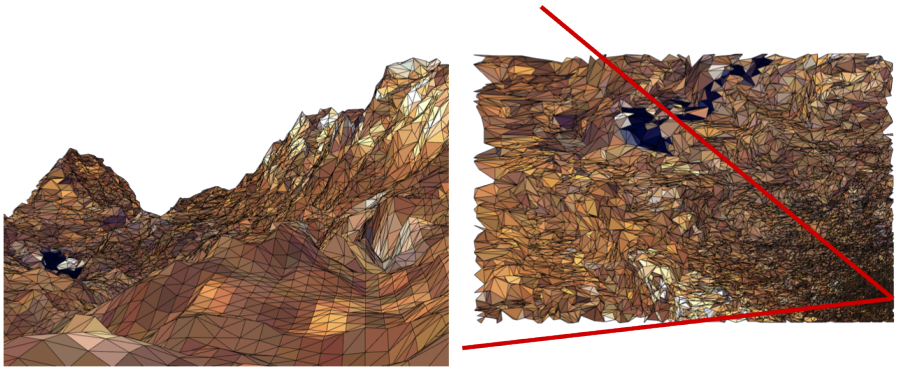
Figure 1.8: Common LOD artifacts: (a) cracks; (b) T-Junctions.

– *Small storage penalty*. The storage cost of the multiresolution representation should not be significantly larger than the original object. A limited constant factor of increment is accepted and, actually, unavoidable.

– *Continuous surface*. None of the approximations retrieved from the multiresolution model should present any inconsistencies or holes in the surface. A common problematic situation in multiresolution models is when adjacent triangles exist at different levels of detail. In this situation, some *cracks* may appear along the edge, since the higher LOD introduces extra vertices that are not present on the lower LOD. During rendering these cracks can cause holes and annoying artifacts in the rendered image. Another related artifact appears when the vertex from a higher LOD triangle does not share a vertex in the adjacent lower LOD triangle, forming a *T-junction*. This can also result in visible shading differences across such edges. Figure 1.8 illustrates both of these cases.

– *Smooth transition between consecutive approximations*. Transition from one simplified approximation to another (either to increase or to decrease detail) should be carried out smoothly, avoiding abrupt changes and popping effects.

– *Avoiding loss of information*. The multiresolution model should be also capable of representing the original object accurately.



(a) (b)

Figure 1.9: View-dependent LOD rendering from [47].

Discrete and continuous LODs are designed to work well when the graphic scene contains a large number of small objects. View-dependent LOD approaches, on the other hand, are useful for rendering very complex models representing physically large objects, such as terrains, that cannot be adequately simplified without view-dependent techniques. Rendering very large scale terrain data poses an additional issue that need to be specifically addressed: since the model cover large areas of terrain, the viewpoint is simultaneously very close and very far away from the terrain surface. Using a uniformly detailed multiresolution model in this case is not sufficient to obtain a good quality visualization. The solution is to use an anisotropic or view-dependent multiresolution model. In a view-dependent multiresolution model, the same object can span multiple levels of simplification at the same time. For example, nearby portions of the object will be shown at higher resolution than distant ones. Hence, when applied to DTMs, terrain areas lying nearer the viewpoint could have a finer resolution than ones farther away. Figure 1.9 shows an example of this technique: at left, Figure 1.9(a) shows a high quality rennder of a terrain model obtained with a view-dependent method; at right, Figure 1.9(b) depicts exactly the same scene but rendered from top and also showing the view frustum with red lines. As it is expected, triangle density is much higher near the viewpoint and it becomes progressively lower at larger distances from it. This leads to an optimized use of the resources since geometry primitives are allocated where they are most needed within objects, avoid the aliasing effects associated to the different spatial precisions visible in the image. Therefore, a view-dependent simplification criteria is used to select dy-

namically the most appropriate level-of-detail for any model region taking into account the features of the region and the viewpoint location.

The key benefit of this approach is the local adaptation of the model geometric complexity to continuously changing view parameters. However, this dynamic re-tessellation adds a run-time overhead which should be considered, as the performance gain obtained because of the better simplification has to be greater than the additional time spent in the selection algorithm.

Geovisualization applications were among the first where those ideas were adopted and implemented, due to the huge size of the models. In the next subsections a brief overview of the different error metrics used in the generation and rendering of multiresolution models is presented first, and then a review of the most important grid-based and TIN-based multiresolution methods. For a exhaustive review of the field we recommend [25, 48, 57] as well established references.

1.3.1 Error metrics for multiresolution rendering

Using multiresolution methods in a geovisualization system means that some kind of progressive approximation of the original model is used to increase the performance or viability of the system, since the original model would be too large to be appropriately handled by the application [57]. Consequently, the coarser approximations used in the visualization introduce errors in the output that need to be adequately measured and managed to control the quality of the visualization. For this purpose, different error metrics have been developed in the literature, largely classified in two main approaches: those measuring the geometric error of the approximation in the object-space or those measuring the error produced in the image-space of the output. Both approaches are not mutually exclusive and they are often combined in the same multiresolution algorithm.

The object-space metric can be used to select the appropriate simplified representation in non-view dependent multiresolution method, as it measures the actual geometric deviation from the original models. However, an object-space geometric error metric is not sufficient to adaptively simplify terrain models. Due to the typical large extension of this kind of models, far away regions must be simplified more aggressively than nearby areas. As this depends on the camera position, which is continually changing in interactive rendering session, the metric used to select an approximation for rendering must adapt dynamically to the view conditions.

Therefore, object-space error metrics are most typically used during the generation of the hierarchical data structures for the multiresolution representation of a model. In this phase,

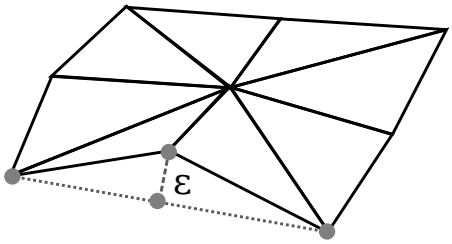


Figure 1.10: Vertical error introduced in the mesh removing the mid-edge vertex.

the error produced by every simplification operation must be measured and accounted. An error metric is used to select the most appropriate simplification operation for each step. A common choice for terrain elevation models was proposed in [76] as the vertical distance of a removed vertex with respect to the resulting height interpolated from its parent nodes, as shown in Figure 1.10. In a formal way, the error (ε) of omitting the data point in the simplified representation is difference between the elevation on the reference surface $F(x,y)$, and the elevation of the data in the final representation s_z:

$$\varepsilon = \| s_z - F(x,y) \| \tag{1.1}$$

Note that the error metric must use the original surface as reference to measure the error at each point and not the immediate parent nodes.Errors are accumulated during the iterative construction process, and thus the final error could be larger than the specified threshold, even if the threshold is respected at every simplification step.

Another question associated to the error metric for the simplification operations is the storage of the computed error values. In a *per-triangle* approach, the error is stored for each triangle appearing in the multiresolution tessellation of the model, which is quite costly in terms of memory usage. In a *per-vertex* approach, however, only one error value is stored for model vertex, largely reducing the memory cost of the metric. Additionally, using a technique called *error saturation*, those per-vertex error values can be also used to enforce different constraints on multiresolution hierarchies, such as dependencies between simplification operations or topology preservation. The error saturation consists in associate to each vertex the maximum error value of all its dependencies and its own computed error, which guarantees that dependency resolution is naturally handled during of the traversal of the multi resolution hierarchy

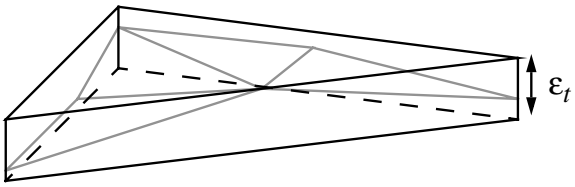


Figure 1.11: The object-space approximation error ε_t defines the size of the wedgie used for the estimation of the image-space error.

As stated above, an image-space error metric is required for the extraction of an optimal approximation of the original model in the rendering of terrains. Since this metric provides a visual fidelity error value taking into account the camera parameters at the rendered frame, it is used to guide the selection of elements from the multiresolution hierarchy to present a faithful representation of the original model without noticeable differences. It does so by estimating the visual error produced by the rendering of the selected approximation compared to the rendering of the original model, that is the number of pixels affected by the simplification. Since in the rendering process the object-space of the model is transformed and projected into image-space using a perspective transform, this screen-space error depends both on the object-space error of the approximation and on the current viewing parameters, such as the viewpoint location or the field of view. Consequently, objects far away of the camera are less significant since, due to the way the perspective transform works, error is larger for objects close to the viewpoint.

Using the perspective transform, the relation between the 3D object-space surface deviation ε and the screen-space deviation measured in pixel units p, can be generally estimated for as:

$$p = \frac{\varepsilon x}{2d \tan \frac{\theta}{2}} \tag{1.2}$$

where θ is the total field of view of the camera, x is the screen horizontal resolution and d is the distance in the viewing direction from the viewpoint to the object.

In [19] a bounding *wedgie* with a height equal to the object-space error of the approximation, as shown in Figure 1.11, is introduced for every triangle to estimate its view-dependent error. Therefore, the image-space error of a triangulation is the maximum projected length of the bounding wedgies of the triangles involved in the triangulation. Another well-known

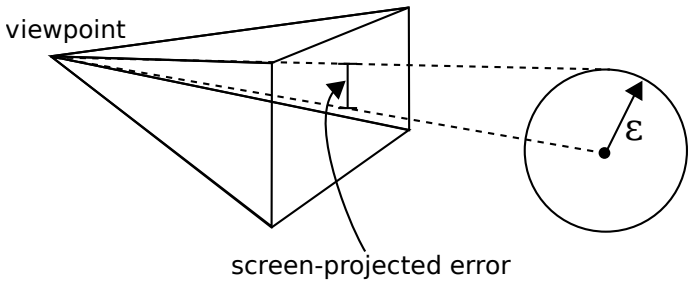


Figure 1.12: Computation of the screen-projected error using a bounding sphere of radius equal to the object-space error.

metric was presented in [44], using a bounding sphere instead of a *wedgie* to compute the image-space error valued based on the geometric error, as depicted in Figure 1.12. This metric can be conservatively *saturated* to simplify the view-dependent refinement of the multiresolution model. This error metric or other similar approaches are very frequently used, however, they only consider the visual error caused by the geometric deviation of the extracted model. More refined metrics take also into account appearance issues introduced by changes in non geometric attributes of the model as colors or textures of the original model.

1.3.2 Grid-based multiresolution approaches

The approaches dealing with regularly distributed height samples are by far the most common approach to the multiresolution rendering of terrain models. A particularly successful and widely used strategy consists in the generation of a regular multiresolution hierarchy which is used during rendering to extract a semiregular adaptive triangulation exclusively composed by isosceles right triangles (an example is depicted in Figure 1.13). Depending on the definition of the strategy made by each author, this strategy may be considered as a quadtree or triangle bin-tree triangulation. In practice, it is accepted that both of them produced the same kind of triangulations. The first methods exploring this strategy were developed by the influential works by Lindstrom [43] and Duchaineau [19]. Other authors followed and refined the same methodology in several works developed in the next years such as [58, 72, 6, 80, 82, 44, 45].

This kind of algorithms focused on reducing the number of triangles needed to render a representation of the model in each frame, assuming that the bottleneck of the system was triangle processing. However, the continuous improvement of the graphics hardware and the

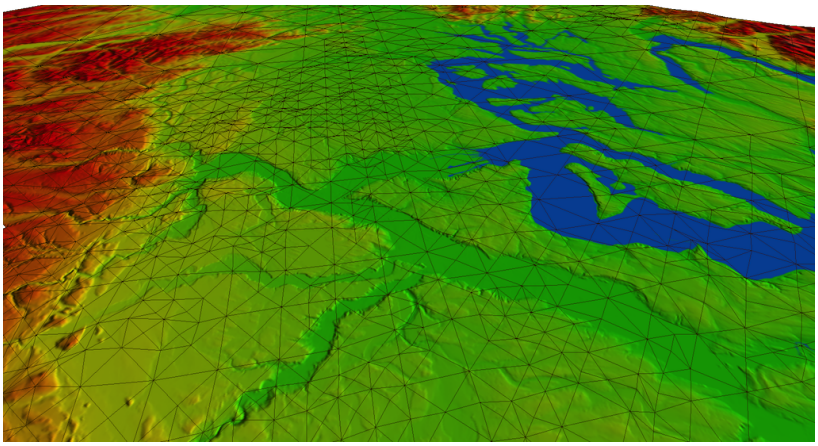
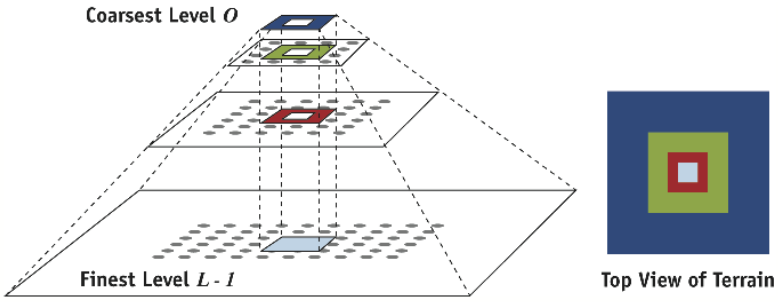


Figure 1.13: Partially simplified terrain mesh using a semiregular adaptive method.

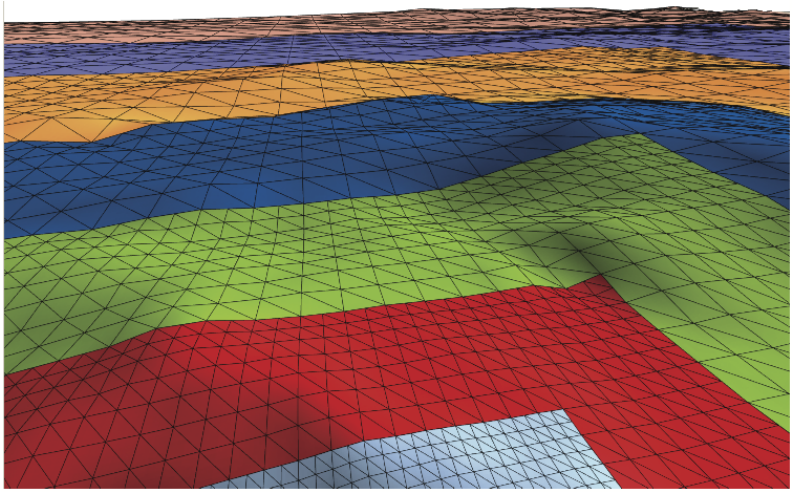
increasing size of the terrain models changed the bottleneck to the CPU, since the refining algorithm selecting the adequate tessellation triangles to be rendered was not fast enough to feed the GPU at convenient rate. Thus, new algorithms appeared reducing the CPU load by adopting a similar scheme of the previous technique but applying the selection algorithm on pre-assembled optimized triangle batches instead of on basic triangles. Namely, [66, 41] are two extensions of the previous work by Duchaineau [19] using this primitive batching approach. Another more recent and performant algorithm following this approach is described in [18].

A different approach for grid-based terrain multiresolution besides the adaptive quadtree-based techniques mentioned above is inspired by texture and image LOD techniques. The algorithms usually combine standard 2D data compressors methods with multiresolution schemes to reduce data transfer bandwidths and memory footprints. In particular, [73] partitions the terrain mesh into square patches tessellated at different resolutions. Each finer level is represented by all coarser level vertices plus the additional ones. Since only the additional vertices need to be sent, the required bandwidth is largely reduced improving the overall communications between CPU and GPU. The main challenge for this method, as for all tiled block techniques, is to stitch block boundaries at run-time.

Another very successful algorithm was presented in [46] and later optimized in [3]. This method organizes the terrain height data in a pre-filtered mipmap pyramid, as illustrated in



(a)



(b)

Figure 1.14: Geometry clipmap example from [3]: (a) data pyramid and clipmap; (b) rendered terrain levels.

Figure 1.14. View-dependent approximations are created by stitching together nested rectangular extents of the pyramid, cached in video memory and incrementally refilled with data as the viewpoint moves, using as refinement criteria the 2D distance from the pyramid center. The technique works best for wide field of views and nearly planar geometry, since the pyramidal scheme limits the adaptivity of the method. The LOD transition scheme works by smoothly blending height data between levels and stitching the level boundaries using zero-area triangles to avoid T-junctions.

1.3.3 TIN-based multiresolution approaches

There is a smaller amount of work devoted to multiresolution terrain rendering using irregular data meshes. Since the generalization of powerful GPU hardware, most of the developed method focused on regular meshes, since they fit better to be optimized by the hardware.

One of the first methods using TIN-based multiresolution was described in [67, 16], partially based on the previous work of the authors in pyramidal and hierarchical terrain models. This MultiTriangulation model is actually a very general framework with works by progressively refining or simplifying an initial TIN through a series of local updates, such as adding or removing vertices from the mesh, and retriangulating the results. Dependency relationships between mesh elements, represented as a directed acyclic graph, provide a way to enforce the creation of continuous meshes. There are several algorithms and data structures that have been adapted to this method, providing a quite large number of features and capabilities, as frame coherence or out-of-core data management.

Another important work was developed in [31], and is actually a refinement of his previous works [32, 30] in view-dependent LOD rendering. The method was specially adapted to large terrain meshes by partitioning the model into separate blocks which are combined at run-time. To avoid tears or cracks between adjacent blocks, the simplification of vertices lying on a block boundary are not allowed. After the preprocessing step, a coarse base mesh together with an ordered list of vertex split operations are obtained, which allows the local refinement or simplification of the surface vertices. A screen-space criterion was employed as error metric for the refinement operations. Since this algorithm was very successful at that time, in the following years different authors developed similar or derived methods, such as [21, 35, 33].

The same idea of combining a hierarchy of binary trees or quad-trees with cached groups of geometric primitives, featured by some grid-based multiresolution method, as mentioned

above, has been also employed with irregular datasets. The *BDAM* method described in [13, 14] is one of the most relevants and efficient algorithms. In *BDAM* the nodes of the regular hierarchy are formed by precomputed irregular patches of small surface region. Similarly, in [39], a previous work featuring an efficient quadtree triangulation hierarchy over an irregularly sampled point set presented in [59] was extended to use a hierarchy of blocks with pre-calculated triangulations instead of a hierarchy of vertices.

1.4 Hybrid meshing rendering

Multiresolution modeling is an intelligent strategy for dealing with the processing and rendering of large terrain models. However, using LOD representations of high-resolution grid-based terrain surface models is not always optimal to represent terrain areas with very irregular features. The grid terrain models commonly used in multiresolution modeling can only approximate these features if the resolution largely increases, which implies much larger storage requirements and memory accesses, even when view-dependent multiresolution optimizations are employed. TIN models, on the other hand, are more topologically and geometrically precise in representing complex structures. Using hybrid terrain models combining the benefits of both representations seems a more natural and efficient way of dealing with this kind of terrains.

However, direct rendering of hybrid terrain models can generate meshes with holes and geometric discontinuities between the borders of the different parts. Thus, some kind of strategy is needed to attain a coherent mesh during the visualization. In [5], for instance, it is pointed out that component meshes of a hybrid terrain model should be locally adapted in order to avoid artifacts at the boundaries. In [20], the authors suggested an adaptive tessellation procedure to avoid discontinuities in the borders between the different representation models. In order to connect the meshes, they suggest generating new triangles joining their boundaries, but they do not propose any specific tessellation algorithm to perform the task. A more specific approach for the generation and visualization of hybrid terrain models of geological faults is presented in [87]. This method, which is also able to cope with heterogeneous data sources, uses a combination of multiple planes to approximate the complicated surfaces and geometric shapes of geological faults.

Since the advantages of using hybrid models to integrate heterogeneous terrain data sets have been barely studied in the literature, the objective of this thesis is to develop a hybrid

terrain model approach for the interactive visualization of terrains integrating heterogeneous data sources. An additional benefit derived of the followed approach consists in the preservation of the original data integrity, since it does not require the modification of the original datasets. Hence, the original Hybrid Meshing (HM) algorithm with the extensions developed in this thesis, as well as the new EDP algorithm, the other important line of research followed during the development of this work, represent a valuable contribution to the geovisualization and terrain rendering research fields. This dissertation first focus on the HM algorithm and the work developed to improve and optimize the algorithm, and then presents the work around the new method for the visualization of hybrid terrain models developed in this thesis, the EDP algorithm.

Chapter 2 contains a complete review and analysis of the original HM algorithm based on previous publications [9, 2, 8], as the departing point of this thesis. The HM algorithm is a modern approach for integrating a base grid model with high resolution TIN meshes from different datasets in a single, coherent, crack-free hybrid terrain model. It supports multiresolution rendering in the large grid part and achieves interactive rendering of the whole hybrid model by using a local tessellation strategy, based on the grid cells, and assisted by a new theoretical hardware unit in the graphics pipeline. A local adaptive tessellation of the partially overlapped grid cells is partially precomputed in a preprocessing phase, and encoded with a LOD-independent data encoding technique. Thus, the local tessellation can be selectively reconstructed during the interactive visualization and a coherent hybrid model is obtained without requiring an explicit remeshing of the original data.

Chapter 3 presents two different approaches of a hybrid terrain model visualization software. The first approach is based on the HM algorithm, adapting the original method to a conventional graphics pipeline, without requiring the proposed hardware extension. The second implementation presented in this chapter uses a conventional tessellation algorithm to generate a coherent hybrid model during the visualization process. This method does not require the partial precomputation of the local tessellations in the grid cells, but also manages to visualize a crack-free hybrid model. This work has been published in [60]

Chapter 4 describes the GPU-HM method, a parallel implementation of a hybrid terrain model visualizer running on the GPU. This proposal is based on the original HM algorithm but adds several enhancements such as a more compact encoding of the preprocessed data lists, an optimized memory access pattern, a more efficient decoding of the data and, the

most important, it manages to perform several local adaptive tessellation in parallel using the Geometry Shader unit of the GPU. This work has been published in [62].

Chapter 5 introduces the EHM algorithm, a new method to visualize hybrid models supporting multiresolution rendering of the high resolution TIN meshes as well as the base grid. The EHM method achieves the complete multiresolution view-dependent rendering of a hybrid model by ensuring that the precomputed local tessellation of the TIN boundary can be reconstructed although some vertices are missing. This is enforced by simple preconditions during the generation of the multiresolution representation of the TIN part. The work related to the EHM algorithm has been published in [61].

Chapter 6 details the EDP algorithm, a new approach for generating hybrid terrain models not related to the HM algorithm. The EDP method overcomes some limitations of the HM algorithm by using edges as the new basic primitive to join the boundaries of the regular grid and the irregular high-resolution TIN meshes. This approach manages to avoid the insertion of additional vertices in the irregular mesh as well as simplifies the preprocessing phase and achieves a better parallelism during the rendering phase. The work related to this algorithm is in the process to be submitted to publication[63].

CHAPTER 2

HYBRID MESHING ALGORITHM

The interactive rendering of hybrid terrain models is a barely studied subject in the field of terrain visualization. The Hybrid Meshing (HM) algorithm was presented in [9] by members of the Computer Architecture Group in University of A Coruña and University of Santiago de Compostela, in collaboration with Prof. Dr. Jürgen Döllner of the Computer Graphics Systems in the Hasso-Plattner-Institut. It is, to our knowledge, the first method for the real-time rendering of hybrid terrain models that does not require the elaboration of a complete new model based on the geometry of the source models. The HM algorithm is capable of rendering multiresolution hybrid models formed by a base multiresolution regular grid mesh with highly detailed TIN meshes embedded.

The HM algorithm uses an efficient data representation scheme which is not dependent on the selected LOD for the multiresolution regular part. Instead, an adaptive tessellation is generated to join the regular and irregular parts of the model. This adaptive tessellation is partially computed during a preprocessing stage, and encoded using a specifically data structure which allows the reconstruction on demand during rendering. Since the tessellation is computed in local patches and generation is made on demand fashion, the adaptive tessellation may adapt to different cell sizes in the regular part.

The original approach proposed an implementation of the method using a new simple hardware extension in the graphic pipeline to achieve interactive visualization of high quality hybrid terrains [2]. This thesis takes the HM algorithm as the starting point to develop an efficient proposal for rendering hybrid terrain models. During this thesis, the HM method has been implemented, evaluated and improved in terms of performance (an efficient implemen-

tation using GPUs) and flexibility (a new approach using multiresolution rendering methods in both the base grid and the TIN meshes). Furthermore, a new different approach has been proposed to solve some of the limitations detected in the original algorithm.

Since the HM algorithm is the starting point of this thesis, this chapter is devoted to explain the algorithm in detail. Furthermore, a deep knowledge of the HM algorithm is needed to understand the problems solved in this work and the new features and enhancements developed. An overview of the structure and the different phases of the algorithm is presented first. Next, the specific steps of the processing performed at each phase are explained, as well as the model representation and the encoding of the data structures. At the end of the chapter, a detailed analysis of the algorithm is presented.

2.1 HM Algorithm overview

The HM algorithm works following a local strategy to perform an adaptive tessellation between the borders of the different parts of the model, which is simple and fits well for LOD rendering. Taking into account that the regular grid can be represented with different resolutions, computing and sending the tessellations required for a specific LOD would not be efficient. In the solution HM algorithm proposes, a unified representation of all LOD-dependent tessellations is generated. This information is employed by the proposed hardware extension in the GPU to decode the adaptive tessellation required for a specific LOD.

This is schematically depicted in Figure 2.1. The TIN and grid meshes are sent to the graphics pipeline together with the unified tessellation representation. The primitives associated with the adaptive tessellation are decoded and generated in the GPU, which performs among other tasks the joint of the borders between the non overlapped parts of the input models. Moreover, only the existing vertices in the LOD approximation of the grid selected for the rendering are employed in the process.

Appropriate tessellations for joining every LOD of the grid model are computed and encoded using a cell-based strategy, following an efficient and level independent scheme, during an initialization phase. Then, during visualization, the triangles required for the selected LOD are decoded on demand from this unified representation, as represented in Figure 2.1. The final hybrid model is formed by the union of the view-dependent refinement of the grid and the single-resolution TIN, linked in a single, coherent mesh.

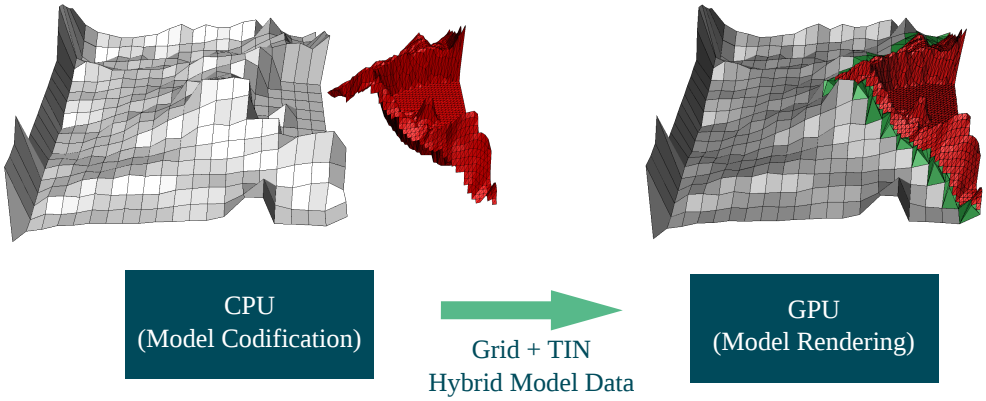


Figure 2.1: HM algorithm structure.

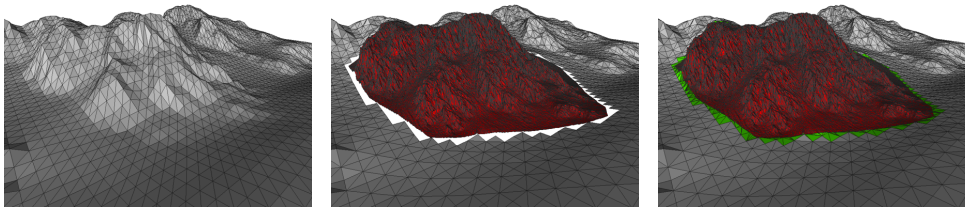


Figure 2.2: HM algorithm example: (a) base regular mesh; (b) base regular mesh and detailed TIN mesh; (c) single coherent mesh obtained by the HM algorithm.

Therefore, the final hybrid model is formed by the union of the view-dependent refinement of the grid, the single-resolution TIN, and the local adaptive tessellation between their borders. Furthermore, this method does not alter the original data and it also supports multiresolution rendering in the grid model. Figure 2.2 depicts an example of the source regular (a) and irregular (b) parts of the model and the final mesh rendered by the HM algorithm (c).

The rendering steps followed by the algorithm for every frame are summarized in Figure 2.3. First, the new level-of-detail approximation of the grid mesh is computed depending on the view conditions and the regular multiresolution algorithm employed. Note that the HM

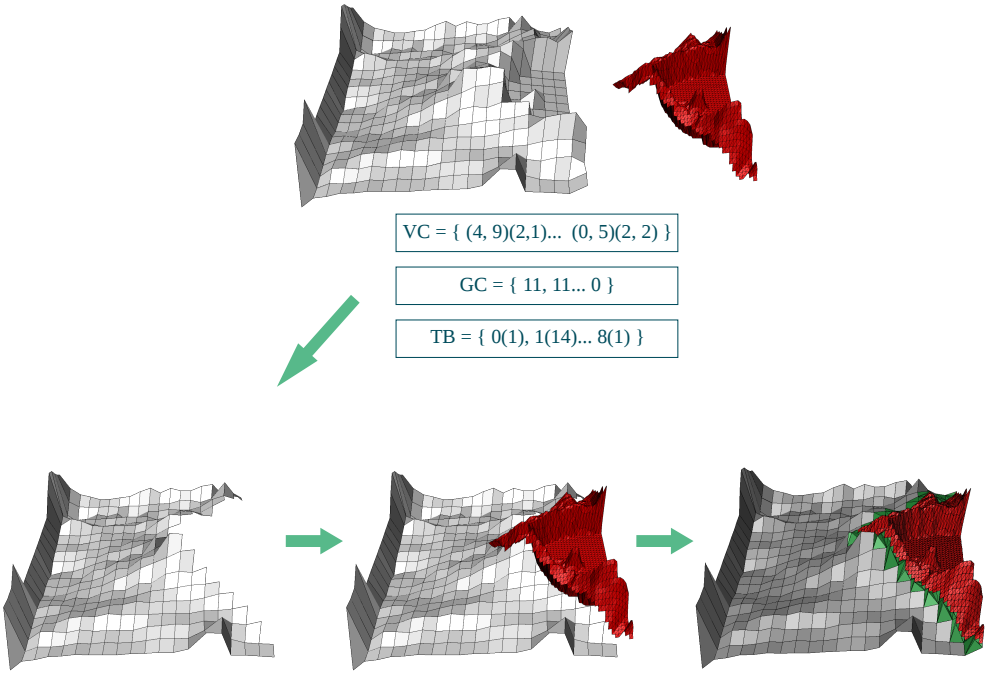


Figure 2.3: HM algorithm rendering steps.

algorithm does not rely in any specific implementation detail of the multiresolution rendering approach providing it is a cell-based method. Once this step is finished, the grid cells Non Covered by the TIN (NC cells) are sent to the next step in the rendering pipeline, while the completely covered grid cells (CC cells) are discarded, as they will be replaced by the more detailed TIN data. Finally, the TIN part is also rendered and the partially covered grid cells (PC cells) where the borders of the TIN lay, are adaptively tessellated to join both parts.

To perform the tessellation of the PC cells, the HM algorithm is based on local convexification of the TIN boundary inside each cell. The convexification of triangle meshes is a well studied but complex subject for which several solutions have been developed [74, 55]. In the HM proposal, the convexification for the different levels of detail are computed and encoded in a compact and efficient way as a pre-processing step in the CPU. This is possible because the convexification is performed incrementally from the finest to the coarsest level-of-detail

associated with the grid. This strategy permits the generation of a unified representation of all convexification levels and, on the other hand, permits the generation of coherent transitions between contiguous cells corresponding to different LODs.

Therefore, rendering procedure used in the HM algorithm consists of two steps: the local convexification of the TIN boundary (TB) and inclusion of additional triangles to to join the grid cells and the resulting local convex hull of the TIN. Triangles associated to the convexification of the TB are generated during the run-time visualization by means of simple decoding operations of the precomputed data structure. Once the TB is turned into a convex structure, the remaining connection between TB and PC cell corners is easy to accomplish by generating *corner triangles* between the cell corner vertices and the convex TB.

In the following subsections all these steps are explained. For the sake of clarity, the order is deliberately change to begin with the final tessellation process performed after the TIN boundary has been convexified.

2.2 Adaptive tessellation of the grid cells

Assuming that the TIN has been previously convexified, corner tessellation triangles are formed during the tessellation process to join the uncovered corners of the cell with consecutive vertices of the convex TIN border. The tessellation algorithm starts connecting the vertices of the already convexified TB with the first uncovered corner of the cell in clockwise order. This starting corner is stored in one of the different lists encoding the relevant information of the ongoing process of tessellation. Then, it continues linking consecutive vertices of the TB while the introduced triangle does not overlap with the TIN. When this happens, a *shift* in clockwise order to the next cell corner is made before continuing with the triangulation.

Since the TIN boundary is at this point of the process a convex structure, it is easy to detect the overlapping situation by evaluating the angles between the corner and the consecutive vertices of the TIN, which is implemented in a effective way testing the sign of the cross product of consecutive edges in the TIN boundary.

Figure 2.4 shows an example where a PC cell and the corresponding convex TIN silhouette are depicted. The TIN boundary vertices $(1,2,3,4,5)$ and the cell corners uncovered by the TIN (c_3,c_0,c_1) are processed in clockwise order. The tessellation algorithm connects the list of corners with the list of vertices, following a sequential order. Hence, c_3 corner is consecutively connected with the vertices of the boundary until an overlap in the mesh is

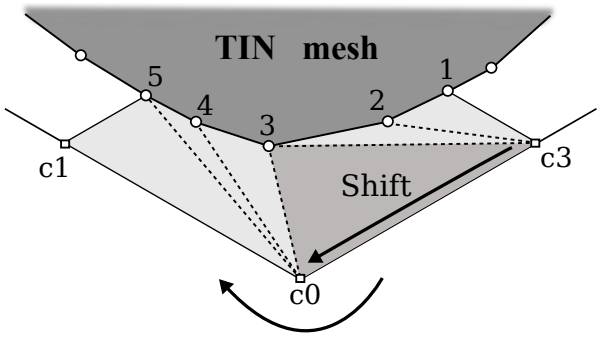


Figure 2.4: Cell tessellation.

detected. In this example, corner c_3 is connected with vertices 1, 2 and 3, but not with 4 because the new triangle $(c_3, 3, 4)$ would overlap with the TIN. Consequently, corner c_0 is selected and connected with 3 and 4.

2.3 Incremental convexification generation

In the previous section has been explained the final procedure of the adaptive local tessellation algorithm, which requires a properly convexified TIN to perform the triangulation of the PC cells. Since the local convexification of the irregular boundary of the TIN is a potentially costly operation, it is only calculated once in the preprocessing phase. The resulting convex border is stored in a simple, efficient and lightweight data structure, encoded for all the different detail levels of the grid, as is described in the following section. Similar procedures were previously proposed in the context of non-convex tetrahedralmesh visualization [38, 37].

The convexification of the TIN is performed incrementally for each PC cell between the regular and irregular part of the model, starting from the finest grid resolution level. For each cell at the current level-of-detail the convex hull of the TIN boundary is computed. To simplify the operation, the HM algorithm assumes that a TIN boundary vertex exists in the intersection point between the TIN and grid cell boundaries. If it is not the case, these vertices are trivially included before the beginning of the procedure. After this, the concave parts in the local convex hull of the TB or *caves*, are identified and triangulated. The generation of these triangles inside the caves may be performed indifferently by any tessellation algorithm [40, 74, 55]. Finally, the following coarser levels are processed using an incremental

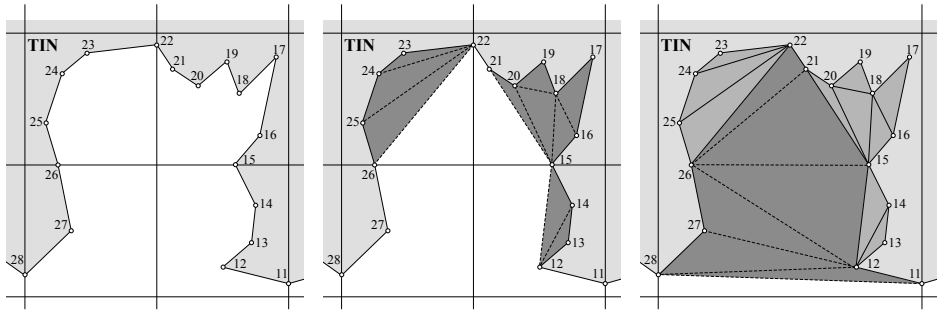


Figure 2.5: Incremental convexification of the TIN boundary: (a) original cells; (b) finest LOD convexification; (c) next LOD convexification.

strategy that preserves the triangles generated during the convexification of previous levels. This incremental procedure continues for each cell of each level of the grid, till the coarsest level-of-detail is completed.

Figure 2.5(a) shows four cells corresponding to the finest resolution level of a grid and the corresponding TIN silhouette covering the area; TIN is depicted in gray and the TIN boundary is explicitly marked using circles to indicate the vertices. In this example, local convex hulls are delimited by vertices $\{11,12,15\}$ (down-right cell), $\{15,21,22\}$ (up-right cell), $\{22,26\}$ (up-left cell) and $\{26,27,28\}$ (down-left cell). For the sake of clarity, TIN boundary vertices are represented in the examples by their indexes in the TB array. The triangulation of the caves after preprocessing the finest LOD are displayed in Figure 2.5(b). In 2.5(c), the following coarser level-of-detail is analyzed and the new local convex hull is determined by vertices $\{11,28\}$. New convexification triangles are shown with a different color of the ones generated in previous steps.

Note that following this incremental convexification approach, the convexification triangles corresponding to the different levels of detail may be efficiently encoded in a unified representation. Furthermore, since cell borders are attached to preserved vertices of the TIN boundary, transitions between different levels of detail in contiguous cells can be handled without discontinuities.

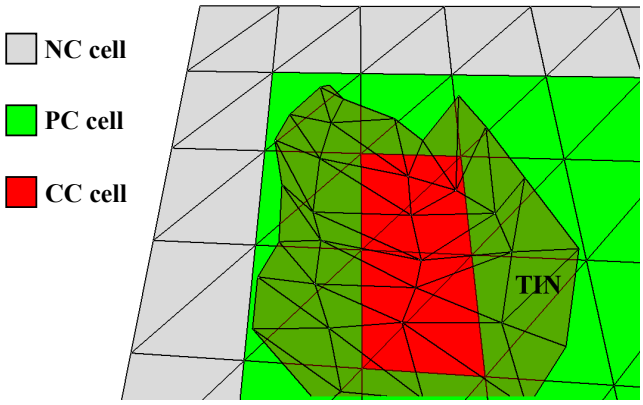


Figure 2.6: Grid cell types according to the coverage pattern with the TIN silhouette.

2.4 Hybrid model representation

In the HM algorithm the adaptive tessellations are mostly pre-computed and the resulting triangles can be efficiently encoded in a unified representation. This information, together with the grid and the TIN, is sent to the graphics card to generate all the adaptive tessellations required for any LOD. In the following, the additional information to be pre-processed and sent from the CPU to the graphics card is indicated.

2.4.1 Grid classification list

The relationship and overlapping pattern of the regular and irregular parts of the hybrid model is determined by the classification of the grid cells as PC, NC and CC, as it is depicted in Figure 2.6. Since these different kind of cells will be processed in different ways, they have to be easily identified. The Grid Classification List (GC) is employed for this purpose.

The GC list is the sequence of codes of the different cells of the regular grid for a given LOD. As usually the TIN only cover small sections of the terrain model, a variable length code is used, assigning the shortest code associated to the NC cells: 0 code for NC cells, 10 for CC cells and 11 for PC cells.

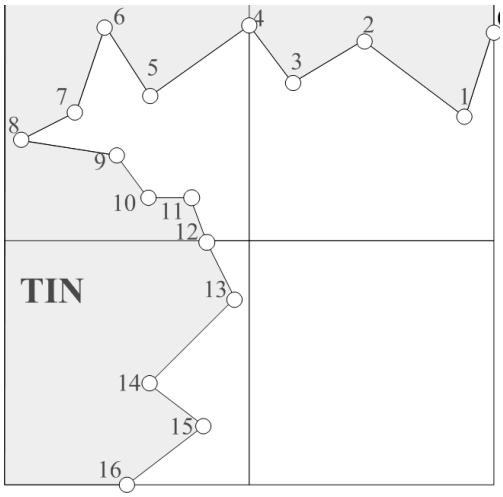


Figure 2.7: Detail of PC and CC cells contained in a fragment of a real TIN boundary.

As an example of a real GC list, the four cells of Figure 2.7 could be represented as (in row order, from left to right):

$$GC = \{11, 11, 11, 0\} \tag{2.1}$$

This list indicates that the first three cells have to be adaptively tessellated while the fourth one can be directly rendered.

2.4.2 TIN boundary

The encoding of the convexification triangles is accomplished by means of the list of TB vertices together with some additional connectivity information. Assuming that the TB is stored following a clockwise ring structure, the connectivity associated to each vertex indicates the distance, that is, the number of vertices between that vertex and the most distant one in the ring connected to it. This way, if connectivity of vertex v_i is j, it means that the farthest vertex in the ring connected to it is v_{i+j}. This connectivity value is the only additional piece of information to be stored per boundary vertex.

Let us consider again the example described in Figure 2.5 to illustrate this storing strategy. The TB corresponding to the TIN depicted in that figure can be represented with the TB list:

$$TB = \begin{array}{l} \{ \ldots\ 11(17), 12(16), 13(1), 14(1), 15(11), 16(2), 17(1), 18(2), 19(1), \\ \quad 20(1), 21(5), 22(4), 23(1), 24(1), 25(1), 26(1), 27(1)\ \ldots \} \end{array} \tag{2.2}$$

where connectivity value of each vertex is indicated within brackets. For example, vertex 15, with a connectivity value of 11, is connected with vertex 26 and all the vertices between them not placed inside a nested cave. Nested caves are simply detected by looking for vertices with connectivity values greater than one, which indicates that the vertex is the beginning of a cave. In this case, connectivity values indicate three caves: between 16 and 18, between 18 and 20 and between 21 and 26. The algorithm assumes a sequential connection of the cave starting vertex to the following vertices until the cave ending, but this connecting structure is broken if nested caves exist. For this example, vertex 15 is connected to all vertices between 16 and 26 not belonging to a nested cavity, that is: $\{16, 18, 20, 21, 26\}$.

Note that the connectivity values of the coarsest LOD can be also used for the tessellation of a different LOD, as it is explained in [9]. Hence, this unified representation of the local tessellation data triangles may be employed for the reconstruction of TB convex hull at different levels of detail.

2.4.3 Vertex classification list

Finally, one complementary list is used to classify the vertices implied in the local PC cell tessellations: the Vertex Classification (VC) list. This list maps the TIN Boundary vertices to the corresponding cell of the grid to be employed in the local tessellation.

That is, since the TB vertices, as well as the cell corner vertices, are sequentially numbered in clockwise order, the information related to a PC cell can be encoded as a 4-tuple $\{A, L, C, I\}$, where A represents the first TB vertex falling into the cell, L is the total number of TB vertices present in the cell, C is the first corner vertex not covered by the TIN mesh, and finally I is the number of uncovered corners in the cell. Note that the corners notation employed is: 0 (up-left), 1 (up-right), 2 (down-right) and 3 (down-left).

For example, the PC cell depicted in Figure 2.8 can be represented, following this scheme, as the tuple $\{10, 7, 1, 3\}$, since it contains the TB vertices from 10 to 16 and the uncovered cell corners from $c1$ to $c3$.

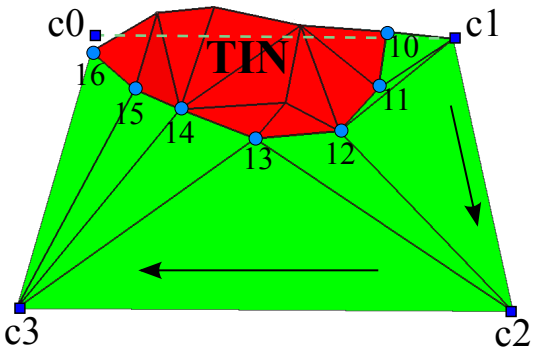


Figure 2.8: Local cell tessellation in the HM algorithm.

2.5 Rendering hybrid meshes with the HM algorithm

Due to the efficient encoding of the most complex tessellation operations using the connectivity indices, the HM algorithm is able to generate the adaptive tessellation and rendering the content of a PC cell by using simple decoding operations. The structure of this PC cell rendering algorithm is shown in the pseudo-code listing of Figures 2.9 and 2.10.

2.5.1 Convexification triangles generation

The algorithm involved in the reconstruction of the triangles associated with the convexification is presented first (see the algorithm pseudocode in Figure 2.9). In the listing, the TIN boundary vertices inside the cell are represented as $\{v_n, \ldots, v_{n+L-1}\}$ and their connectivity values as $\{v_n.c, \ldots, v_{n+L-1}.c\}$. In addition, T_1 , T_2 and T_3 denote the vertices of each triangle T forming part of the generated tessellation. The starting vertex of each nested cavity under construction is stored in an array called S_k with $k > 0$. $S_k.count$ temporally stores the already processed vertices inside a cavity once a nested cavity k is opened.

The TIN boundary vertices forming part of the cell are sequentially processed $\{v_n, \ldots, v_{n+L-1}\}$ (line 2). The vertices not included in any cavity are considered part of the convex hull (lines 4 and 5). Once a cavity starts, the following vertices build the triangles in the cavity under construction (lines 7 to 13). Additionally, when a vertex closes a cavity, the starting vertex of the cavity is eliminated from the S list and an additional external triangle is identified (lines 14 to 26). Finally, if the connectivity of the vertex is larger than one it is

```
1: k ← 0
2: count ← 0
3: triangle ← { , , }
4: S ← [ ]
5: Send ← [ ]
6:
7: {– TIN boundary convexification –}
8: for all vertex i with i = n,...,n+L−1 do
9:    count ← count + 1
10:   if k = 0 then
11:      AddToConvexHull(vertex i)
12:   else {– Building triangles inside cavities –}
13:      if (vertex i).conn = 1 then
14:         if triangle is complete then
15:            Generate(triangle)
16:         else
17:            AddToTriangle(vertex i, triangle)
18:         end if
19:      end if
20:      while count >= (vertex S_k).conn do {– Closing a cavity –}
21:         triangle ← {vertex S_{k−1}, vertex S_k, vertex i}
22:         Generate(triangle)
23:      end while
24:      k ← k − 1 {– Eliminating starting vertex –}
25:      count ← count + Send_k {– Updating count of current starting vertex –}
26:      AddToTriangle(vertex i, triangle)
27:      if k = 0 then
28:         AddToConvexHull(vertex i)
29:      end if
30:   end if
31:
32:   if (vertex i).conn ≠ 1 and (vertex i).conn + i < n + L − 1) then {– Starting a cavity –}
33:      Send_k ← count {– Storing previous counter –}
34:      k ← k + 1
35:      S_k ← i {– Starting vertex storage –}
36:      count ← 0
37:   end if
38: end for
```

Figure 2.9: Pseudocode for the TIN boundary convexification part of the tessellation algorithm for partially covered grid cells.

```
1:
2: {– Generation of corner triangles –}
3: triangle ← {hullVertex 0, corner 0, hullVertex 1} {– First triangle –}
4: Generate(triangle)
5:
6: j ← 0 {– Corner index –}
7: m ← Length(ConvexHull)
8: for all vertex i with i = 1,...,m−2 do {– Intermediate triangles –}
9:    triangle_1 ← vertex i
10:   sign1 ← Sign(CrossProduct(vertex i − corner j, corner j+1 − corner j)_z)
11:   sign2 ← Sign(CrossProduct(vertex i − corner j, vertex i+1 − corner j)_z)
12:   if sign1 = sign2 then
13:      triangle_2 ← corner j
14:      triangle_3 ← vertex i+1
15:      Generate(triangle)
16:   else {– Corner shift –}
17:      triangle_2 ← corner j
18:      triangle_3 ← corner j+1
19:      j ← j+1
20:      i ← i−1
21:      Generate(triangle)
22:   end if
23: end for
24: while j < r−1 do {– Remaining triangles –}
25:   triangle_1 ← vertex m−1}
26:   triangle_2 ← corner j
27:   triangle_3 ← corner j+1
28:   Generate(triangle)
29:   j ← j+1
30: end while
```

Figure 2.10: Pseudocode for the generation of corner triangles part of the tessellation algorithm for partially covered grid cells.

included in the S list as starting point of a new nested cavity (lines 30 to 38). It can be derived from looking the algorithm pseudo-code, the computational requirements are quite low, since only additions and comparisons are involved.

2.5.2 Corner triangles generation

The corner tessellation procedure receives the TIN boundary vertices of the convex hull (after the convexification procedure) together with the sorted corners for the tessellation procedure (information contained in the VC list). The tessellation algorithm involves different steps indicated in the pseudo-code of Figure 2.10. Note that the vertices of the convex hull have been relabelled with consecutive superindices to simplify the explanation. We employ superindices to emphasize that convex hull vertices do not necessarily correspond to consecutive vertices of the TIN boundary. We consider m vertices v^i with $i = \{0, \dots m-1\}$ in the convex hull. The r uncovered corners are also relabelled for simplicity as c^j with $j = \{0, \dots r-1\}$.

The first triangle is composed of the first corner and the first two vertices (lines 1 to 5). After this the following vertices of the convex hull (line 8) are processed to compose the intermediate triangles.

The corner shift control is performed through the evaluation of the sign of the z-coordinate of two cross products (lines 10 and 11). When no corner shift is required, the triangle is built by the current corner and two consecutive vertices (lines 12 to 15). When a corner shift is required, the triangle is built by the two corners and the current vertex (lines 16 to 22). This vertex is reemployed as part of the following triangle (line 19).

Once all convex hull vertices have been processed, the remaining non-processed corners have to be considered (lines 24 to 28). The final triangles are composed of the last vertex of the convex hull and two consecutive corners.

This simple procedure permits the reconstruction of the triangles associated with the convex hull of the TIN boundary and the corner cells. The computational requirements of the algorithm are very low, as they are mainly associated with the corner shift control.

2.6 Hybrid extension with sub-cell generation

The original HM algorithm also features an extension to generate a hybrid structure per cell with sub-cells and triangles. The only additional requirement is the pre-computation of a tree

storing a sub-cell hierarchy and, as result, this additional information permits two different tessellation procedures.

The new procedure is based on the hierarchical/recursive subdivision of partially covered cells into sub-cells. The subdivision is repeated until the size of the grid cell size achieves the size of the finest LOD resolution. Then, to complete the procedure, the unit PC sub-cells are tessellated into triangles. The triangles generated represents the area not covered by the TIN and connect the sub-cell corners with the TIN boundary vertices.

The objective of this combined system is to preserve the original grid information where possible and adapting the structures only in close proximities of the TIN boundary. Thus, this technique attains the exploitation of the information offered by both representations limiting the areas of interconnection and manipulation of the original data.

Let us consider the example of Figure 2.11(a) to explain the extension. In this figure a PC cell of the grid and the TIN silhouette inside it are depicted. Let us label the current LOD of the grid representation as LOD 2 and let us assume that there are only two finer resolution LODs for the grid (LOD 1 and LOD 0). This means that the cell has to be recursively subdivided two times until the unit cell size (LOD 0) is achieved. In the first subdivision procedure (see Figure 2.11(b)) two *PC* and two *NC* sub-cells are identified. The *NC* sub-cells can be directly rendered while the *PC* sub-cells are subdivided again. As result (see Figure 2.11(c)), three *NC* and five *PC* unit size sub-cells are additionally generated. The *NC* sub-cells are sent to the graphics pipeline while the *PC* unit sub-cells are triangulated. This is indicated in Figure 2.11(d).

The tessellation of each unit size sub-cell is based on the already explained TIN convexification and corner tessellation procedures. This means that extending the methodology previously presented to this new tessellation procedure can be directly performed if a procedure for sub-cell generation is included. Note that due to the complexity of the algorithm involved, this proposed extension is performed as a pre-processing step and the result is efficiently encoded, following the standard HM algorithm strategy.

2.6.1 Sub-cell structure: Tree representation

The recursive subdivision to be performed in each *PC* cell can be efficiently stored in a tree representation. As an example let us consider the system of four cells and a TIN structure depicted in Figure 2.12(a). The *GC* list associated to this system is $0, 11, 0, 11$ where the cells are sequentially listed following a row order. As was previously indicated, each *PC* sub-

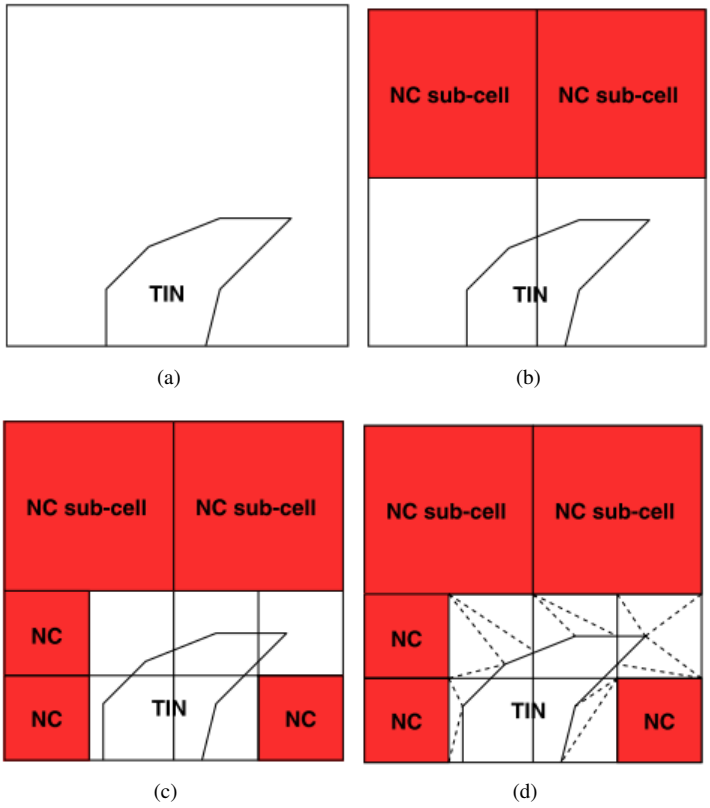


Figure 2.11: *PC* subdivision example for the hybrid algorithm.

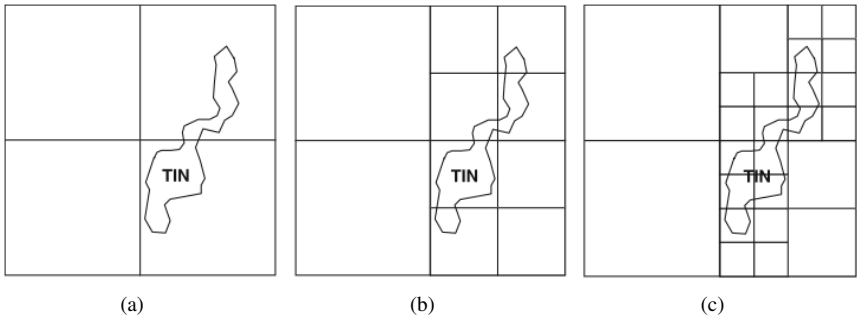


Figure 2.12: Grid subdivision example for the hybrid algorithm.

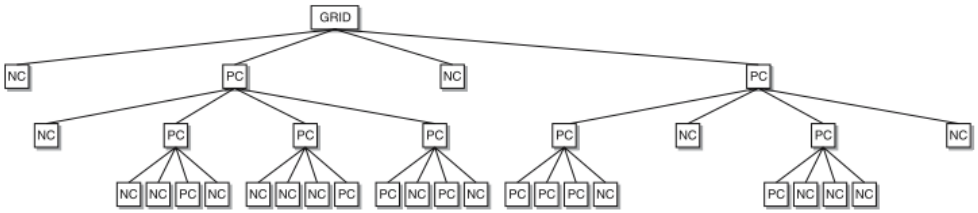


Figure 2.13: Tree representation.

cell is subdivided into four sub-cells and the procedure is recursively repeated until the finest resolution. Let us assume in this example that there are two finer LODs. The result of the first step of subdivision is depicted in Figure 2.12(b). That is, the up-right cell is subdivided into four sub-cells that are classified as (NC, PC, PC, PC) that is $(0, 11, 11, 11)$, and the right-down cell is subdivided into four sub-cells (PC, NC, PC, NC), that is $(11, 0, 11, 0)$. Each *PC* sub-cell is subdivided again obtaining the structure of Figure 2.12(c).

This structure can be efficiently represented in a tree structure. Each node of the tree represents a cell/sub-cell and each row of the tree a step in the subdivision process. For this example the tree is depicted in Figure 2.13. This tree can be schematically represented by the sequence of nodes in each row. Specifically for this example:

$$\begin{aligned} Tree = \quad & (0,11,0,11)(0,11,11,11)(11,0,11,0)(0,0,11,0)(0,0,0,11) \\ & (11,0,11,0)(11,11,11,0)(11,0,0,0) \end{aligned}$$

This tree contents all required information for the subdivision of each cell into sub-cells. This information, together with the local information for local convexification and corner tessellation of each unit size sub-cell permits the hybrid representation.

2.6.2 Tessellation algorithm

The tree storing the sub-cell structure is generated as pre-processing step. This information is employed during the rendering process to identify the *PC* cells, to subdivide them into *CC* sub-cells (to be eliminated), *NC* sub-cells (to be sent to the graphics pipeline) and *PC* unit sub-cells. The *PC* sub-cells are further subdivided. The process is repeated till the finest resolution level. The final *PC* unit sub-cells are tessellated according to the convexification and corner tessellation procedure previously presented.

Each cell subdivision implies the computation of the coordinates of five new points. To avoid cracks among cells simple linear interpolations are performed to generate the new sub-cells corners. This implies that the subdivision procedure only implies decoding the tree information and linear interpolation operations. Once the unit-size *PC* sub-cells are generated and identified, the convexification and corner tessellation procedure are performed (see Section 2.5).

2.7 Analysis of HM algorithm

In this section, the properties of the two tessellation variants of the HM algorithm, the HM based on triangle generation and the HM with sub-cells tessellation algorithms, are analyzed. They share the same core structure to perform the adaptive tessellation between the partially overlapped grid cells and the contour of the TIN: local convexification of the TIN boundary and corner tessellation inside the partially overlaped cells. The method is based on a mixed strategy where part of the information is pre-computed and only simple operations are performed during rendering, which leads to the efficient generation of an adequate tessellation for any level-of-detail in the regular mesh.

In terms of storage requirements, the data structures generated by the HM algorithm are strongly dependent on the input models. Thus, the size and structure of the TIN, particularly the number of the boundary vertices, are the most determining variables for the computation of these requirements. In the normal HM tessellation algorithm, based on triangles generation, the convexification and corner tessellation procedures require the computation of the TIN boundary (vertices and connections) and the GC and VC lists. The TIN boundary information is independent of the level-of-detail selected so that the information is calculated only once. In this case, the data structures storage depends on the number of vertices on the boundary of the TIN and the connectivity structure between them. The connectivity index range is determined by the maximum number of TIN boundary vertices per cell, therefore, assuming a maximum number of vertices L_{max}, the connectivity index associated with each TIN boundary vertex would require $log_2(L_{max})$ bits. For example, considering up to $L_{max} = 256$ only 8 additional bits per TIN boundary vertex are required.

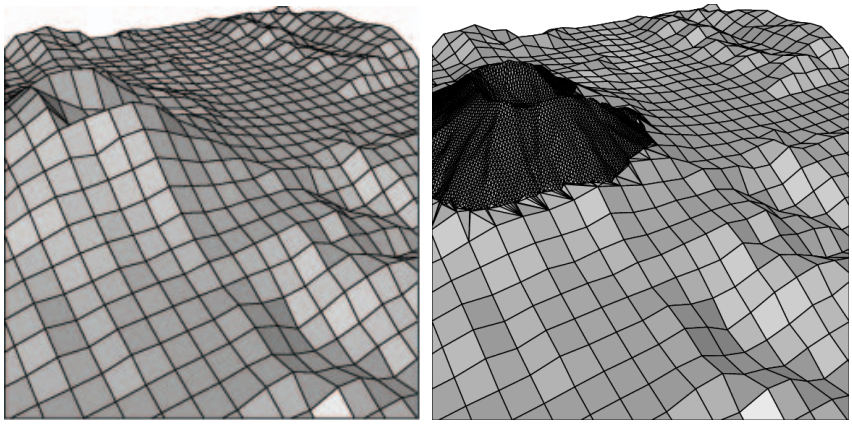
Regarding the GC list, the information to identify each cell as PC, CC or NC is needed. Since there are only 3 different types of cells to encode, a variable length code of up to two bits per cell is employed. The VC list stores additional information required per PC cell. This

information includes two fields, one for the addressing of the TIN boundary array (A,L), and the other for indicating the corners involved in the tessellation (C,I). The A value is determined by the number of vertices in the TIN boundary and depends directly on the TIN model. For example, working with a TIN model whose boundary contains 65k vertices, 16 bits are required to address a specific vertex. On the other hand, $log_2(L_{max})$ bits are required for encoding the section length. Continuing with the previous example, if $L_max = 256$ vertices only 8 bits are required. With respect to the corner information, 4 bits are required for encoding the C and I fields. In summary, in this case 24 bits per PC cell would be required for the VC list. These requirements are small if we take into account that 32 bytes are usually required for a vertex in a standard triangular mesh.

In the sub-cells tessellation algorithm, an additional tree structure has to be included for storing the sub-cells. Analyzing the storage requirements associated to this tree, we can see that the subdivision of one PC cell into sub-cells could generate up to 4^i PC sub-cells in each subdivision step$(i = 0,\ldots,N-1)$. As 2 bits are required per sub-cell, the storage requirements for the PC cells of the coarsest regular grid is up to: $\sum_{i=0}^{N-1} 2 \cdot 4^i$, considering a regular grid with N different LODs. For example, in the case of $N = 3$, the maximum number of bits per PC cell for the tree representation would be 42 bits. This in an upper limit representing the situation in which each PC sub-cell is subdivided recursively into four PC sub-cells. The requirements associated with the tree grow exponentially with the number of LODs. However, due to the low number of PC cells and PC sub-cells these requirements should be low in practical situations.

With respect to the quality of the models generated, the benefits of the two HM tessellation strategies are clear taking into account the increment in quality when the DEM representation is enriched with the available TIN meshes representing detailed areas of the terrain (see as an example Figure 2.14). The subdivision strategy generates finely detailed tessellations featuring a smoother transition between the grid and TIN parts and thus the user can choose between two tessellation procedures in real-time following quality criteria.

However, the HM with sub-cell generation algorithm generates also more triangles and requires a larger amount of data and pre-computations. The adaptive tessellation generated for both algorithms is noticeably different as it can be seen in the models represented in Figure 2.15. Three levels of detail were depicted for both algorithms, from finest (left) to coarsest (right). The first algorithm generates less primitives for rendering, while the second one reduces the area of triangle generation to the locality of the TIN contour. For example, for



(a) (b)

Figure 2.14: Hybrid model rendered with the HM algorithm: (a) base grid mesh; (b) grid and TIN meshes linked during rendering using the HM algorithm.

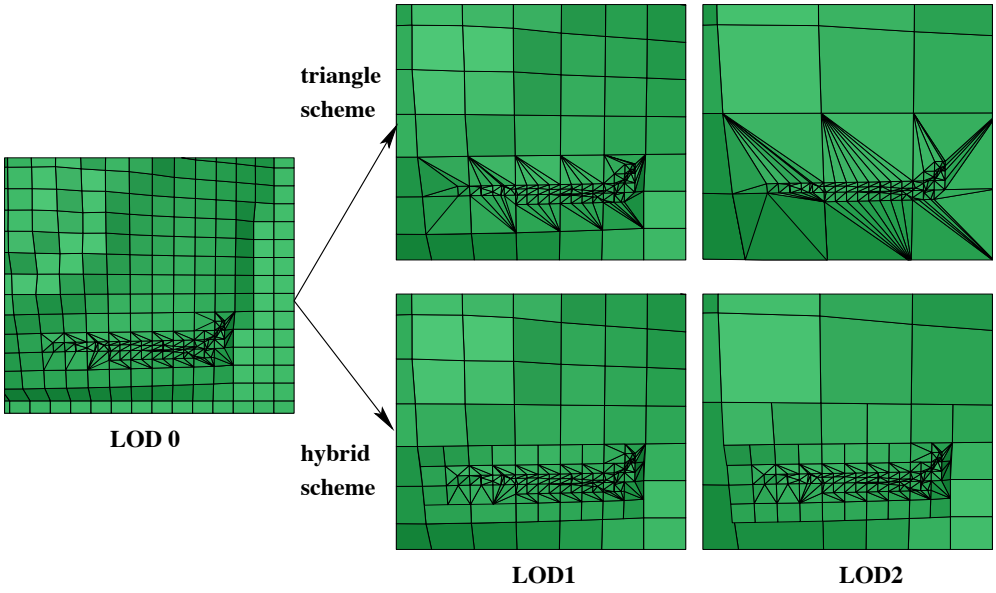


Figure 2.15: Adaptive tessellation example for both schemes.

the coarsest level-of-detail the number of triangles generated with the first algorithm is about 50% the number of primitives generated by the second algorithm. Moreover, for the second algorithm, at very coarse levels of detail the application would probably not represent the TIN since this level of quality is not required. Also note that although a reduced number of primitives for coarser levels of detail is usually desired, as this corresponds to areas that will be depicted with low resolution, using fewer triangles in the tessellation implies that triangles generated by the first algorithm are typically larger and narrow, and thus more likely to be conflictive sliver triangles. Hence, the benefits of the subdivision strategy are specially relevant when the difference in the size of the grid cell and TIN boundary edges is really evident, as it provides a more gradual transition between them. Since this is the major contribution of this strategy, and the same problem is better solved by the new EHM approach (see Chapter 4) developed in this thesis, the subdivision approach has not been considered as a useful in this research.

2.8 Conclusions

The HM algorithm achieves the hybrid representation of terrain models, where a multiresolution grid together with a high resolution TIN can be combined into a unified model, by performing the adaptive tessellation of each grid cell partially covered by the TIN, solely based on the TIN boundary information. High quality rendering is obtained, as no discontinuities are generated between representations. The method is based on the efficient representation of the information to be sent from the CPU to the graphics pipeline. The representation we propose can be easily extended to include different tessellation procedures. The operations to be performed to decode and complete the tessellation structure are very simple.

Although hybrid representation of terrains combining different data sets is of great interest, the traditional solution is the disconnected representation of the data sets or the preprocessing in software of the combined structure. These strategies are not appropriate when quality or real-time rendering are the objectives. We did not find any proposal in the bibliography that attempts to solve the challenge of real-time rendering of a multiresolution hybrid representation composed by TINs and DEMs. In this sense, no comparisons in terms of quality with other proposals can be performed.

CHAPTER 3

HM ALGORITHM: A GENERAL FRAMEWORK FOR RENDERING HYBRID TERRAIN MODELS

The HM algorithm, exhaustively reviewed in the previous chapter, is an efficient method for the interactive rendering of hybrid terrain models. The algorithm attains a perfectly watertight mesh by performing a local adaptive tessellation of the multiresolution grid cells placed in the boundary between the grid and TIN models.

The original HM algorithm, introduced in [9], was a proposal to render hybrid terrain models using the GPU to decode specially designed data lists, previously encoded in the CPU. Due to the lack of programmability of old GPUs, a new theoretical hardware unit was designed in [2]. In this chapter, the HM algorithm is used as the base component to build visualization software running in standard CPUs. The local adaptive tessellation approach followed by the HM algorithm is quite adaptable, as different algorithms can be implemented to join the TIN and grid boundaries inside the partially covered cells. Thus, two different implementations are presented in this chapter, both of them following the core HM proposal but attaining the local cell tessellation by different ways.

The first one follows the same approach used by the HM algorithm of joining both component meshes by generating local triangulations between their boundaries. However, this implementation uses a regular polygon triangulation algorithm to generate dynamically the adaptive tessellation triangles, instead of decoding a precomputed data structure representing

the convexification triangles, as the HM algorithm does. The second implementation tries to implement as faithfully as possible the original HM algorithm. It means that the complete execution of the algorithm is performed in the CPU and final mesh is sent to the standard graphic pipeline.

The results obtained for different hybrid models have been analyzed and compared for both implementations at the end of the chapter. Results confirm that the convenient properties of the original HM algorithm are accurately preserved by the software implementation, and the superior performance obtained by a hybrid terrain visualizer using the HM tessellation algorithm against one using a regular polygon tessellation algorithm. These two implementations were originally presented in [60].

3.1 Overview of the algorithms

The two visualizer implementations presented in this chapter manage to perform visually pleasant representations of a hybrid model formed by regular and irregular meshes following a local cell-based strategy.

The local strategy used in these implementations generates the hybrid mesh by performing a local tessellation of the grid cells containing the frontier between the meshes boundaries. This approach brings many advantages to the visualizer: it does not interfere with the rendering of the component meshes and thus can be used in conjunction with LOD rendering techniques; it is flexible since any tessellation method can be used to generate the triangles of the local tessellations; the resulting model uses the maximum resolution mesh available for every part of the terrain and, finally, since any kind of full remeshing procedure for the whole mesh is avoided, the approach is lighter in computational terms and also non-destructive, since the original data from the different meshes is not modified but integrated in a coherent whole. Moreover, an additional benefit associated to the local tessellation approach is that it does not enforce a strong dependency between the component meshes; thus, even if the high-resolution meshes were not temporally available, the visualizer could work by drawing on the low-resolution representation.

Thus, to generate the local tessellations of the cells containing the boundaries of the component meshes two different methods have been tested. The first one computes the additional triangles using a standard tessellation algorithm. There is a large number of tessellation methods to choose from since polygon tessellation is an important subject in computational ge-

	Seidel based implementation	Original HM
Preprocessing	Compute and encode preprocessed data: – Overlapping cells mask (GC list)	Compute and encode preprocessed data: – Extraction of TB boundary – Creation of GC list – Incremental convexification: VC and TB lists
Visualization	Render TIN and NC grid cells For every PC cell: – Generate trapezoidal decomposition – Partition in monotone polygons – Triangulate polygons	Render TIN and NC grid cells For every PC cell: – Convexification (TB array decoding) – Linking (VC list decoding)

Table 3.1: Structure of two different implementations of a hybrid terrain model renderer.

ometry [55]. In this case, the incremental randomized triangulation algorithm for polygons developed by Seidel [74] has been used, since is a well-known, simple and efficient method.

The second method adapts as close as possible the HM algorithm —explained in detail in Chapter 2— to a software implementation using conventional hardware. Note again that the original HM algorithm was developed as a hardware-based method using a new theoretical tessellation unit in the graphic processor.

These two implementations are based on the HM algorithm but they use different approaches to obtain valid tessellations of the hole between the grid and TIN meshes and thus differing on both the preprocessing and rendering phases. In the preprocessing phase, the *Seidel* based implementation computes the covering information between the grid and TIN models. On the other hand, the HM implementation uses the preprocessing phase to compute the incremental convexification of the TIN boundary, encoding the results using the efficient HM data lists.

Both implementations follow the same pattern of the HM algorithm in the rendering phase: first, the entire TIN and the non-covered cells of the grid (*NC cells*) are directly rendered; next, completely covered grid cells (*CC cells*) are eliminated as they will be replaced for the more detailed TIN data; finally, those grid cells partially covered (*PC cells*) by the TIN are adaptively tessellated and rendered. However, the adaptive tessellation between the TIN and grid is attained by different ways depending on the implementation. In the Seidel based implementation the rendering phase involves a heavy processing since the local tessellations are actually computed in this phase. The HM implementation generates the corner triangles according to a simple procedure and the convexification triangles by decoding the precomputed lists, which is an efficient and light-weight process.

An evaluation of the performance and the quality of the results obtained with the two implementations is presented at Section 3.4.

3.2 Implementation based on Seidel's Incremental Randomized Triangulation algorithm

The implementation based on the Seidel's Incremental Randomized Triangulation Algorithm does not use precomputed data structures besides the overlapping list used to mark the parts of the grid covered by the high-resolution TIN. This direct approach to the problem implies that, instead of decoding the convexification triangles first and then generating the corner triangles, the whole adaptive tessellation between the component meshes boundaries needs to be completely computed in the visualization phase.

In the preprocessing phase, the non-covered and partially covered parts of the grid are just identified and tagged. In the visualization phase, the same process is repeated for every frame: first, the multiresolution grid is updated according to the scene conditions; then the grid is rendered using the preprocessed coverage information to discard the covered and partially covered grid cells; once the non-covered grid cells have been rendered, it is turn for the whole TIN mesh; finally, Seidel's triangulation algorithm is used to generate the tessellation between both meshes.

As mentioned above, the tessellation process is performed in a local basis for the partially covered cells. The first step is the identification of the polygon to be triangulated formed by the non-covered cell corners and the vertices of the TIN boundary contained in the cell; that is, the points whose projection over the XY plane fall in the interior of the cell. An example

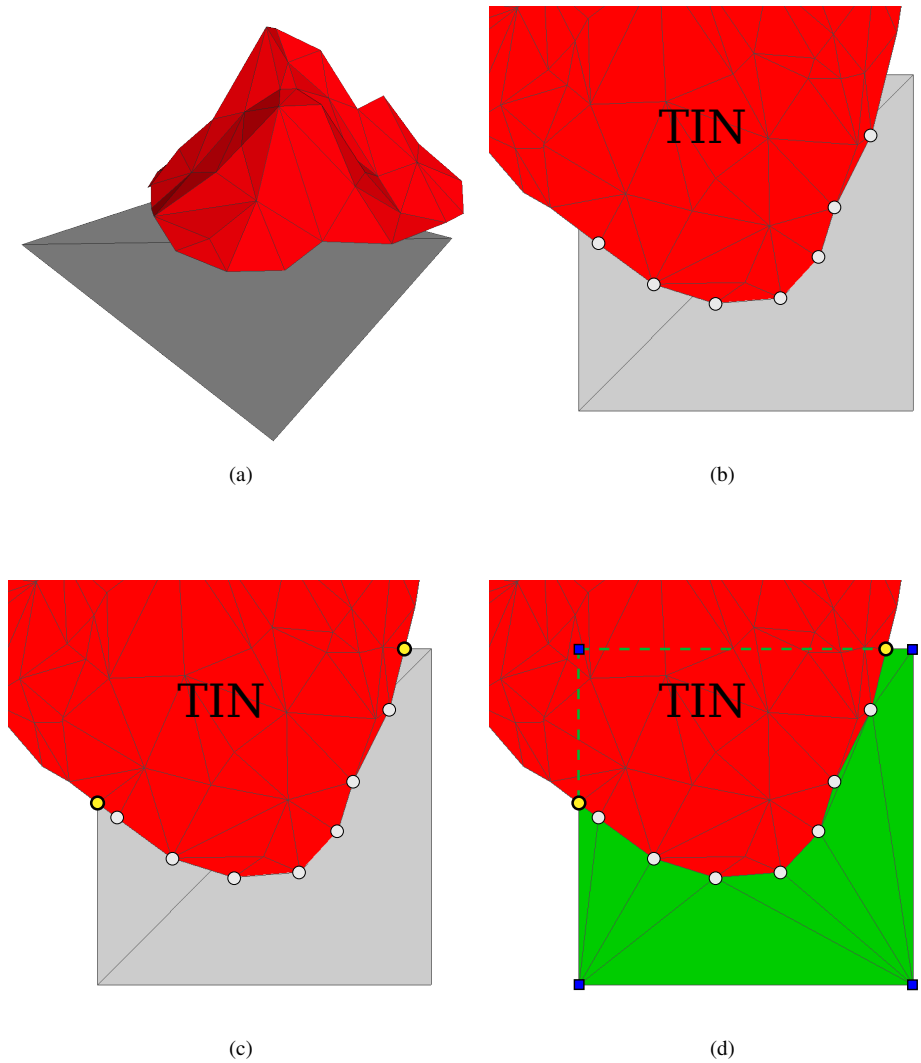


Figure 3.1: Local cell tessellation procedure using Seidel's algorithm: (a) part of the TIN boundary and the correspondant grid cell; (b) 2D projection of the TIN boundary over the XY plane; (c) addition of the extra vertices placed in the cell boundaries; (d) local tessellation of the grid cell.

of the whole process is depicted in Figure 3.1. Part of the TIN boundary and the original grid cell are shown in Figure 3.1(a) and then projected in the XY plane in Figure 3.1(b)); then, additional inter-cells vertices —represented in yellow in Figure 3.1(b))— are added to the TIN boundary; finally, the Seidel tesellation algorithm is used to generate the triangles linking the TIN boundary vertices and the grid cell corners, depicted in blue in Figure 3.1(d).

Once the polygon representing the hole between both boundary meshes has been identified, the tessellation algorithm is used to generate a triangle decomposition which is immediately sent to the render pipeline. Obviously, since the cell vertices forming part of the polygon depend on the active LOD in the grid, the tessellations are recalculated every frame.

With real-time rendering in mind, the Incremental Randomized Triangulation algorithm [74] was chosen for this implementation among all other polygon tessellation algorithms. The algorithm is explained in detail in the next subsection.

3.2.1 Analysis of the Incremental Randomized algorithm

This algorithm has been widely implemented due to its simplicity and efficiency and presents some convenient theoretical features. It runs in almost linear time for any simple polygon with n vertices, despite the theoretical complexity is $O(n \cdot log^* n)$. Let $log^{(i)} n$ denote the ith iterated logarithm, i.e. $log^{(0)} n = n$ and for $i > 0$ we have $log^{(i)} n = log(log^{(i-1)} n)$. For $n > 0$ let $log^* n$ denote the largest integer l so that $log^{(l)} n \geq 1$, and for $n > 0$ and $0 \leq h \leq log^* n$ let $N(h)$ be shorthand for $\lceil n/log^h n \rceil$.

In practice, it shows a linear runtime performance for simple polygons of n vertices, even though the theoretical complexity of the algorithm is higher. The Incremental Randomized Triangulation algorithm generates a triangulation without creating new vertices by decomposing the polygon in $n-2$ triangles. Using this method the triangulation of a polygon P is performed by creating first a trapezoidal decomposition of P, from which is easy to perform a triangulation of the original polygon in linear time. This trapezoidal decomposition is sometimes referred to *horizontal visibility map*, since edges of P that are horizontally visible are connected by segments. The following is a detailed description of the algorithm.

The algorithm proceeds in three progressive steps as is depicted in Figure 3.2. First, it decompose the polygon into trapezoids, next decompose the trapezoids into monotone polygons and, finally, triangulate the monotone polygons, which can be performed in linear time by using a greedy algorithm which progressively eliminates the convex corners of the polygon.

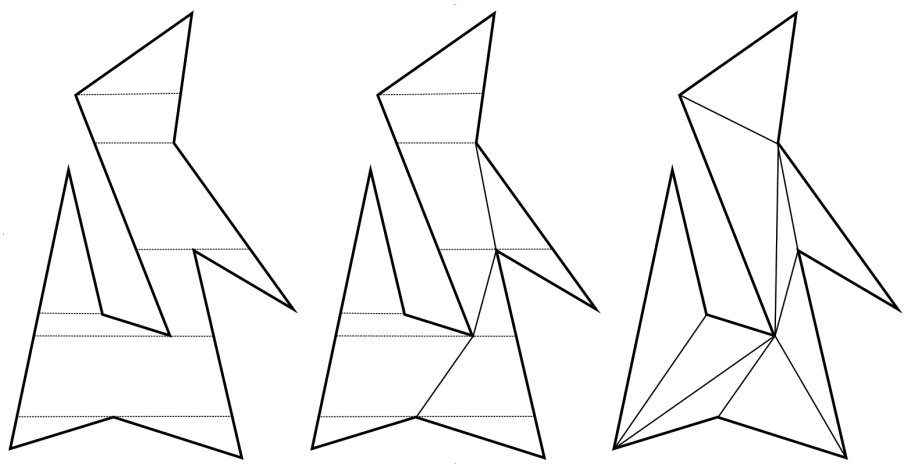


Figure 3.2: Polygon tessellation using the Incremental Randomized Triangulation algorithm: (a) descomposition of the input polygon into trapezoids; (b) descomposition of the formed trapezoids into monotone polygons (using diagonals); (c) triangulation of the monotone polygons.

1. Decomposition of the input polygon into trapezoids. The first phase consists in the analysis and decomposition of the input polygon (see Figure 3.2(a)).

   Let S be a set of non-horizontal, non-intersecting line segments of the polygon. The randomized algorithm is used to create the trapezoidal decomposition of the $X-Y$ plane arising due the segments of set S. This is done by taking a random ordering $s_1 \cdots s_N$ of the segments in S and adding one segment at a time to incrementally construct the trapezoids. This divides the polygon into trapezoids (which can degenerate into a triangle if any of the horizontal segments of the trapezoid is of zero length). The restriction that the segments be non-horizontal is necessary to limit the number of neighbors of any trapezoid. However, no generality is lost due to this assumption as it can be simulated using lexicographic ordering. That is, if two points have the same y-coordinate then the one with larger x-coordinate is considered higher. The number of trapezoids is linear in the number of segments. Seidel proves that if each permutation of $s_1 \cdots s_N$ is equally likely then trapezoid formation takes $O(n \cdot log^* n)$ expected time.

2. Decomposition of the formed trapezoids into monotone polygons. In this phase, each one of the trapezoids computed in the previous step are divided in monotone polygons, if needed (see Figure 3.2(b)).

   A monotone polygon is a polygon whose boundary consists of two y-monotone chains. These polygons are computed from the trapezoidal decomposition by checking whether the two vertices of the original polygon lie on the same side. This is a linear time operation.

3. Triangulation of the monotone polygons. Once the monotone polygons have been created the final triangulation process becomes simple and convenient. The strategy consists in drawing diagonals from vertices in the left or right boundary of the polygon in consecutive order (see Figure 3.2(c)).

   A monotone polygon can be triangulated in linear time by using a simple greedy algorithm which repeatedly cuts off the convex corners of the polygon [24]. Hence, all the monotone polygons can be triangulated in $O(n)$ time.

Results obtained with this proposal are included and analyzed in section 3.4. As will be shown in that section, good results in terms of quality are obtained following this cell-based strategy.

3.3 Implementation of the Hybrid Meshing algorithm

The second implementation of a hybrid terrain viewer uses an adapted version of the HM algorithm for a coherent and crack-free visualization of hybrid models. The HM algorithm, as it was explained in the previous chapter, was initially designed as a hardware assisted method to achieve high performance with good quality results in the interactive visualization of hybrid terrains. In this case, the additional hardware unit has been replaced by a software decoding step computed by the CPU.

In this implementation, the regular and irregular models are linked together to form a single coherent mesh in the visualization phase, using the data lists precomputed in the preprocessing. Note that this approach is basically the opposite of the previous implementation: it runs the heavier computational tasks in the preprocessing phase and encodes the results in a convenient representation, to allow a fast decoding during the visualization step.

The HM algorithm relies on two main operational cores: the local convexification of the TIN boundary for each active PC grid cell in the current LOD, and the adaptive tessellation of the hole between the convexified TIN boundary and the grid cell corners.

In the first row of Table 3.1 the computational and encoding tasks performed in the preprocessing phase are exposed. The first step is the extraction and clockwise ordering of the TIN mesh vertices placed in the boundary. The resulting polyline representing the TIN boundary is then projected on the base plane of the grid, that is, the XY plane, since the elevation measures are considered as distance values in the Z axis. The GC list is computed using the 2D footprint of the TIN boundary, tagging those grid cells containing any point of the footprint as PC cells, those entirely placed in the interior of the polyline as CC cells, and the remaining cells as NC cells, since they do not interact with the TIN mesh.

The final task in the preprocessing is performing the incremental convexification of the TIN boundary and encoding the results in the TB array and the VC list. At the beginning of the process, inter-cells vertices are added to the TIN mesh when needed, to ensure that the TIN vertices and grid edges form a delimited polygon in every PC Cell. Then, the convexification process begins from the coarsest LOD to the finest one, preserving the convexification triangles computed in previous levels. For every PC cell in the grid model some info is collected and encoded in the VC list (the enter and exit vertices TB vertices and the uncovered corners) and the newly generated convexification triangles added to the TB array.

In the visualization phase, the same general process of the previous implementation is repeated for every frame: first, the grid LOD is updated to the current viewpoint; next, the TIN mesh and the non-covered grid cells are sent to the rendering pipeline; finally, the adaptive tessellation of the hole between the meshes boundaries is efficiently rebuilt in the active PC cells from the preprocessed information. The needed triangles are generated in run-time by simple decoding operations and this will result, as it will be shown in the following sections, in good results in terms of execution time for this algorithm.

3.4 Experimental results

In this section, the results obtained testing the two different implementations of a hybrid terrain model are exposed and analyzed. The two strategies presented in Section 3.2 and Section 3.3 have been implemented and tested in the same hybrid terrain visualization application. The software can visualize interactively regular and irregular models on their own and, when

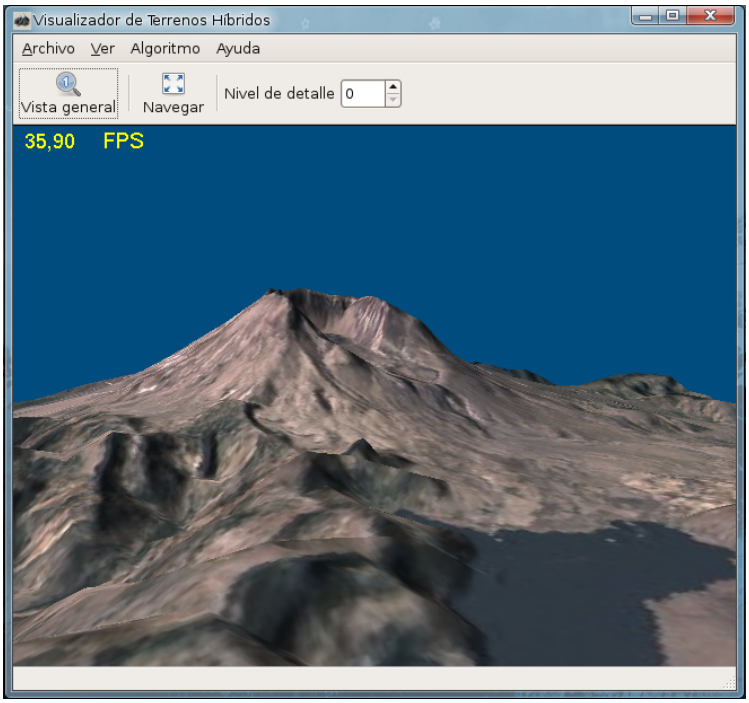
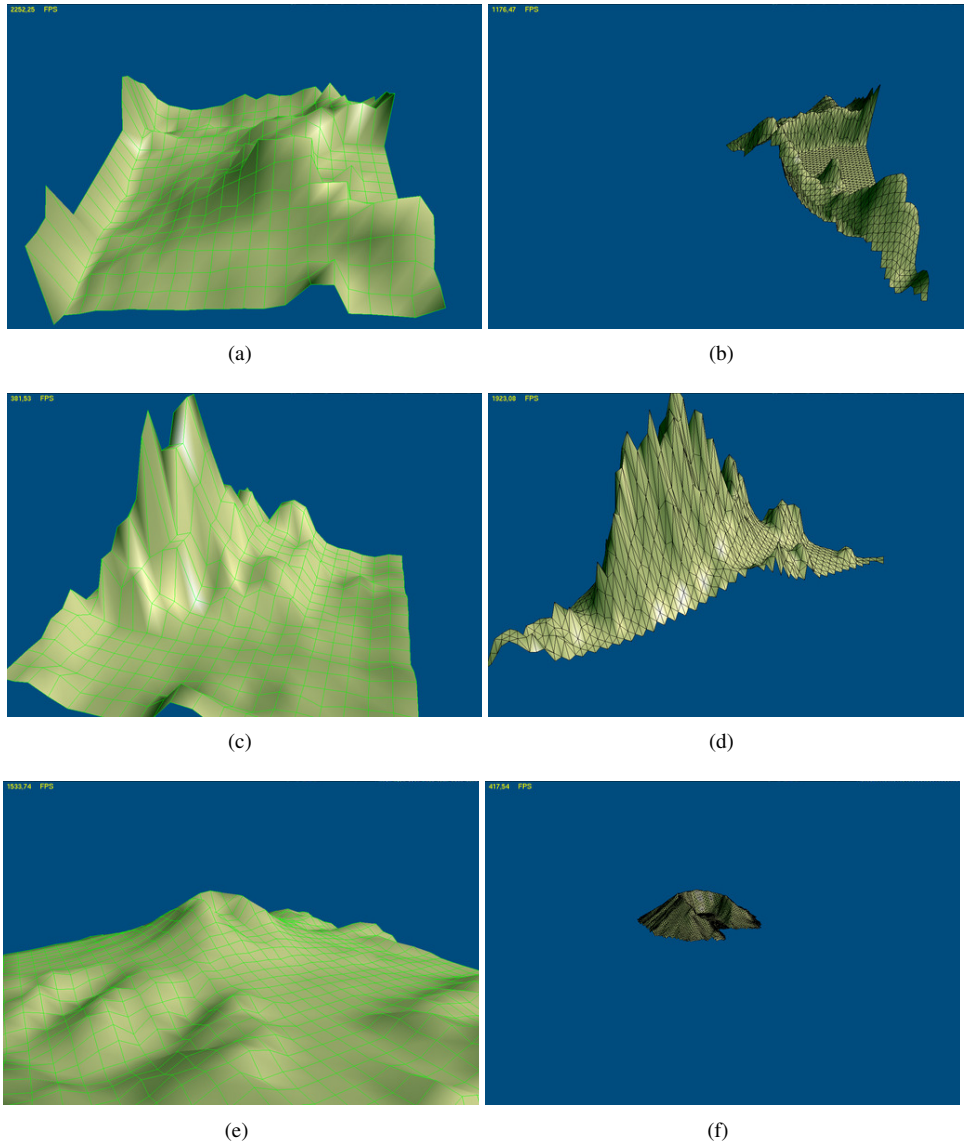


Figure 3.3: Hybrid terrain visualizer showing a textured hybrid model.

both meshes are loaded at the same time, the adaptive triangulation may be activated to render the resulting hybrid model without cracks using any of the proposed methods.

Multiresolution rendering of regular grid models is provided using a standard quad-tree approach based on the method described in [72]. The software has been designed according to the classic Model-View-Controller architectural software pattern [10], which separates the management of the model data from the user interaction. The implementation has been coded in C language, using GTK as the GUI library and OpenGL 2.0 as the graphic API. A screenshot of the application is depicted in Figure 3.3.

The hardware system used for testing has been a PC workstation with an Intel Core 2 Duo E6600 processor, 2 GB of RAM memory and a GeForce 8800GT 512 MB graphic card. Four different scenes have been used in our testing. The first three scenes, depicted in Figure 3.4, are formed by a regular grid partially covered by a TIN. The fourth scene is a synthetic model consisting of five copies of the meshes present in Scene 3 to validate the application when



(a) (b)

(c) (d)

(e) (f)

Figure 3.4: Hybrid test models: (a) scene 1 grid; (b) scene 1 TIN; (c) scene 2 grid; (d) scene 2 TIN; (e) scene 3 grid; (f) scene 3 TIN.

Table 3.2: Size and complexity of the test scenes.

	Grid cells	TIN triangles
Scene 1	400	3144
Scene 2	400	1563
Scene 3	1089	9872
Scene 4	5445	49360

Table 3.3: Results obtained with both proposals.

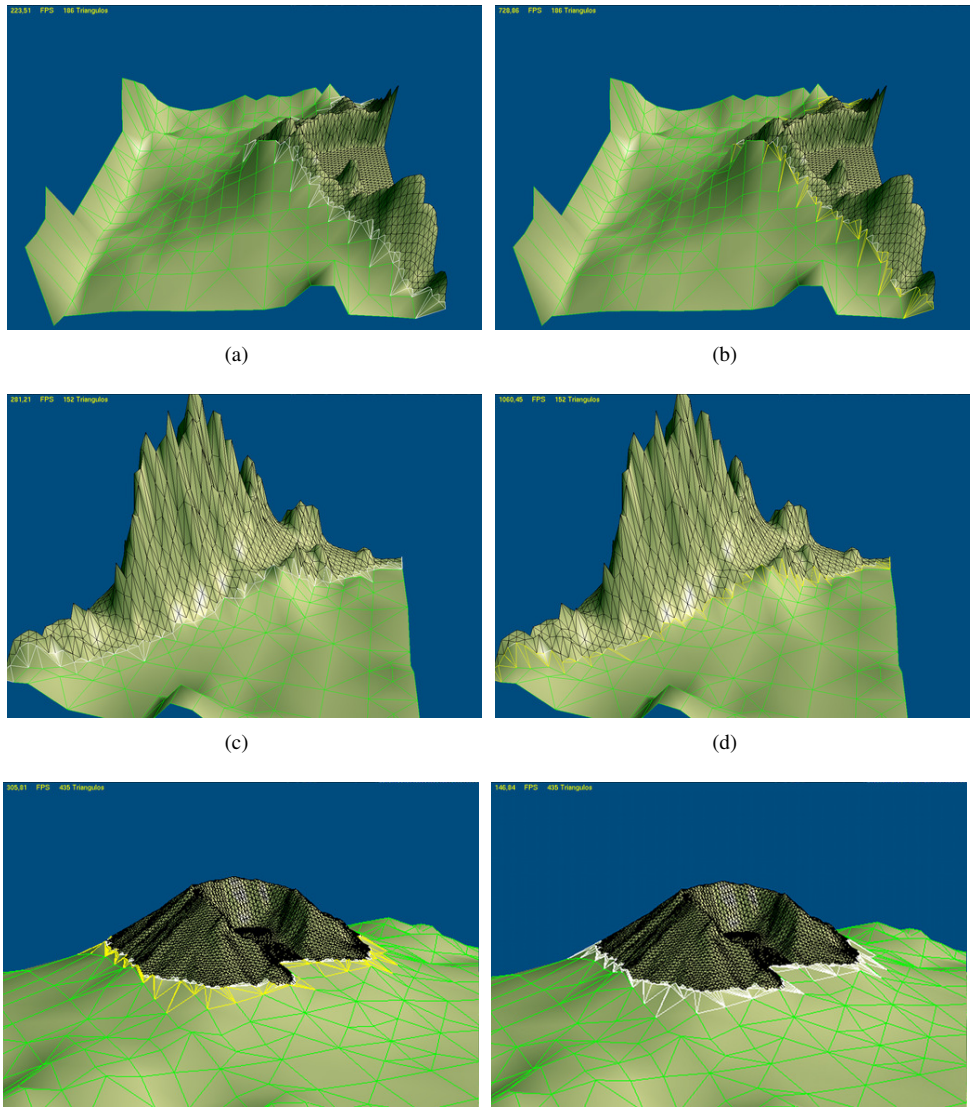
	Triangles generated	Inc. triang. based proposal	HM based proposal
Scene 1	186	442 fps	1356 fps
Scene 2	158	523 fps	1776 fps
Scene 3	444	308 fps	753 fps
Scene 4	2220	46 fps	132 fps

using several meshes at the same time. The final number of grid cells and TIN triangles of each scene is shown in Table 3.2.

The main results obtained with the two proposals are summarized in Table 3.3. The number of triangles generated to connect the different representation models (second column) is the same for the two methods. This is coherent with the fact that the polygons to be triangulated are the same no matter which method is employed. Provided that degenerated triangles[1] are not generated, the number should always be the same since every triangulation of an polygon formed by n vertices is composed by $n-2$ triangles.

As is shown in the table, the HM based proposal (fourth column) clearly outperforms the one based on Incremental Randomized Triangulation (third column). HM method is, in the worst scenery, 2.44 times faster, and it reaches the maximum difference in the *Scene 1*, where it is 3.40 times faster. This is a direct consequence of using the preprocessed connectivity information of the TIN boundary in the HM algorithm, to generate the adaptive tessellation during run-time. In this case, the triangles associated to the convexification procedure in the HM algorithm are precomputed. In run-time these triangles are extracted from the TB information and only the final triangles forming the cell corners and the convex TIN vertices have to be really computed. As the method based on the incremental randomized triangulation

[1] Triangles having two or three vertices repeated and therefore topologically similar to a line or a point



(a) (b)

(c) (d)

(e) (f)

Figure 3.5: Adaptive tessellations generated by both proposals. Subfigures (a), (c) and (d) are generated by the proposal based on the incremental randomized triangulation; subfigures (b), (d) and (f) are generated by the HM based proposal.

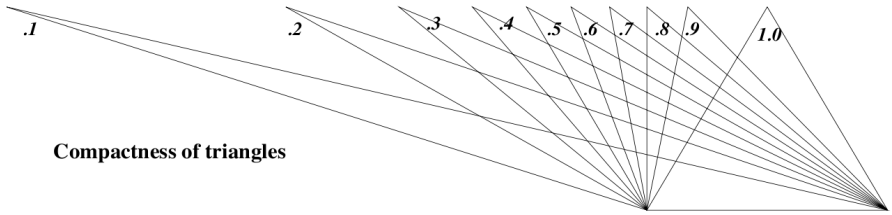


Figure 3.6: Example of triangles with associated compactness values increasing from 0.1 to 1.0 in 0.1 steps. Extracted from [28].

Table 3.4: Average compactness value of the triangles obtained with both proposals.

	Inc. triang. based proposal	HM based proposal
Scene 1	0.53	0.49
Scene 2	0.53	0.47
Scene 3	0.49	0.49

does not perform any preprocessing of the meshes, the tessellation is computed directly from the selected vertices of the boundaries for each grid cell.

With respect to the quality of the triangulation, high quality models can be generated with both methods. An example of application is shown in Figure 3.5. These are the resulting models when a detailed TIN is applied to scenes 1, 2 and 3. For both algorithms the meshes are softly joined and the connections are performed locally to each cell.

Even though the triangles generated by the two methods are different, in both cases the holes or cracks in the mesh are eliminated and the visual quality obtained is quite similar. To compare quantitatively the quality of the final triangles we use the *compactness* value [27] is an indication of the triangles shape, usually employed as a measure of quality, being zero for a degenerated triangle and one for an equilateral triangle as shown in Figure 3.6. The formal definition of the compactness c of a triangle is:

$$c = \frac{4\sqrt{3}a}{l_0^2 + l_1^2 + l_2^2} \tag{3.1}$$

where a is the positive area of the triangle and $(l_0^2 + l_1^2 + l_2^2)$ are the lengths of the three sides. Table 3.4 shows the average compactness values obtained for the two implementations. In some tests, triangles obtained with the incremental randomized triangulation method

seem slightly more compact and regular than the triangles generated with the HM algorithm, however, the results presented in this table verify that the triangulations obtained with both algorithms are very similar.

In summary, the results obtained in terms of execution times and the quality of the visualization using the HM algorithm implementation make this proposal a more suitable solution for the problem of hybrid terrain rendering. Although the proposal using Seidel's tessellation algorithm may produce marginally better quality tessellations in some specific cases, the better performance of the HM algorithm implementation allows for rendering much larger and more complex hybrid models.

3.5 Conclusions

In this chapter we have presented two different contributions to the field of the interactive visualization of hybrid terrain meshes. Both proposal are based on the local approach followed by the HM algorithm, and thus an adaptive local tessellation is generated between the boundaries of the multiresolution regular part and the irregular TIN parts.

The first proposal uses a standard polygon triangulation algorithm for the task —in this case, the Seidel's Incremental Randomized Triangulation [74]— to generate the tessellation in a straightforward way. In this simple, both models are connected using the tessellation algorithm to directly generate the triangles between the grid cell corners and the TIN boundary vertices during visualization.

On the contrary, the second proposal follows a more elaborated approach: it precomputes the most demanding tasks of the tessellation process during the preprocessing, and then uses this information to rebuild the needed triangles using only lightweight operations. This proposal also manages to implement the original HM algorithm, which used a proposed a unit hardware unit in the graphics hardware, to a software implementation using conventional hardware.

The two proposals have been implemented and tested in our own visualization software. The results of our tests indicate that both methods can obtain high quality meshes, without holes or any other triangulation artifacts. In terms of performance, however, the HM implementation is far more efficient and faster than the standard polygon triangulation strategy due to the efficiently encoded representation of the convexification information in the TIN bound-

ary, used in the HM algorithm. This means that part of the triangles are precomputed and can be decoded in run-time with very simple operations.

CHAPTER 4

GPU-HM: A PARALLEL HYBRID MESHING ALGORITHM USING THE GPU

Developing applications that make efficient use of parallel graphics hardware is a complex and time consuming task. Several factors have to be considered, such as the underlying hardware architecture, the selected programming paradigm or the specific compiler and the APIs (*Application Programming Interfaces*) available on each platform —which usually make debugging and performance tuning even harder. As a result, the development process is highly dependent on the hardware advances, especially in domains that change as fast as modern GPUs, and it is difficult to select the optimal strategy for each problem.

In this chapter the GPU-HM method is presented; to our knowledge, the first parallel implementation of a hybrid terrain model renderer. The GPU-HM algorithm achieves the parallelization of the hybrid model rendering method by adapting the original HM algorithm strategy to the Geometry Shader unit of the GPU. Therefore, the GPU-HM method exploits the GPU computational resources by combining the inherent thread parallelism of the GPU with an efficient memory access pattern to the data structures placed in device memory.

A brief introduction to the standard graphic rendering pipeline and the GPU hardware architecture is first presented. Following that, the new GPU-based implementation is explained in detail, the results obtained in the experimental tests are analyzed and, finally, the conclusions of our work are discussed. The work presented in this chapter was originally introduced in [62].

4.1 Shader-based graphic rendering

Over the past decade, graphics hardware has suffered an accelerated evolution from partially configurable implementations of a fixed-function pipeline, to highly programmable units where developers can implement specific algorithms. This flexibility has been achieved by means of programmable *Shaders*, that is programmable logical units running on the GPU. They were introduced to provide a way to replace the *fixed* functionality of the existing hardware rendering pipeline with user-generated code. The first versions of the shader-based rendering pipeline contained only vertex and fragment shaders, although additional types of shaders have been included in more recent generations of graphic hardware.

Vertex shaders enables various operations (including transformations and deformations) to be performed on each vertex. Similarly, the pixel shader processes individual pixels, allowing complex shading equations to be evaluated per pixel. Therefore, by installing custom shader programs in the GPU programmable shader units, the user can completely override the fixed implementation of core per-vertex and per-pixel behavior. Since GPUs typically have many processing units, multiple shader threads can execute in parallel potentially performing a broad list of tasks, from procedural geometry and non-photorealistic lighting to advanced textures, fog, shadows, raycasting, and visual effects.

Graphic hardware supporting Shader Model 4 or higher use an unified-shader approach. This means that the vertex, pixel, and other types of shaders share the same programming model, providing a powerful and extensible mechanism for programming the processing stages of the GPU pipeline using C-like shading languages such as High Level Shader Language (HLSL) [77], C for Graphics (Cg) [50], or OpenGL Shading Language (GLSL) [71].

HLSL is the shading language developed by Microsoft to set up the programmable 3D pipeline in DirectX 9. Before DirectX 10, the compiler generated the assembly instructions directly from the source code. However, starting from DirectX 10, an intermediary representation is generated during compilation and later translated to the native hardware instruction set of the particular GPU where the shader is executed.

Cg is a high-level GPU shader language developed by NVIDIA. The language is said to layer on top of OpenGL and DirectX since the Cg compiler may directly generate GPU assembly, or high-level GLSL and HLSL code in the different formats supported by the graphics APIs. Although the Cg Language Specification is open and may be implemented and used it by other vendors, in practice, is primarily supported by NVIDIA tools.

The OpenGL Shading Language is a high-level procedural language designed for use in OpenGL [86]. It is designed to take advantage of the capabilities of programmable graphics hardware. Using OpenGL Shading Language for shader development brings important benefits such as built-in access to existing OpenGL state, reuse of API entry points that are already familiar to application developers, and a close coupling with the existing architecture of OpenGL. Note that in this case no intermediate languages are used; thus, conforming OpenGL implementations that support the OpenGL Shading Language will compile and execute properly the source code provided by the application at runtime.

Although some minor differences still exist in performance and functionality of the shader programs depending on the shader language used, all of them provide a flexible approach to leverage the power of the very capable modern graphic hardware. A more important aspect, related to the visualization of hybrid terrain models using a shader-based rendering approach, is to decide the method for the geometry generation, since current shading models offer different possibilities. Actually, this is the most critical aspect of the implementation since using one pipeline stage or another to create the adaptive tessellation between the boundaries of the models, implies using completely different strategies.

In the following subsections, the two shader units which currently have the capability of generate new geometry in the pipeline —the Geometry Shader and the Tessellator Unit— are reviewed in detail. Note that these units belong to different hardware generations and, since the Geometry Shader was developed first, it is supported by a broader range of GPU hardware.

4.1.1 Geometry generation in the GPU using the Geometry Shader

The Geometry Shader is a functional stage of the rendering pipeline introduced with Direct3D 10 [7] which habilitates the generation and destruction of geometric data at primitive level on the graphics processor. In OpenGL, the Geometry Shader was first introduced in the form of an extension [53] and then made part of the OpenGL core specification in the OpenGL 3.2 revision.

Figure 4.1 depicts a functional view of the rendering pipeline in Direct3D 10. The tasks performed of the programmable stages, represented as rounded corner boxes, are defined by the developer using shader programs. The fixed stages moves data output from one programmable stage to the the next one performing the necessary interpolations and adaptations between them. The Geometry Shader stage is placed just after the primitive assembly stage and before the rasterizer. The Stream Output stage is a new functionality in Direct3D 10

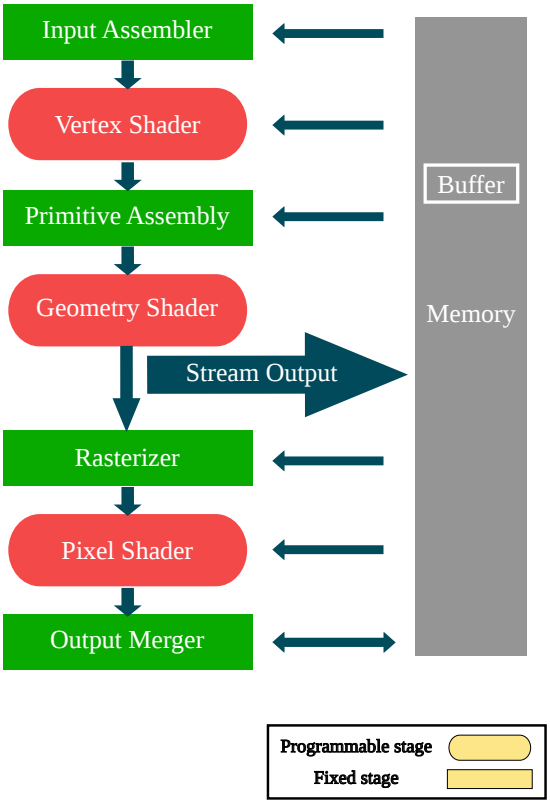


Figure 4.1: Direct3D 10 pipeline functional representation from [65].

located just before the rasterization stage, whose purpose is to stream vertex data from the geometry-shader stage to buffers in GPU memory. Data streamed out to memory can be read back into the pipeline in a subsequent rendering pass, or can be copied to CPU main memory.

The Geometry Shader stage is a new programmable stage that is executed in the middle of the two original programmable stages: the vertex and fragment shader. In contrast to the other shader types that can only access information about the input element —and thus are strictly one-in, one-out programs— the Geometry Shader runs once per input primitive and has access to all the vertex data for the vertices that form the input primitive being processed.

A Geometry Shader program can also produce additional elements, i.e., it is able to add or delete some elements in the geometry stream sent to the rendering pipeline in a programmatic

way, by using explicit functions (*EmitVertex()* and *EndPrimitive()*) to produce vertices and primitives (this is called *amplification*). The vertex shader cannot generate new vertices nor it can stop the vertex from being processed further in the pipeline. Similarly, the fragment shader processes a single fragment at a time, cannot create new fragments, and can only destroy fragments by discarding them. In DirectX 11 hardware, Tessellation Shaders are also able to increase or decrease the amount of work in the pipeline, but only implicitly by setting the tessellation level for the patch (see Section 4.1.2).

Note also that geometry shaders, unlike vertex and fragment shaders, are an optional part of the rendering pipeline. When no geometry shader is present, the rendering pipeline operates normally using the outputs from the vertex shader, interpolated across the primitive being rendered, as inputs to the fragment shader. However, if a geometry shader is present between vertex and fragment shader, the outputs of the vertex shader become the inputs to the geometry shader, and the outputs of the geometry shader are what are interpolated and fed to the fragment shader.

The Geometry Shader accepts different primitives as inputs and produce a list of primitives as outputs, which can be of a different type and with a different number of vertices of the input one. Furthermore, geometry shader can be used with a special kind of input primitives containing vertices that are not part of the ordinary primitives, but neighboring vertices that are adjacent to the two line segment end points or the three triangle edges. These vertices can be accessed by geometry shaders and used to match up the vertices emitted by the geometry shader with those of neighboring primitives.

Thus, the Geometry Shader can discard the input geometry or insert new primitives into it. Adjacency information and texture data are also available from within the Geometry Shader, so that for each triangle the information of neighboring triangles can be accessed.

4.1.2 Geometry generation using the Tessellator Unit

Direct3D 11 [12], the renewed version of the Microsoft's graphics API, provided access to important new features of modern graphics hardware. One of those features is support for an extended pipeline including several tessellation stages performed in the GPU. Figure 4.2 depicts in light blue color the new pipeline stages: the Hull Shader, the Tessellator Unit and the Domain Shader. Note that the Tessellator unit is represented as a fixed stage because it is not directly programmed by the developer, although the developer can control the tessellation

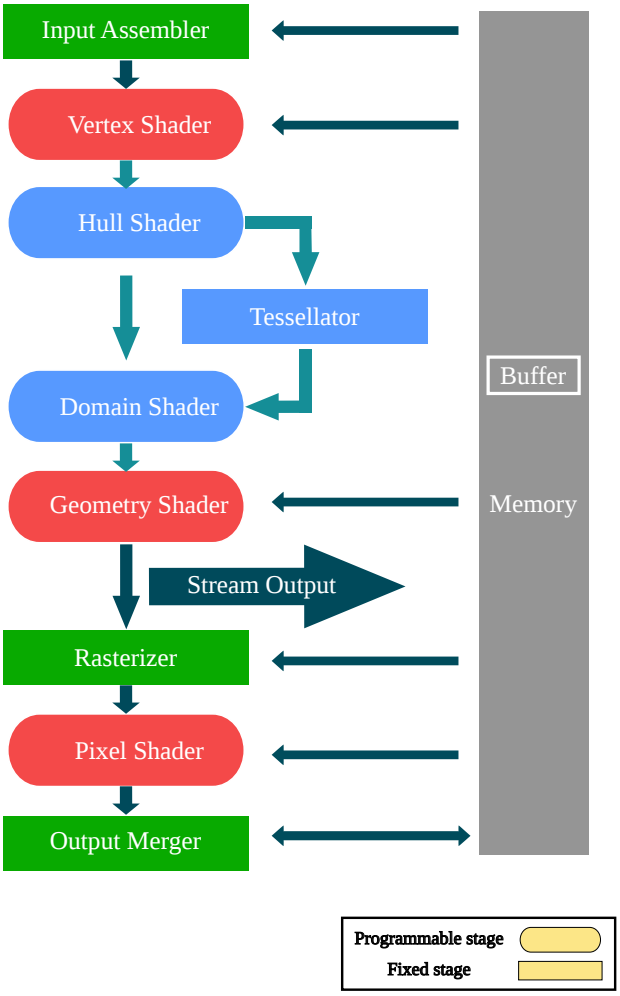


Figure 4.2: Direct3D 11 pipeline representation.

pattern through parameters. The main goal of these new pipeline stages is to enable efficient rendering of higher order surfaces, such as approximations to subdivision surfaces.

The new stages stand between the Vertex Shader and the Geometry Shader. As their name implies, the Hull Shader and Domain Shader are programmable stages, while the Tessellator Unit, although fairly flexible and configurable, is a fixed function stage. Direct3D 11 also introduces a new primitive type for using when the tessellation stages are enabled: the *patch*. Actually, patches are the only primitive type that is supported in the tessellation pipeline. This new primitive type have an arbitrary number of vertices between 1 and 32 and do not have any implied topology; a patch is just a disconnected set of vertices. This means, for instance, that a patch with three vertices is not necessarily a triangle. The interpretation of the patch vertices corresponds to the programmer writing the shaders.

In the tessellation pipeline the Vertex Shader is still the first programmable stage, basically used to transform vertices from object to world space. After the Vertex Shader, the Hull Shader is invoked for each patch with all its transformed vertices. The Hull Shader can perform per primitive operations and has a fixed output that has to be declared in advance. The Hull Shader serves for computing per edge tessellation factors later provided to the Tessellator Unit. Additionally, the Hull Shader may also perform computations that are invariant for all the vertices that will be generated in the Domain Shader as, for example, transform the input vertices from one basis to another, in case of the input representation is not useful for direct evaluation.

The following stage is the Tessellator, which is the component were the data expansion is actually performed. The Tessellator generates a semi-regular tessellation pattern for the set of edges and interior tessellation factors computed in the Hull Shader. The actual pattern used can be partially configured by the developer. The generated tessellation patterns are symmetric along the edges and both the triangle and quad domains are supported. The tessellator can generate triangles in clockwise or counter-clockwise order.

Finally, the Domain Shader takes the parametric coordinates of the vertices generated by the Tessellator and the control points output by the Hull Shader and uses them to evaluate the surface. In this stage, one thread is created for each generated vertex, in order to parallelize the evaluation the surface. Furthermore, it is necessary to use a surface representation that is amenable for direct evaluation. That is, given a parametric coordinates, it should be possible to evaluate at least the position and normal of the surface at that location. In addition to position and normal, texture coordinates and sample textures can also be interpolated by the

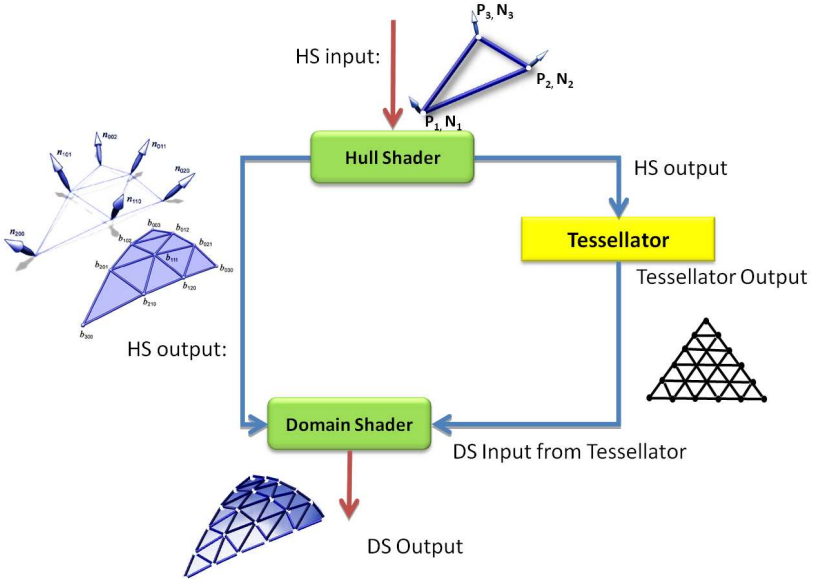
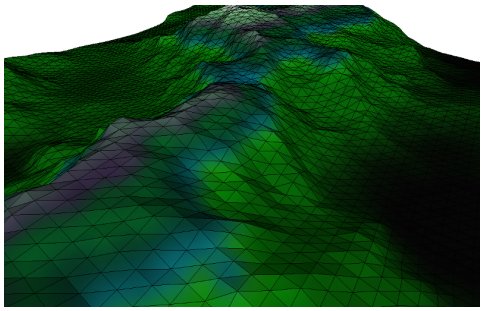


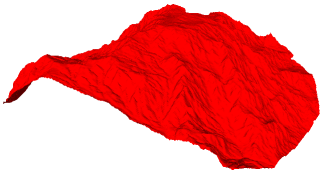
Figure 4.3: Direct3D 11 hardware tessellation example using PN Triangles from [52].

Domain Shader to apply displacement maps. Finally, the Domain Shader also runs other tasks traditionally performed by the Vertex Shader, as projecting the vertex position to screen, or transforming normals and light vectors to the same space. Once the vertices have been transformed by the Domain Shader, the primitives generated by the Tessellator may be optionally processed by the Geometry Shader or directly sent to the triangle setup stage for rasterization.

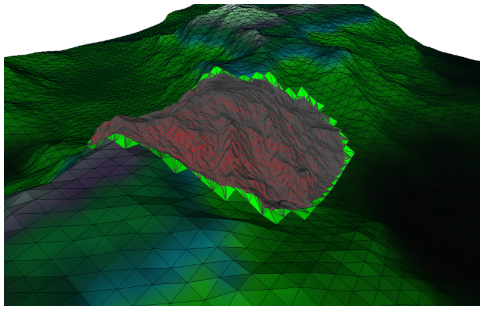
An example of this process is depicted in Figure 4.3. In this case, the Tessellator stages perform a triangle tessellation using PN (Point-Normal) Triangles and Bézier surface to smooth a low-resolution mesh. The Hull Shader computes the interpolation control points for the geometry and normals of the the input triangle while the Tessellator stage effectively subdivides the input triangle and creates new vertices. The final smoothed geometry patch is generated by the Domain Shader by using the control points to alter the attributes of the vertices in the subdivided geometry patch.



(a)



(b)



(c)

Figure 4.4: Example of hybrid terrain model obtained with our GPU method: (a) base regular grid; (b) detailed TIN component mesh; (c) resulting hybrid model with different components highlighted.

4.2 GPU-HM architecture

Several parallelization strategies were considered for the development of this method. The hardware tessellation pipeline described in the previous section seemed at first a promising and powerful approach. The Shader Model 5 hardware tessellation pipeline described in the previous section seemed a promising and powerful approach but it was discarded due to several incompatibilities of this approach with the requirements of an adaptive tessellation for hybrid terrain models. The most important drawback is that the Tessellator unit does not support a free adaptive tessellation approach, which is key for the local adaptive tessellation of the PC grid cells. Other noticeable limitations for our purposes are that the hardware tessella-

tion approach requires primitives supporting arbitrary parametric evaluation, which is hard to achieve in this case, and that the Domain Shader operates on a single parametric location, preventing the use of incremental schemes and therefore to reuse local computations and memory accesses.

A Geometry Shader based approach was the best option to parallelize the HM algorithm. In the GPU-HM method, the Geometry Shader is used to decode the precomputed data structures and to generate the triangles needed for the tessellation of each PC cell without the severe limitations of using a regular Tessellator pattern. Both the local TIN boundary convexification and the corner triangles generation steps can be efficiently implemented by using the Geometry Shader and exploiting the local coherence of the process at cell level, which can not be easily leveraged in any of the new DirectX 11 pipeline steps. The GPU parallelism is automatically exploited as several cell tessellations are executed at the same time in multicore GPUs. To obtain better performance during rendering, an optimized version of the original data structures is used to improve the memory access.

A completely different approach for the parallelization of the HM algorithm is to use GPGPUs (General-Purpose Computation on Graphics Processing Units) frameworks such as NVIDIA CUDA [36] or OpenCL [78], a standard programming language for heterogeneous computing. A GPGPU strategy was also considered, since it has become a relevant and very common solution for the parallelization of some tasks —mainly sustained by the increasing computational power, good scalability and comparatively lower cost of GPU hardware. The specialized hardware design of modern GPUs can perform much faster than a normal CPU.

Although CUDA or OpenCL kernels should be as efficient as shading programs and are arguably more flexible for general parallel computations -allowing some synchronization primitives and shared memory between different threads- shading languages tend to be more efficient for graphics processing applications since GPUs are originally designed for graphics. For example, when using shading languages most of the boilerplate rendering code is already implemented in the fixed steps of the graphic pipeline and the number of running shader instances is automatically selected based on the screen resolution. Moreover, some graphics specific functionalities such as Mipmapping, fast evaluation of mathematical functions or fast data down-sampling and up-sampling are only available when using shading programs.

The next subsections present a detailed description of our proposal. The optimized data structures are presented first and then the tessellation and rendering phase of the algorithm are described.

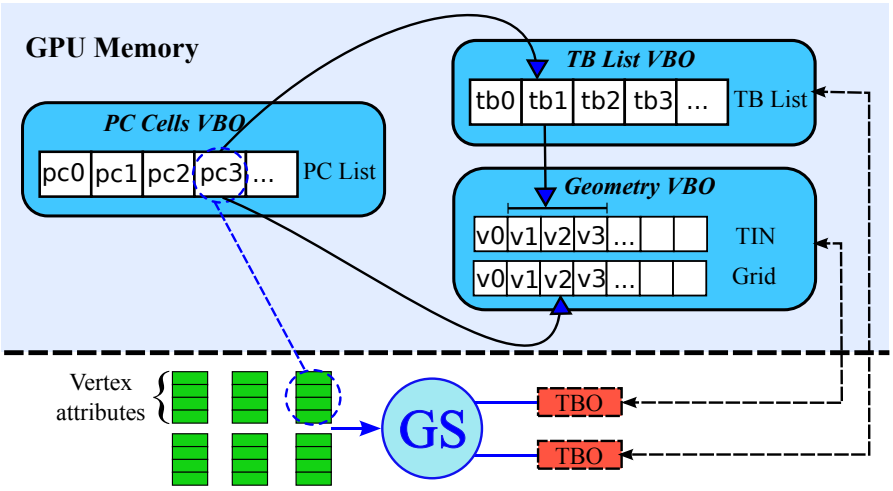


Figure 4.5: Organization of data structures in memory.

4.3 GPU and CPU data structures

The main data structures used by the original HM algorithm were presented in Chapter 2: the GC list, the list of PC cells and the TB list. Since in the GPU-HM proposal the rendering phase of the HM algorithm is performed by the Geometry Shader, every shader thread needs to access some of these structures, as well as the meshes' geometric data. However, the original HM data layout can not be efficiently accessed from the GPU. Thus, in our proposal, the data structures are stored and accessed in different ways, depending upon their accessing pattern. A visual diagram showing the location and access methods used for the data structures is depicted in Figure 4.5.

The GC list is only needed to identify the covered, uncovered and partially covered cells during the rendering of the grid mesh. Once one grid cell has been identified, it is rendered, discarded or tessellated using the GPU. Thus, the GC list is not used by the Geometry Shader (labeled as GS in the figure) during the PC cells tessellation.

During the PC cell tessellation stage the HM rendering algorithm uses the geometric data of the TB vertices and the grid cell being tessellated, together with the TB data and the list of PC cells. Consequently, these data structures have to be maintained in GPU memory. They are all stored in different Vertex Buffer Objects (VBO), but they are accessed by the shader in different ways. The geometry data and the TB list need an array-like access, since the

processing of one PC cell may involve reading non adjacent elements in random order. Hence, in our implementation, Texture Buffer Objects (TBO), associated to the corresponding VBOs, are used to read this data within the shader.

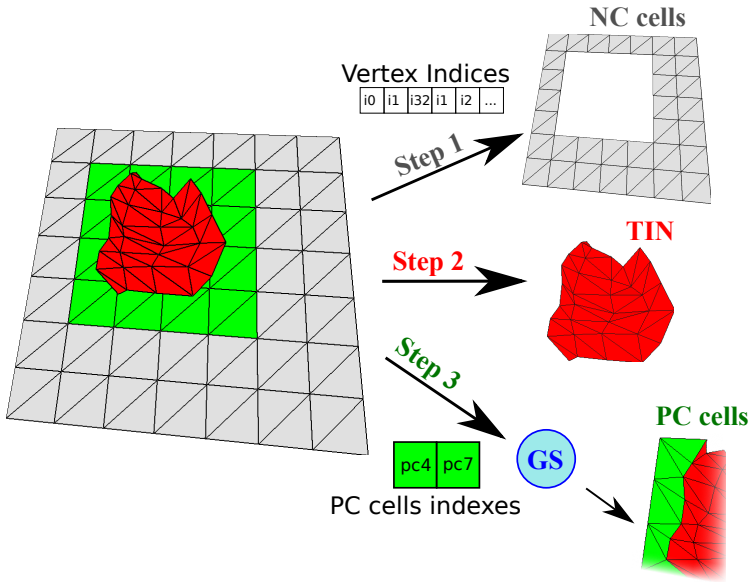
Texture Buffer Objects [54] are a common and efficient way to obtain array-like access to large data buffers residing in the GPU memory. They provide a convenient interface to the data, simulating a 1D texture where the tex coordinate corresponds to the offset in the buffer object. Furthermore, accessing latencies can be effectively hidden by overlapping the readings with data processing operations, as in the HM rendering stage there are several processing operations.

An additional optimization attaining superior data locality in accesses to the TB list has been implemented in our proposal. The TIN mesh vertices have been reordered in the vertex buffer, to guarantee that the TB boundary vertices are stored at the beginning of the vertex buffer containing the geometry data. In this way, the offset of the vertex data in the buffer object represents its position in the TB list. Thus, vertex information can be accessed by only using its TB index, reducing one level of indirection regarding to the original data structures. Another relevant consequence of this optimization is that adjacent vertices in the TB are now also adjacent in the buffer object, improving the cache behavior due to a better data locality.

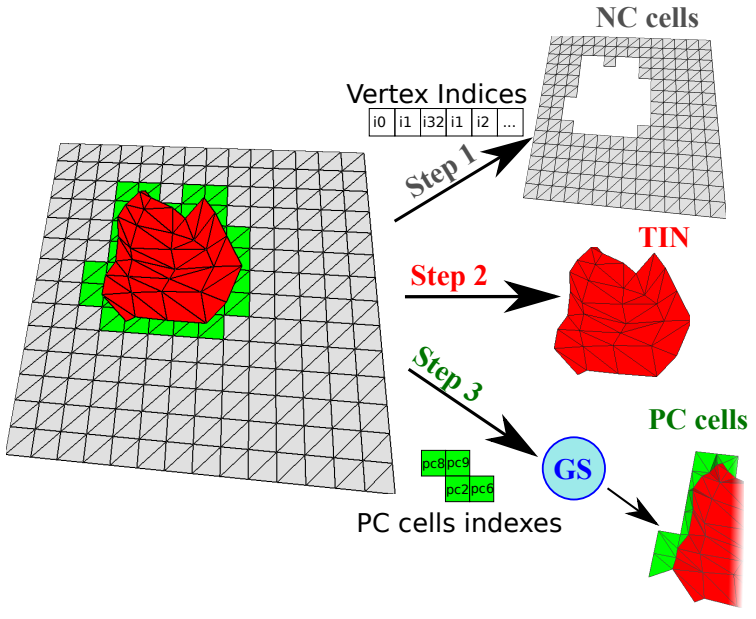
The PC list, on the other hand, is made available to the shader through normal input vertex attributes. There is a one-to-one relationship between PC cells and vertex shader threads; therefore, packing the items in the PC list as input vertex attributes is the most effective way to send the right information for every shader thread. Moreover, data transfer between CPU and GPU during rendering is reduced, given that we can use indexed draw calls to indicate the active PC cells being tessellated every frame, as the data are already stored in the GPU memory. This list of active PC cells is easily constructed on-the-fly by the CPU, according to the view-dependent grid LOD, and its small size allows a efficient communication with the GPU.

4.4 Rendering algorithm

Our GPU-HM proposal consists of two main phases: preprocessing and rendering. In the preprocessing phase, the auxiliary data structures are computed and encoded following the steps described in Section 4.3. Then, during the rendering phase, these data structures are used to guide the visualization process.



(a)



(b)

Figure 4.6: HM rendering algorithm steps: (a) coarse LOD render; (b) fine LOD render.

The main steps of the HM rendering phase are similar to those exposed in Chapter 2. An overview of the process for a sample terrain is represented in Figure 4.6. The visualization of a coarse grid LOD is shown in Figure 4.6(a) while a finer LOD is used in Figure 4.6(b). Note that the TIN mesh is the same in both cases, but the PC grid cells depend on the grid LOD and thus different cells are rendered. In those figures, the input grid and TIN meshes are rendered following the order indicated by the numerical labels. The additional parameters needed are represented as inputs for each step.

The process begins by identifying the *active* NC, CC, and PC cells, i.e., the cells belonging to the selected grid LOD in the current frame. Once this is done, the NC cells are rendered by indexed draw calls (Step 1 in Figures 4.6(a) and 4.6(b)), as the vertices data are already stored in the GPU memory. Then, CC cells are discarded and replaced by the whole TIN mesh (Step 2 in the same figures). This operation is lightweight since TIN meshes are static, usually cover around 20-30% of the grid extension and the geometry data are also stored in the GPU memory, which makes the rendering very efficient.

Finally, the Geometry Shader which performs the PC cells adaptive tessellation is loaded and the list of active PC cells are sent to the GPU, where the actual tessellation is computed (Step 3 in the same figures). The input of each shader thread is the tessellation data of the PC cell that is going to be processed: every data field encoded in a different vertex attribute. By the time the Geometry Shader finishes the processing, all the convexification and corner triangles have been generated, and therefore the PC cell is tessellated.

During the rendering, each one of the several Geometry Shader threads processes the TB vertices in the PC cell sequentially. Listing 4.1 shows a fragment of the GPU-HM geometry shader main function and Listing 4.2 reproduces relevant parts of the triangle generation code.

Vertex connectivity values greater than one mark the beginning of a cave in the local convex hull (line 34 in the Listing 4.1). Then, the associated convexification triangles are reconstructed and emitted by the Geometry Shader, according to the original tessellation algorithm. This is performed in the shader function *generateTbTriangle()* (line 21 in the Listing 4.2). The coordinates of the convexification triangle points are directly read from a Vertex Buffer in the GPU memory using a Texture Buffer Object (*texelFetchBuffer()* commands at lines 22, 26 and 29) and, after applying the correspondant *ModelView* transformation matrix, the triangle is emitted by the shader (line 33). Note that the $VDATA\_ITEM\_OFFSET$ is a simple constant value encoding the size of every TB data item in memory.

Listing 4.1: Fragments of the main function in the *hmshader* source

```
void main() {
   ... // Initialize temp data

   int k = 0;
   int starti = int(alciData.x); // START_TB
   int i = starti;
   int top = i + int(alciData.y); // COUNT_TB
   int lastHullvIdx = 0;

   for( ; i < top; i++) {
      int vIdx = int(mod(i, tbSize));
      int conn;
      readTbInfo(vIdx, conn);

      // Close caves
      while ((k > 0) && (i >= startVertexLast[ k - 1 ]))
         k--;

      if ((k > 0) && (i + conn < top)) {
         // Inner point. Tessellate cave if it is inside the cell
         int vIdx2 = int(mod(i + conn, tbSize));
         int conn2;
         readTbInfo(vIdx2, conn2, vIdx2);
         generateTbTriangle(startVertexIdx[ k - 1 ], vIdx, vIdx2);
      }
      else {
         // Point in convex hull. Join to grid corner
         if (i > starti)
            generateLink(lastHullvIdx, vIdx);
         lastHullvIdx = vIdx;
      }

      // Begin a new cave
      if (conn > 1) {
         startVertex[k] = i;
         startVertexIdx[k] = vIdx;
         startVertexLast[k] = i + conn;
         k++;
      }
   }

   vec3 vActiveCorner = readActiveCorner();
   vec3 vLastHull = (texelFetchBuffer(vArrayTex, vOffsetTin +lastHullvIdx
        * VDATA_ITEM_OFFSET)).xyz;
   while (activeCorner != topCorner) {
      vec3 vPreviousCorner = vActiveCorner;
      activeCorner = int(mod(activeCorner + 1, 4));
      vActiveCorner = readActiveCorner();
      emitTriangle(vPreviousCorner, vLastHull, vActiveCorner);
   }
}
```

Listing 4.2: Fragments of the triangle generation functions in the *hmshader* source

```
1  void generateLinkTriangles(in int lastHullvIdx, in int vIdx) {
2     vec3 vActiveCorner = readActiveCorner();
3     vec3 vLastHull = (texelFetchBuffer(vArrayTex, vOffsetTin +lastHullvIdx
           * VDATA_ITEM_OFFSET)).xyz;
4     vec3 vActiveHull = (texelFetchBuffer(vArrayTex, vOffsetTin +vIdx *
          VDATA_ITEM_OFFSET)).xyz;
5
6     // Check if corner shit is needed
7     float cpz = (cross(vLastHull - vActiveCorner, vActiveHull -
          vActiveCorner)).z;
8     while ( (sign(cpz) <= 0.0) && (activeCorner != topCorner) ) {
9        // Generate 'shift' triangle
10       vec3 vPreviousCorner = vActiveCorner;
11       activeCorner = int(mod(activeCorner + 1, 4));
12       vActiveCorner = readActiveCorner();
13       emitTriangle(vPreviousCorner, vLastHull, vActiveCorner);
14       cpz = (cross(vLastHull - vActiveCorner,
15                    vActiveHull - vActiveCorner)).z;
16    }
17
18    emitTriangle(vLastHull, vActiveHull, vActiveCorner);
19 }
20
21 void generateTbTriangle(in int idx0, in int idx1, in int idx2) {
22    vec4 pos = texelFetchBuffer(vArrayTex, vOffsetTin +idx0 *
          VDATA_ITEM_OFFSET);
23    vec4 origPos = pos;
24    gl_Position = gl_ModelViewProjectionMatrix * pos;
25    EmitVertex();
26    pos = texelFetchBuffer(vArrayTex, vOffsetTin + idx1 *
          VDATA_ITEM_OFFSET);
27    gl_Position = gl_ModelViewProjectionMatrix * pos;
28    EmitVertex();
29    pos = texelFetchBuffer(vArrayTex,
30                           vOffsetTin + idx2 * VDATA_ITEM_OFFSET);
31    gl_Position = gl_ModelViewProjectionMatrix * pos;
32    EmitVertex();
33    EndPrimitive();
34 }
```

Vertices belonging to the local convex hull of the TIN boundary are also connected to the grid cell vertices, by generating additional corner triangles in the shader function *generateLinkTriangles()* (line 1 in the Listing 4.2). In this case, the TIN points are read from the Vertex Buffer using the same *texelFetchBuffer()* command (lines 3 and 4) and then the grid vertex coordinates are read from the buffer with this same mechanism at line 12. Since the index of the grid point needs to be calculated from its relative position, the *texelFetchBuffer()* command is encapsulated in the *readActiveCorner()* function which performs both operations at the same time.

In our proposal, the sequential tessellation operations have been reordered and optimized for GPU execution. Several optimization techniques have been applied, resulting in an implementation of the algorithm with fewer conditional branch statements, only one iteration over the TB vertices (the main loop at line 10 in the Listing 4.1) and a reduced number of memory reading operations. Other optimization techniques have been additionally used when possible, such as loop unrolling and transformation of complex conditional expressions into arithmetic expressions or several simpler expressions for faster evaluation. Another relevant performance improvement derives from the reordering of the TIN vertices in the vertex buffer. Due to this optimization, the data of a TB vertex is accessed with the same index in the TB list and in the vertex buffer, eliminating the pointer to the geometry data position used in the original TB list (lines 11 and 21 in the Listing 4.1). With this optimization, not only are data reading operation reduced to almost one half of the original number, but also many read operations in the same thread could benefit from data locality and attain much better cache behavior.

The implementation strategy presented in [9] shows that the decoding and generation of the incremental convexification triangles, and the generation of corner triangles are performed in two stages. The GPU-HM method succeeds in merging those different stages together, reducing the overall complexity of the source code and the number of conditional evaluations of our implementation.

4.5 Experimental results

The results of our tests with the GPU implementation of the HM algorithm are shown in this section. The software application used for testing has been coded in C++ language and GLSL in the shaders, since OpenGL is the hardware acceleration API. The results were collected in

Model	Grid cells	TIN Δ	TB elems.
Coruña	998K	1739K	17K
Sil	998K	657K	14K
Pmouros	998K	656K	16K

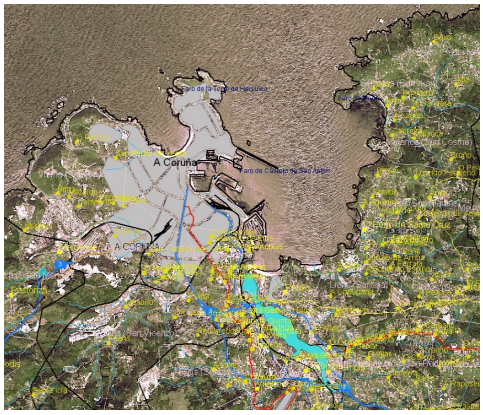
Table 4.1: Size of the test models.

a Ubuntu Linux system with an Intel Core2 Quad 2.6 GHz processor. Two different Nvidia GeForce graphic cards, an GTX480 and a GTX280, were used in the tests. The Geforce GTX 480 video card is equipped with a GPU with 480 cores running at a maximum clock speed of 700 MHz and 1.5 GB of GDDR5 video memory. It offers support for DirectX 11 and OpenGL 4. On the other hand, the GTX280 has 240 cores running at 600 MHz and 1 GB of GDDR3 memory.

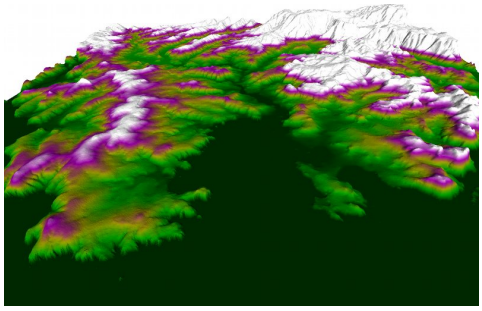
For the tests, three sample models of locations in the region of Galicia were generated from freely available data in the Spanish GIS database (IDEE) [34]: *Coruña*, *Sil* and *Pmouros*. The *Coruña* model shown in Figure 4.7 represents the Coruña bay area: an aerial view of the area is depicted in Subfigure 4.7(a), a rendered image of the grid model in Subfigure 4.7(b), the TIN model in Subfigure 4.7(c) and the obtained hybrid model in Subfigure 4.7(d). The *Sil* model (see Figure 4.8) corresponds to part of the canyon formed around the Sil river; as in the previous figure an aerial view of the area is depicted in Subfigure 4.8(a), a rendered image of the grid model in Subfigure 4.8(b), the TIN model in Subfigure 4.8(c) and the obtained hybrid model in Subfigure 4.8(d). The *Pmouros* model (see Figure 4.9) represents the area around the Portodemouros dam. Again, an aerial view of the area is depicted in Subfigure 4.9(a), a rendered image of the grid model in Subfigure 4.9(b), the TIN model in Subfigure 4.9(c) and the obtained hybrid model in Subfigure 4.9(d).

The global sizes of the models are summarized in Table 4.1. The first column contains the number of cells in the maximum resolution version of the base grids, the second one shows the number of triangles forming the TIN meshes and the final one represents the number of vertices in the TB.

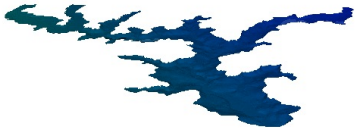
In Table 4.2, the number of PC, CC and NC cells are shown for each grid LOD from the coarsest level ($L0$) to the finest one ($L3$). The ratio of each one of this cell types over the total number of cells is also indicated in parentheses. Since the cells are smaller at finer levels of detail, it may be seen that the ratio of PC cells decreases while, at the same time, the ratio of NC and CC cells raises.



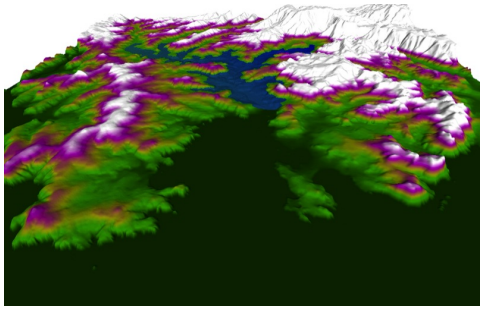
(a)



(b)

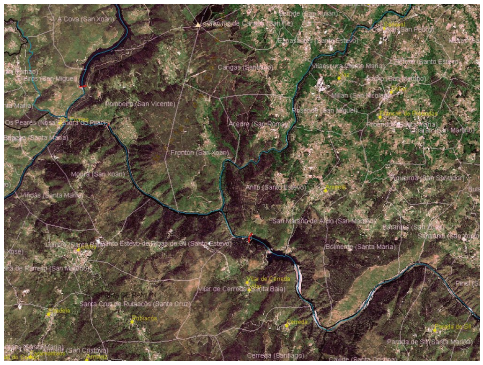


(c)

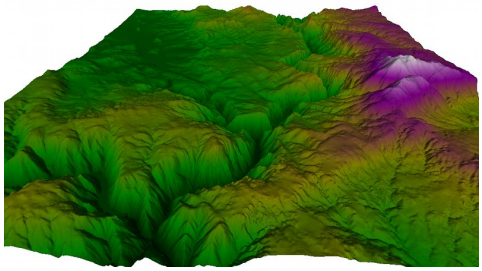


(d)

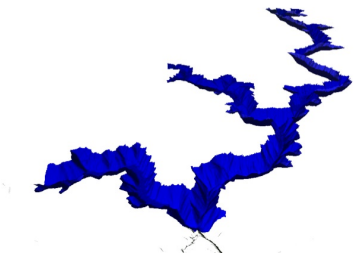
Figure 4.7: *Coruña* model: (a) aerial view of the area from the IDEE [34]; (b) grid model; (c) TIN model; (d) hybrid model, the TIN mesh is highlighted in blue.



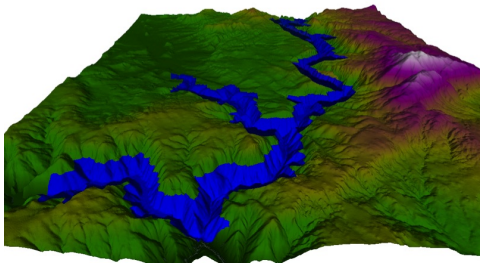
(a)



(b)

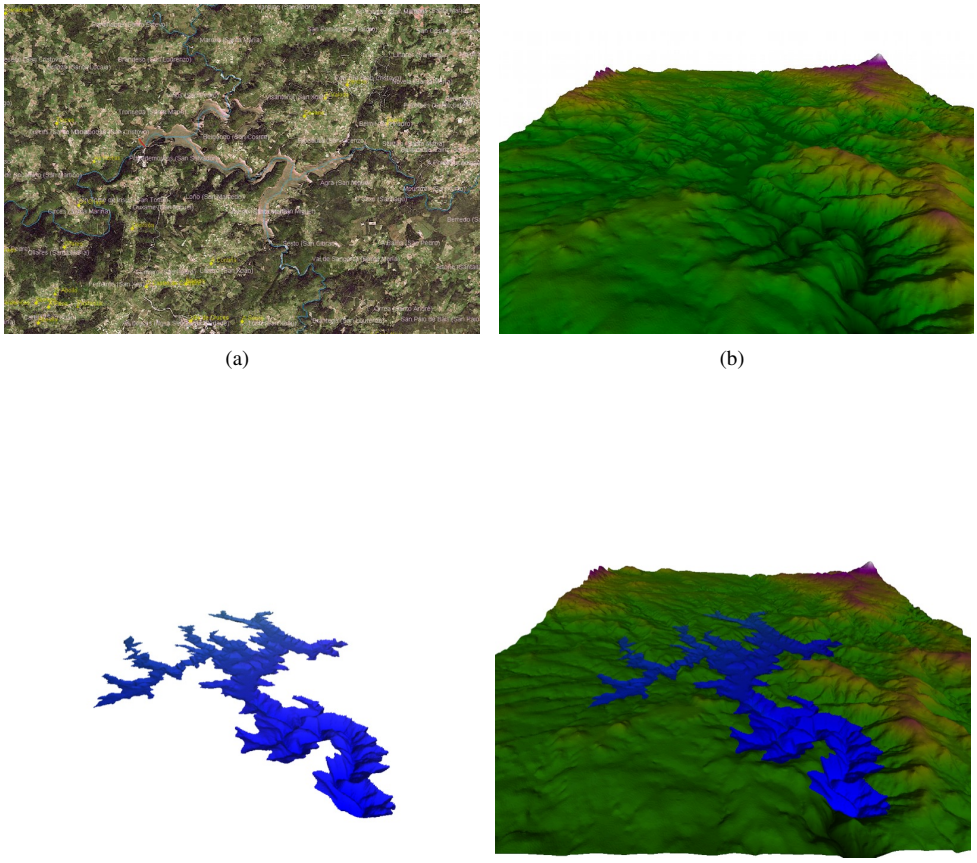


(c)



(d)

Figure 4.8: *Sil* model: (a) aerial view of the area from the IDEE [34]; (b) grid model; (c) TIN model; (d) hybrid model, the TIN mesh is highlighted in blue.



(a) (b)

(c) (d)

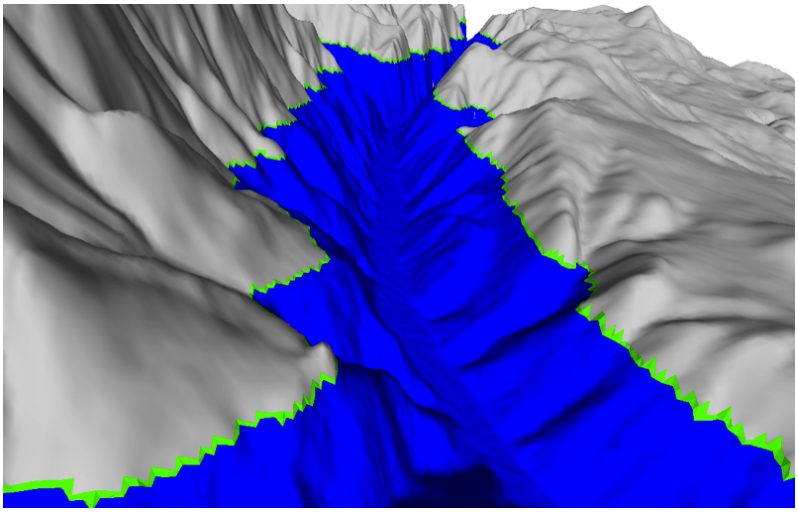
Figure 4.9: *Pmouros* model: (a) aerial view of the area from the IDEE [34]; (b) grid model; (c) TIN model; (d) hybrid model, the TIN mesh is highlighted in blue.

Coruña				
	L0	L1	L2	L3
PC cells	517 (3.36%)	1058 (1.71%)	2142 (0.86%)	4233 (0.43%)
NC cells	14418 (93.77%)	58606 (94.52%)	236079 (94.81%)	949668 (95.16%)
CC cells	441 (2.87%)	2337 (3.76%)	10780 (4.33%)	44100 (4.42%)
Rendered Δ	1787K	1876K	2231K	3661K
PC cells Δ	19K (1,06%)	19K (1,01%)	20K (0,89%)	22K (0,60%)
Sil				
	L0	L1	L2	L3
PC cells	857 (5.57%)	1775 (2.86%)	3510 (1.41%)	7630 (0.76%)
NC cells	13428 (87.33%)	54666 (88.17%)	217694 (87.43%)	904681 (90.65%)
CC cells	1091 (7.10%)	5560 (8.97%)	27797 (11.16%)	85690 (7.10%)
Rendered Δ	701K	784K	1111K	2489K
PC cells Δ	17K (2,38%)	17K (2,17%)	19K (1,66%)	23K (0,90%)
	Pmouros			
	L0	L1	L2	L3
PC cells	821 (5.34%)	1697 (2.74%)	3329(1.34%)	7912 (0.79%)
NC cells	13640 (88.71%)	55752 (89.92%)	225396 (90.52%)	906434 (90.83%)
CC cells	915 (5.95%)	4552 (7.34%)	20276 (8.14%)	83655 (8.38%)
Rendered Δ	704K	804K	1128K	2497K
PC cells Δ	20K (2,87%)	36K (4,44%)	20K (1,80%)	27K (1,09%)

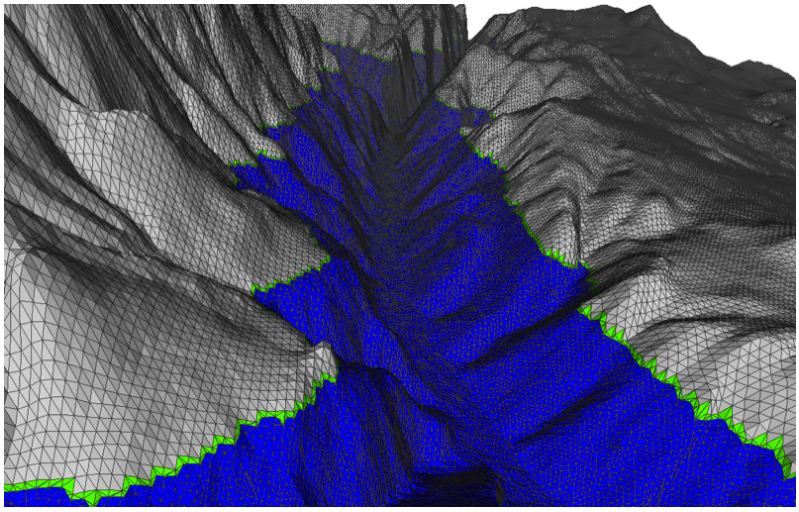
Table 4.2: Detailed description of the test models composition for each grid LOD.

The last two rows show the total number of triangles used for rendering the model and the portion of them corresponding to the adaptive tessellation of the PC cells. Note that the number of PC cells is similar for the three models, which is an usual indicator of the boundaries of the TIN models have similar lengths over the grid. However, in the *Coruña* model, around twice the number of triangles are rendered, since the TIN mesh is much higher detailed than in the other models. Also note that there is an important increment in the number of triangles used for the most detailed levels: 3.5 times more triangles in the the *Sil* and *Pmouros* model and around 2 times for the *Coruña* model.

In the tests, the performance of the GPU implementation as well as the quality of the rendered hybrid models have been evaluated. In terms of quality, the GPU implementation of the HM algorithm improves substantially the visualization of the hybrid terrain models, since the component meshes are effectively joined, without any discontinuities in the borders. Figure 4.10 shows an detail image of the rendered *Sil* hybrid model in solid and wireframe



(a)



(b)

Figure 4.10: HM adaptive tessellation close-up: (a) solid mode; (b) wireframe mode.

GTX 480					
	Method	L0	L1	L2	L3
Coruña	NC + TIN	330.90	319.73	283.64	192.91
	HM	232.20 (29.83%)	233.62 (26.93%)	213.31 (24.80%)	155.42 (19.43%)
Sil	NC + TIN	758.97	707.51	561.93	293.89
	HM	407.83 (46.27%)	409.74 (42.09%)	353.59 (37.08%)	216.00 (26.50%)
Pmouros	NC + TIN	813.42	754.78	591.35	313.56
	HM	276.58 (66.00%)	256.32 (66.04%)	348.41 (41.08%)	210.86 (32.75%)
GTX 280					
	Method	L0	L1	L2	L3
Coruña	NC + TIN	90.30	87.99	79.15	62.02
	HM	76.06 (15.77%)	87.61 (0.43%)	78.99 (0.20%)	61.82 (0.32%)
Sil	NC + TIN	306.06	283.13	213.35	125.66
	HM	196.54 (35.78%)	194.82 (31.19%)	156.93 (26.44%)	101.49 (19.23%)
Pmouros	NC + TIN	354.43	322.90	233.64	134.77
	HM	142.74 (59.73%)	148.63 (53.97%)	159.75 (31.63%)	94.43 (29.93%)

Table 4.3: Performance results obtained with the HM algorithm, measured in FPS, using GTX480 and GTX280 GPUs.

mode (see Figures 4.10(a) and 4.10(b), respectively). TIN mesh is depicted in blue, and the adaptive triangles automatically generated by the HM shader, in green.

The performance results of the GPU-HM implementation are shown in Table 4.3. This table contains the averaged FPS obtained, for every test model, using the GPU-HM algorithm and also the FPS obtained rendering only the NC grid cells and the TIN mesh. Note that the final terrain achieved with the GPU-HM method is a perfectly tight hybrid model, while the result presented by the NC + TIN rendering is a non-coherent model with holes in the boundary region (although it manages to avoid the overlapping between the TIN and the grid meshes). The two values are shown as a way to evaluate properly the performance of the tessellation algorithm performed in the GPU, without taking into account the small processing involved to discard the overlapping grid cells, since it is similar in both cases. Thus, the relative performance drop of the renderer when using the GPU-HM to generate the adaptive tessellation between the meshes boundaries is shown in parentheses. For each sample model, the tests results are divided according the active LOD in the grid ($L0-L3$) for a more detailed view.

As can be seen in the results table, the GPU-HM test implementation achieves the rendering of large terrain models at interactive frame rates. The performance of the PC cells

tessellation performed in the geometry shader is quite good and allows a reliable interactive rendering, although it depends heavily on the number and the LOD of the PC cells being rendered.

Thus, the best performance results are obtained when the coarser LOD (L0) is selected. For finer LODs, performance tends to decrease, as the total number of rendered triangles is much higher. The cost of the HM algorithm implementation, however, is usually higher for coarser grid LODs, which have a low degree of parallelism. As shown in Table 4.2, the number of PC cells nearly doubles for consecutive finer LODs, while the number of PC tessellation triangles rises only marginally. Since a new parallel thread is used for the tessellation of a PC cell, this difference in the number of PC cells directly affects to the degree of parallelism in the implementation, and thus the global performance. This also explains that the performance gain using the more powerful GTX 480 graphic card (480 cores) compared to the GTX 280 (280 cores) is not as large as expected at very high frame rates. Since the GTX 480 has a larger number of cores, at coarser LODs some cores may remain unused.

In conclusion, the application is able to render the largest test model, *Coruña bay*, at the maximum available detail (roughly 3.7 millions of triangles) at 155 FPS using the GTX480 card. Hence, the GPU implementation of the HM algorithm represents a simple and efficient solution for improving the interactive rendering of hybrid terrain models.

4.6 Contributions of the GPU-based implementation

This chapter has introduced the GPU-HM algorithm: a new parallel method for hybrid terrain model rendering based on the original HM algorithm. The method leverages the GPU parallel hardware using a straightforward shader based approach which can be easily included in rendering pipelines compatible with Shader Model 4 specification, as it does not use the more recent Tessellator unit.

The method follows a two phases approach similar to the HM algorithm. Therefore, the hardest and most expensive computations are moved to the preprocessing phase and their results encoded in simple data structures. During the interactive rendering phase, these structures can be easily decoded in the GPU where are also generated the adaptive tessellation triangles needed to join both component meshes.

Besides the modifications needed to adapt the HM algorithm to the Geometry Shader unit, the GPU-HM method features additional improvements as the reordering of the TB vertices in

the TIN geometry buffer, which reduces the storage requirements of the method and improves the data locality; the combination of the convexification and linking steps in the visualization phase to reduce the overall complexity and nesting of the process; and the application of conventional strategies to enhance shaders performance as loop unrolling and the elimination of bifurcations in the shader code.

The performance of the implementation, as well as the quality of the rendered hybrid models, has been demonstrated in several tests, as shown in the results subsection. The GPU-HM implementation manages to render models of several millions of triangles, without geometric discontinuities or overlapping, at interactive frame rates. To our knowledge, no other hybrid terrain rendering algorithm has been implemented using GPUs.

Chapter 5

EHM Algorithm

HM algorithm is a valuable approach for visualizing hybrid terrains, as has been analyzed in previous chapters. It is simple, efficient, inherently flexible —due to the independence of the multiresolution method used in the regular part of the model— and obtains good quality tessellations on the dynamically generated mesh. However, on the HM algorithm level-of-detail techniques are exploited only on the base grid, while TIN models are used to add supplementary details to especially relevant areas. Thus, the number of rendered primitives is unnecessarily incremented by processing the full detailed TIN models in conditions where such a highly detailed models are unnecessary. In these situations, e.g., when the TIN model is placed far from the viewpoint, the number of vertices processed in the TIN mesh will be disproportionate.

In this chapter we introduce a new Extended Hybrid Meshing (EHM) algorithm we have developed to add support for multiresolution TIN models. Overcoming the limitation of using single-resolution TIN models poses a significant challenge since a multiresolution TIN model implies the dynamic simplification of the TIN boundary, which is indeed a key structure in the original tessellation process. Our proposal, presented in [61], is based on an approach similar to the original method, that is, encoding the precomputed local tessellations for any level-of-detail in the model, but in this case both TIN and grid LODs are considered. Furthermore, a new tessellation algorithm has been developed, managing to detect which boundary vertices are active in the current LOD and then consequently readjusting the generation of the tessellation triangles. This dynamic reconfiguration of the precomputed tessellation is possible due to some convenient restrictions in the order of elimination of the TIN boundary vertices,

enforced during the preprocessing. Therefore, our proposal retains most of the interesting properties of the original HM algorithm, like the small memory overhead, high quality local tessellation and preservation of the original data without alterations, and it furthermore delivers a complete multiresolution rendering method of hybrid models. In our solution, the grid and TIN parts are dynamically simplified depending upon the view point and the resulting hybrid model preserves all the visible details and, at the same time, reduces the number of rendered triangles.

In the following sections the EHM algorithm is introduced and discussed. First, the overall strategy of the method is presented. Next, each of the algorithm phases is explained, starting with the visualization phase and the tessellation algorithm, and following with the discussion of the preprocessing phase and the data encoding strategy. Then, the results of some performance and quality tests performed with the EHM algorithm are exposed. Finally, the contributions of the new method are summarized in the last section.

5.1 Algorithm structure

The key point in the original HM algorithm is the unified representation of the convexification triangles for all the levels of detail in the grid, encoded in a unified representation: the TB list. The objective of the EHM algorithm is to extend the capabilities of the HM algorithm, to allow the multiresolution rendering in the grid and TIN parts.

Including a multiresolution TIN in the EHM framework entails dynamic changes in the grid and TIN boundaries during interactive visualization. Thus, only the TB vertices existing in the selected LOD of the TIN mesh are rendered. In this case, a TIN boundary substantially different from the full resolution version will be generated, complicating the tessellation procedure between the models.

If all the vertices in the original TB list are used for generating the convexification triangles, undesired overlapping artifacts will appear. By way of example, Figure 5.1 shows the result of the original tessellation procedure when applied to a multiresolution TIN. The original TIN boundary shown in Figure 5.1(a) can be locally convexified by adding the local convexification triangles portrayed in Figure 5.1(b) in a darker tone. The local convexification of the example has been performed according to the TB list:

$$\begin{aligned} TB = \quad & \{ \ldots\ 11(17), 12(16), 13(1), 14(1), 15(11), 16(2), 17(1), 18(2), 19(1), \\ & 20(1), 21(5), 22(4), 23(1), 24(1), 25(1), 26(1), 27(1)\ \ldots \} \end{aligned} \tag{5.1}$$

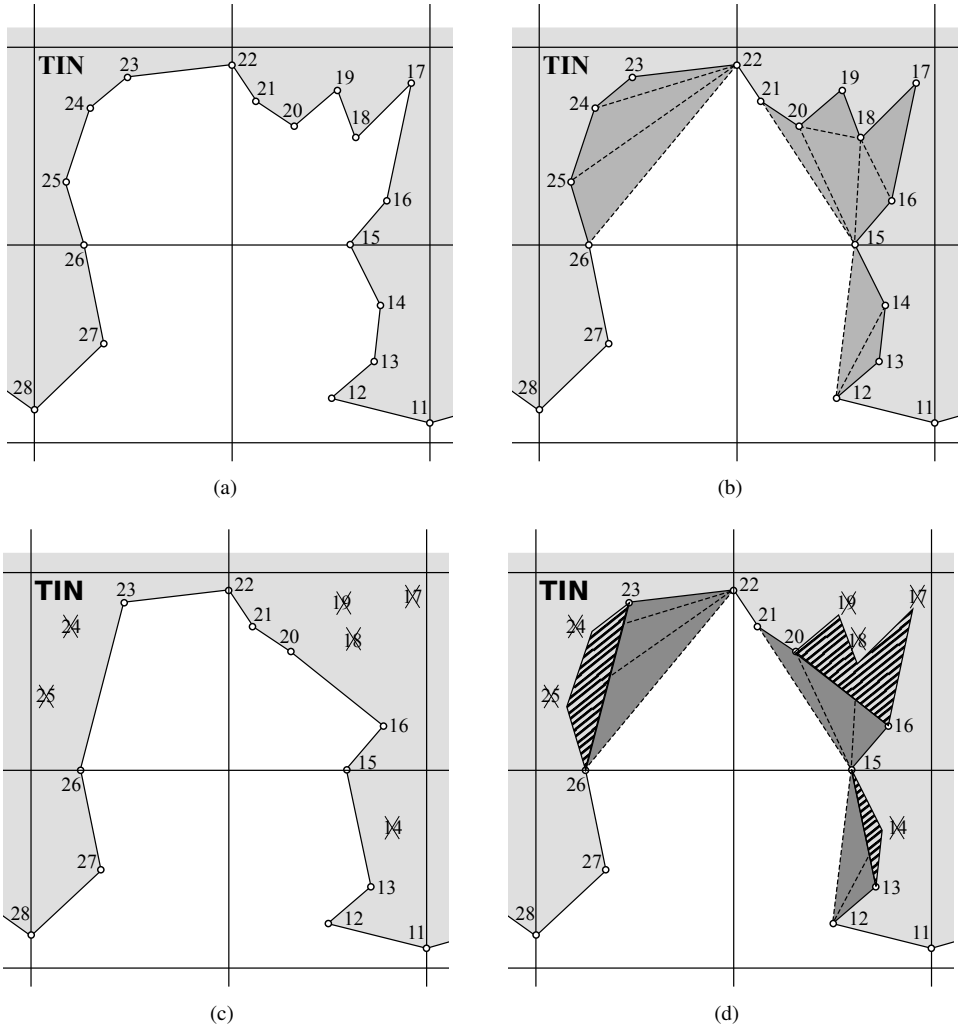


Figure 5.1: Example of overlapping problems in the tessellation between the TIN and grid boundaries when using a multiresolution method in the TIN: (a) original TIN boundary and grid cells; (b) original TIN boundary locally convexified; (c) partially simplified TIN boundary; (d) overlapping areas between the original convexification triangles and the partially simplified TIN boundary.

Figure 5.1(c) reflects the changes in the TIN boundary when some of the vertices are not rendered in a different TIN resolution level. In this figure, *inactive* vertices (i.e., vertices that are not visible in the current level-of-detail of the TIN) have been crossed out and the new simplified TIN shape is depicted; note that, in this case, the grid resolution has not changed. Finally, Figure 5.1(d) shows the overlapping problems that appear if the HM algorithm is directly applied to this TIN resolution level. Following the original convexification instructions encoded in the TB list, some invalid triangles containing inactive TB vertices will overlap the TIN mesh in the regions marked with a stripe pattern. Therefore, proceeding with the original tessellation strategy produces a model with meshing artifacts around the vertices which have been removed from the TIN boundary in the selected LOD.

Our solution combines precomputed multiresolution TINs and dynamic tessellation of the boundaries to discard the invalid convexification triangles generated from the original TB list. In a similar spirit to the HM method, the strategy basically comprises of two phases: preprocessing and visualization. The main steps of the EHM algorithm are:

– *Preprocessing.* During this phase, which occurs only once at the beginning of the process, the data structures needed by the EHM and the multiresolution algorithms are initialized in the following order:

  1. Construction of auxiliary level-of-detail structures used by the grid.
  2. Precomputation of the EHM lists such as the TB list.
  3. Construction of TIN multiresolution hierarchy which is required by most multiresolution TIN methods [30, 21, 35]. Since this hierarchy must comply with the simplification preconditions required by the EHM algorithm, it needs to be created after the EHM lists, as will be explained in detail in Section 5.3.

– *Rendering.* For each rendered frame, the following processing steps are performed:

  1. Extract view-dependent grid: the grid multiresolution method is used for the extraction of a new level-of-detail according to the current viewing parameters.
  2. Extract view-dependent TIN: again, using the standard multiresolution method applied in the TIN, a new level-of-detail is extracted for the TIN part.
  3. Render visible NC grid cells belonging to the computed level-of-detail.
  4. Render the computed TIN level-of-detail (corresponding to the CC cells area).

5. Tessellate and render PC cells using the EHM tessellation algorithm to join both level-of-detail models. The EHM tessellation algorithm is carefully designed to generate the appropriate triangles for the extracted grid and TIN LODs. Therefore, information in the TB list is not blindly processed to generate triangles, but interpreted according to the refined models.

Hence, the EHM algorithm incorporates view-dependent rendering of the TIN mesh and can be regarded as a new solution for achieving a full multiresolution hybrid model. The next section presents the PC cells tessellation algorithm, which corresponds to the final step in the rendering phase. This step is explained before the preprocessing phase for reasons of clarity, since it is easier to understand the requirements in the preprocessing and encoding after having seen the rendering phase.

5.2 Tessellation algorithm for the partially covered grid cells

The EHM algorithm is based on local tessellations by means of the real-time decoding of precomputed data structures. Using multiresolution TINs, however, means that dynamic modifications will occur in the TIN boundary. Therefore, a new tessellation algorithm supporting dynamically simplified TB lists has been developed and is presented in the current section.

The EHM tessellation algorithm generates the two kind of triangles used in the PC cells tessellation: convexification triangles filling the caves in the TB convex hull, and corner tessellation triangles effectively linking the TIN convex hull with the grid cell corners.

In the incremental convexification procedure, the vertices of every triangle generated inside a cave present the form:

$$\{caveStart, i, i + conn(i)\} \tag{5.2}$$

where *caveStart* is the TB index of the cave starting point and i corresponds to the index of any inner vertex of the cave. Defining the TB index of a cave ending point as *caveEnding*, the property $caveStart < i < caveEnding$ stands true for every triangle. The third term, *conn(i)*, represents the connectivity value of the i-th TB vertex. Based on this property, a proper convexification of the TIN boundary is achieved for any active resolution level in the TIN mesh. In this way, two cases can be considered for each TB encoded convexification triangle, according to the currently active vertices on the TB: skip the triangle generation or continue with the generation adapted as needed to the TIN boundary. Triangles skipped are those whose

caveStart or *i* component vertices —the first or second vertex indexes following the TB list order as in Expression 5.2— are inactive in the current TIN boundary. Those triangles are not needed because they are connecting an inactive vertex and instead of filling some area inside a TB cave, they can actually overlap the current TIN mesh. However, when the only inactive vertex in a triangle is the last one ($i + conn(i)$), the triangle is needed to fill the cave and thus it is generated using the next active vertex in TB as ending vertex. Note that the indices of vertices in TB list are maintained throughout the entire process, to avoid invalidating the connectivity based scheme of the tessellation. Additionally, it is guaranteed that any view-dependent TB computed by the TIN multiresolution algorithm will be compatible with this scheme, due to the preconditions imposed in the construction of the multiresolution hierarchy, which ensures that TIN vertices are removed in a compatible order (see Section 5.3.2).

In addition to the TB convexification triangles, the tessellation algorithm also generates corner tessellation triangles linking the TIN mesh with the grid cell corners. These triangles follow the scheme of the original algorithm, i.e., they are sequentially connected to the first uncovered cell corner while the generated triangle does not overlap the mesh. When overlapping is detected, the next cell corner is used to generate the triangle until a new overlapping arises or the process finishes.

An outline of the tessellation algorithm processing each PC cell of the model is shown in Figure 5.2. Basically, the algorithm iterates through the TB vertices contained in the cell and generates the required triangles as needed. Some auxiliary lists to maintain cave related information are initialized at the beginning and conveniently updated in each iteration, before and after the geometry generation stage. In this geometry stage, only one kind of convexification or corner tessellation triangle is generated in a single iteration.

Examining the algorithm in detail, the first step is to verify that the current iteration vertex *i* is active (line 7), otherwise the processing is skipped. Then, if vertex *i* is found to be an endpoint of a previously opened cave in the local TIN convex hull (line 11), the opened caves counter *k* is updated. Next, comes the actual geometry generation step, where a new convexification triangle is emitted (line 22) only when it is needed. This condition is determined by checking whether the vertex belongs to the cell convex hull (line 19), which means that the cave is not active. In this case, the vertex is processed as a convex hull point and thus the required corner tessellation triangles are emitted to link the point to the cell corners; otherwise the convexification triangle is safely generated. Finally, if also marks the beginning of a new or a nested cave (line 25), the cave beginning and ending indices are stored in *startL*

Require: $0 \leq N < TB.size$ and $L > 0$
1: {$N =$ *index of the first TB vertex in cell*}
2: {$L =$ *total number of vertices in cell*}
3:
4: $k \leftarrow 0$ {*counter of currently opened caves*}
5: $startL \leftarrow [\,]$ {*list of cave starting points*}
6: $endL \leftarrow [\,]$ {*list of cave ending points*}
7:
8: **for all** i such that $N \leq i < N+L$ **do**
9: {– *Skip processing for inactive vertices* –}
10: **if** vertex i is **not** active **then**
11: continue
12: **end if**
13: {– *Update counters for this iteration* –}
14: **while** $k > 0$ **and** $i \geq conn(endL_k)$ **do** {*Determine if cave ending has been reached*}
15: $k \leftarrow k-1$
16: **end while**
17: $next \leftarrow i+1$ {*Look for next active vertex*}
18: **while** vertex *next* is **not** active **do**
19: $next \leftarrow next + conn(next)$
20: **end while**
21: {– *Geometry generation* –}
22: **if** $k = 0$ **or** $i + conn(i) \geq N+L$ **then** {*Process convex hull vertices*}
23: LinkToCellCorners(vertex i)
24: **else** {*Process cave inner vertex*}
25: MakeTriangle($startL_k$, vertex i, vertex *next*)
26: **end if**
27: {– *Update counters for next iteration* –}
28: $last \leftarrow i + conn(i)$
29: **if** $conn(i) > 1$ **and** is_active(vertex *last*) **then**
30: {*Init new cave if ending is active*}
31: $k \leftarrow k+1$
32: $startL_k \leftarrow i$
33: $endL_k \leftarrow i + conn(i)$
34: **end if**
35: **end for**

Figure 5.2: Tessellation algorithm for the partially covered grid cells.

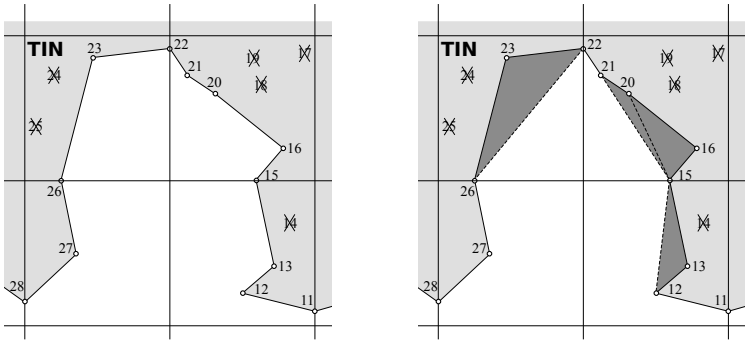


Figure 5.3: Tessellation algorithm example using a multiresolution TIN boundary: (a) active TIN LOD; (b) convexification triangles generated by the tessellation algorithm in the active TIN LOD.

and *endL* lists, and the opened cave counter k is increased. It should be noted that caves are always considered as opened, even if the ending point belongs to a different cell and that cave is later ignored.

Let us illustrate the operation of the algorithm by considering the tessellation of the partially refined TIN mesh presented in Figure 5.1. For convenience, the TIN mesh is pictured again in Figure 5.3(a), while the obtained convexification is represented in Figure 5.3(b). Note that connectivity values of the TB vertices remain unaltered, although some of the original TB vertices are inactive in the current TIN resolution:

$$\begin{aligned} TB = \{ \quad & \ldots\ 11(17), 12(16), 13(1), \cancel{14}(1), \\ & 15(11), 16(2), \cancel{17}(1), \cancel{18}(2), \cancel{19}(1), \\ & 20(1), 21(5), 22(4), 23(1), \cancel{24}(1), \\ & \cancel{25}(1), 26(1), 27(1)\ \ldots \qquad\qquad \} \end{aligned}$$

Table 5.1 shows the operation of the algorithm during the sequential processing of vertices from 11 to 26 in the TB list, divided into three blocks, corresponding to each one of the grid PC cells. The algorithm avoids meshing problems by ignoring the processing of input vertices or generating tessellation triangles when appropriate. For example, let us follow the processing of first vertices in the example TB list. Vertex 11 is active and is not inside any

Vertex i	k	*startL*	*endL*	TB Conv. Triangles	TB Convex Hull
11	1	[11]	[28]	–	[11]
12	2	[11,12]	[28,28]	–	[11,12]
13	2	[11,12]	[28,28]	(12,13,15)	[11,12]
~~14~~	–	–	–	–	–
15	3	[11,12,15]	[28,28,26]	–	[11,12,15]
15	1	[15]	[26]	–	[15]
16	1	[15]	[26]	(15,16,20)	[15]
~~17~~	–	–	–	–	–
~~18~~	–	–	–	–	–
~~19~~	–	–	–	–	–
20	1	[15]	[26]	(15,20,21)	[15]
21	2	[15,21]	[26,26]	–	[15,21]
22	3	[15,21,24]	[26,26,26]	–	[15,21,22]
22	1	[22]	[26]	–	[22]
23	1	[22]	[26]	(22,23,26)	[22]
~~24~~	–	–	–	–	–
~~25~~	–	–	–	–	–
26	0	[ ]	[ ]	–	[22,26]

Table 5.1: Tessellation algorithm operation for the example shown in Figure 5.3.

previously opened cave, thus, it is added to the convex hull. It also has a connectivity value of 17, meaning that a large cave ending in the vertex 28 is opened here; corresponding values are added to the starting (*startL*) and the ending (*endL*) lists. The same processing is performed with vertex 12 and a new nested cave is opened. Then, a new convexification triangle with vertices $\{12,13,15\}$ is generated during the processing of the vertex 13, which has a connectivity value of 1. As the third vertex of the triangle should be $\{13+conn(13)\}$ = 14, which is inactive, vertex 15 is used instead, avoiding any overlapping or meshing problems. Vertex 14 is then ignored, since it is inactive, and the processing continues with vertex 15.

Hence, this strategy of ignoring inactive vertices while proceeding with the cell tessellation may deal transparently with simplified vertices in the TIN boundary. In fact, it also reduces the tessellation overhead since inactive vertices are omitted. This simple tessellation scheme is independent of the TIN and grid multiresolution methods, and works even in complicated scenarios, such as a section of the TB with multiple nested caves and randomly simplified vertices. Moreover, part of the tessellation complexity is eliminated during the preprocessing phase, imposing some preconditions to the TIN multiresolution model which allow

us to simplify the runtime algorithm making convenient assumptions about the simplified TB, as explained in the following section.

5.3 Multiresolution TIN preprocessing

The EHM method does not rely on any particular multiresolution TIN approach. However, the performance and complexity of the EHM algorithm depends partially on the adaptation of its TB-based approach to the multiresolution methods used in the model, as discussed in this section. The first issue to examine is related to the simplification operators used in the preprocessing of the multiresolution TIN model and it is exposed in Subsection 5.3.1. In Subsection 5.3.2, the effects for the EHM algorithm of the preprocessed vertices removal order are analyzed, and two preconditions are derived to guarantee that a valid hybrid model is generated in any case.

5.3.1 Multiresolution TIN algorithm

In the EHM method, as mentioned above, LOD rendering is achieved by using multiresolution models for the grid and TIN meshes. Multiresolution models are generally built by recording a series of simplification operations successively applied on the original geometric data from the terrain dataset. These operations are then organized into a hierarchical data structure, where the coarsest version of the model is at the root, and the fully detailed model in the leaves. Later, during real-time rendering, a view-dependent mesh is extracted from this structure by applying a set of compatible operations. There are many different approaches based on this basic idea [48], using different operators and different methods to build and access the multiresolution hierarchy. Some operators simply delete the faces, edges or vertices of the mesh and do not introduce new elements, while others, replace elements of the original model with a simplified version with lower precision but similar characteristics. A comparative example of both types is shown in Figure 5.4: in Subfigure 5.4(a) a new vertex is introduced with a full edge collapse operator during the simplification of edge $v_a - v_b$; in Subfigure 5.4(b), using a half-edge collapse operator, the same edge is simplified by collapsing the edge endpoint v_a into the other endpoint v_b, without inserting any new vertex.

In fact, the introduction of new vertices in the TB list gives rises to serious consequences for any TB-based approach, such as the EHM algorithm. For example, Figure 5.5 depicts a

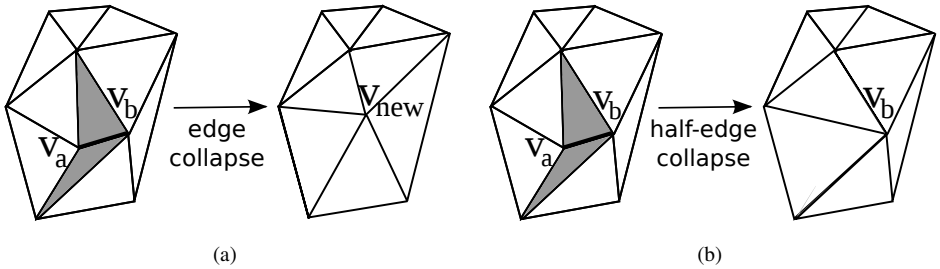


Figure 5.4: Comparison between simplification operators: (a) edge collapse; (b) half-edge collapse.

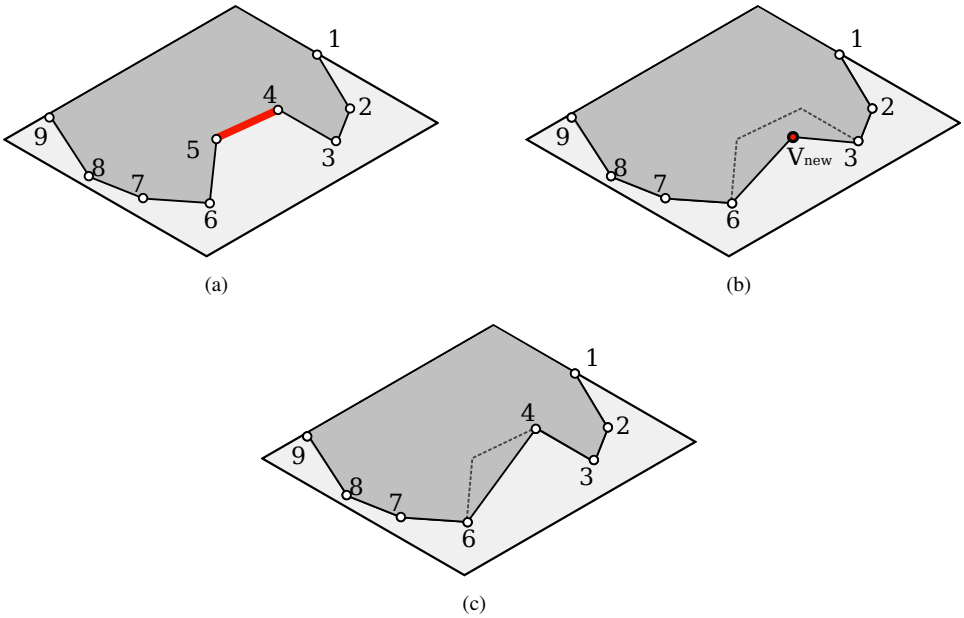


Figure 5.5: Examples of TIN boundary simplification inside a PC cell: (a) original TIN boundary; (b) simplified TIN boundary using an edge collapse operator; (c) simplified TIN boundary using a half-edge collapse operator.

PC cell where a TIN boundary edge inside a PC cell is removed using different operators. The TB information of the original cell in Figure 5.5(a), can be represented as the following array:

$$TB_{original} = \{ \quad 1(1), 2(1), 3(3), 4(1), 5(1), 6(1), 7(1), 8(1) \ \ldots \quad \}$$

In Figure 5.5(b) the TB edge formed between vertices 4 and 5 is collapsed using a full edge collapse operator, while in Figure 5.5(c) the same operation is performed using a half-edge collapse operator. In the first case, a new vertex is introduced in the TIN boundary, which would force the updating of the TB list during visualization to reflect this change:

$$TB_{collapsed\_edge} = \{ \quad 1(1), 2(1), 3(3), \cancel{4(1)}, \underline{NewVertex}(1), \cancel{5(1)}, 6(1), 7(1), 8(1) \ \ldots \quad \}$$

In the second case, the half-edge collapse operator proceeds by removing the vertex 5 endpoint of the edge, without introducing new points in the mesh. The resulting TB list would be:

$$TB_{half\_collapsed\_edge} = \{ \quad 1(1), 2(1), 3(3), 4(1), \cancel{5(1)}, 6(1), 7(1), 8(1) \ \ldots \quad \}$$

The original TB list is not preserved in any case, but using half-edge collapse operators it becomes possible to develop a strategy for ignoring the collapsed vertices during the generation of the convexification triangles, as seen in Section 5.2.

In conclusion, the introduction of new vertices into the original mesh implies that the TIN boundary must be completely recalculated during visualization for each frame. Thus, it becomes clear that operators introducing new elements in the boundaries are difficult to integrate with our approach, due to the implied performance penalty and overall complexity increasing. Using half-edge collapse operators, on the other hand, is compatible with our approach. Therefore, any half-edge collapse based multiresolution algorithm may be used in the TIN mesh, provided that the multiresolution TIN models were built according to the preconditions exposed in next subsection.

5.3.2 Multiresolution TIN preconditions

As stated above our approach does not enforce the use of any particular method, as long as it uses a half-edge collapse operator during simplification. However, to prevent the appearance of any conflictive cases during the tessellation process, we need to guarantee that the dynamically refined TIN boundary complies with certain assumed propositions regarding the vertex

removal order. These propositions can be formalized as preconditions in the resulting TIN multiresolution hierarchy.

Those required preconditions only affect the boundary vertices of the TIN and they do not restrain the elimination of any vertex; i.e. during the construction of the hierarchy the preconditions only modify collapse operations priority, intentionally delaying or bringing them forward, when it is needed. Using this approach the border of any view-dependent refined mesh transparently preserves the consistency of the EHM algorithm. Basically, when a multiresolution hierarchy is built according to these preconditions, any view-dependent refinement of the boundary assumes an inside-out order: the removal of points within the cave before its endpoints is forced, and nested caves are simplified before parent caves, etc. The tessellation algorithm is capable of obtaining a well-formed hybrid model with this simple strategy. The preconditions are presented along with their motivations and reasoning, in the following subsections.

Cave vertices collapse precondition

The first step in the tessellation of the PC cells is to generate the convexification triangles of the TIN boundary. Following the EHM scheme, new triangles are formed for every cave connecting the inner vertices with the beginning endpoint (the initial vertex of the cave in the TB list order), as it is visible from every other vertex in the cave. Vertices belonging to nested caves are not taken into account, as they are joined to their respective beginning cave endpoint.

When the process finishes, generated triangles cover the caves consecutively from beginning endpoints to ending ones. Therefore, all the convexification triangles rely on the corresponding cave beginning vertex and, if it is removed before intermediate vertices in the cave, it will be impossible to complete the convexification of the remaining part of the cave. Furthermore, if the removed vertex marks the beginning of a nested sub-cave, overlapping and wrong oriented convexification triangles are generated from the beginning endpoint of the parent cave. Figure 5.6 shows an example grid cell with the corresponding TB list:

$$\begin{aligned} TB = \{ \quad & 1(12), 2(10), 3(1), 4(4), 5(1), 6(1), 7(1), \\ & 8(4), 9(1), 10(1), 11(1), 12(1), 13(1) \ \ldots \quad \} \end{aligned}$$

This list indicates that several nested caves appear in the cell TB; the cave between vertices $2-8$, for example, contains two sub-caves in vertices $2-4$ and $4-8$.

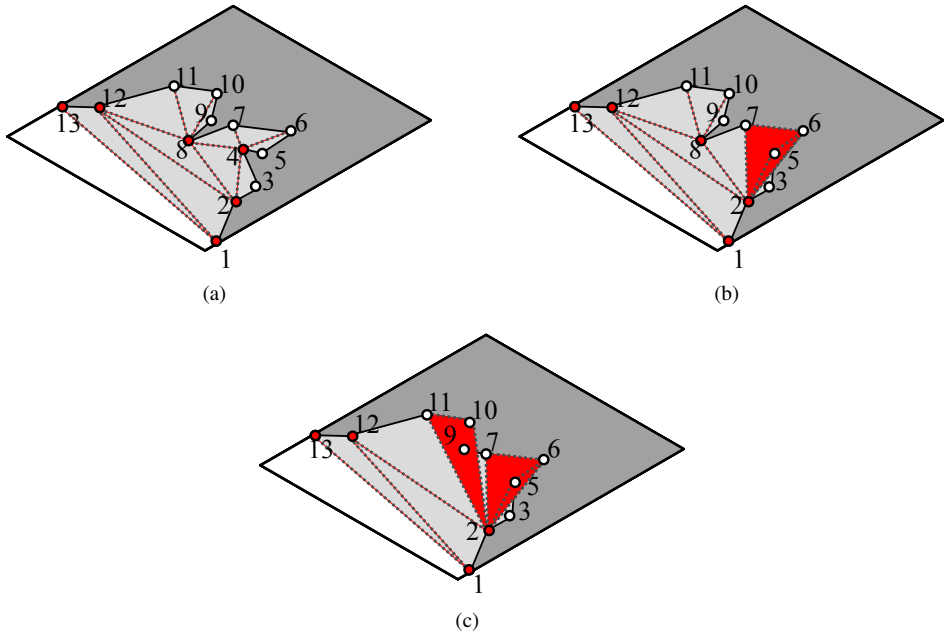


Figure 5.6: Example of a conflictive simplification of the TIN boundary: (a) original full resolution TIN boundary and the associated convexification triangles; (b) resulting mesh after 4 removal; (c) resulting mesh after 8 removal.

As previously noted, vertex 4 (having $conn(4) = 4$) initiates a simple nested cave ending in 8. These two vertices are the endpoints of the cave, while interior vertices between 5 and 7 are connected only to their next consecutive vertices. If vertex 4 is removed before the interior points of the cave, as indicated in Figure 5.6(b), the origin for the remaining convexification triangles of the cave is lost and hence incorrect overlapping triangles, signaled in red in the figure, are formed from the parent cave beginning, vertex 2 in this case. Removing vertex 8, which marks the beginning of the nested cave in vertices between 8 and 12, leads to a similar situation, as shown in Figure 5.6(c). On the other hand, when the cave endpoints are the last ones to be simplified in the cave, conforming convexification triangles can be generated for any partially simplified cave, as illustrated in Figure 5.7. In this figure, the original TIN boundary with the same previously indicated TB list is depicted in Figure 5.7(a). In Figure 5.7(b), vertices $5, 6, 7, 9, 10, 11$ have been eliminated. With the removal of the interior

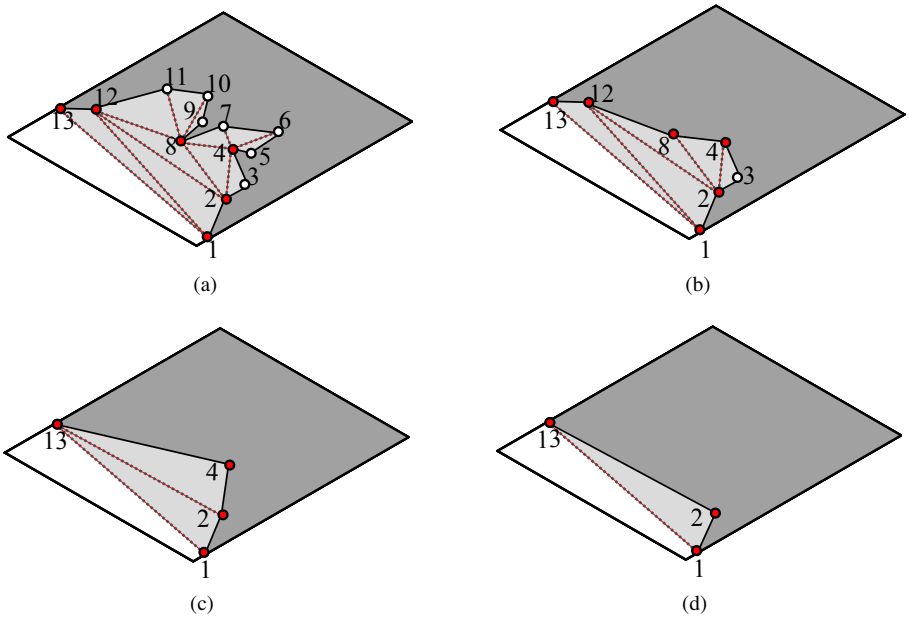


Figure 5.7: Example of an appropriate simplification of the TIN boundary: (a) original full resolution TIN boundary and the associated convexification triangles; (b) resulting mesh after $5, 6, 7, 9, 10, 11$ removal; (c) resulting mesh after $3, 8, 12$ removal; (d) Resulting mesh after 4 removal.

vertices, the cave between vertices 8 and 12 becomes empty and those endpoint vertices can be also removed if not needed. The next simplification step is shown in Figure 5.7(c), where the cave in vertices 8 and 12 finally disappears since the endpoint vertices 4, 8, and the interior vertex 3 have been removed. After this step vertex 4 is not longer necessary, as the interior vertex 3 is not present in the TB, and the simplification procedure may continue. Figure 5.7(d) shows the resulting TB after the removal of vertex 4.

It becomes evident that endpoints are essential for the correct operation of the original HM algorithm, and they must be preserved in the view-dependent refined mesh until all the other vertices of the cave are removed. This requirement can be maintained with no visualization cost, if the relationship is enforced in the multiresolution data structure computed in the preprocessing phase. Thus, we need to set a precondition stating that whatever irregular multiresolution model is used for the TIN mesh, every interior vertex in a cave must be always

simplified before the corresponding endpoints of the cave. Note that this precondition maintains the general flexibility of the algorithm and the precise removal order for every vertex in the cell is not predefined and still depends on the dynamically changing view conditions.

Cell boundaries preservation precondition

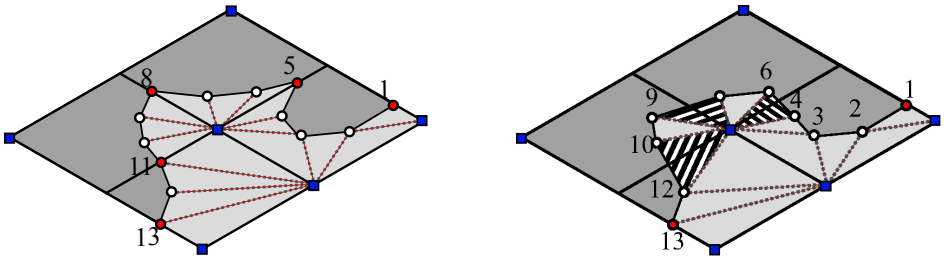
Once the TIN boundary has been locally convexified, the second step in the cell tessellation process is the generation of the linking triangles between the borders of both models. As in the previous step, when using a multiresolution TIN model, the vertices of the boundary may be unpredictably simplified during the rendering phase. Although our new tessellation algorithm is capable of handling the absence of some boundary vertices, an additional precondition related to the cell intersection points must be established to avoid the appearance of unsolvable conflicts in the adaptive tessellation.

Tessellation is performed locally on each PC cell of the grid. At a mesh level, TB vertices are introduced at the intersection points with the cell borders to avoid gaps, as in the original HM approach. Since these intersection vertices belong to neighbor cells, the last vertex of the TB array inside a cell is also the first one in the adjacent one, and thus the cell triangulations are transparently joined and the final mesh does not contain any hole.

However if those intersection vertices are not cave endpoints, they may be removed before other boundary vertices of the cell, leading to the creation of gaps in the resulting mesh. A tessellation example showing gaps between the cells is depicted in Figure 5.8. In this example the TIN already has a convex structure inside each cell so that the tessellation can be performed directly by connecting the TB vertices with the uncovered cell corners of that cell. The tessellation algorithm will proceed joining uncovered cell corners (squares) with active TB vertices (dots) in a cell-by-cell basis.

In this figure vertices $1, 5, 8, 11, 13$ are intersection vertices and, as will be shown below, their elimination could produce undesired results. For the cell delimited by the vertices from 1 to 5, in case (a) as of yet no vertex has been simplified and thus the algorithm will generate triangles linking the three uncovered corners with the TIN convex hull vertices. If vertex 5 is removed, as in case (b), the algorithm will halt the process at vertex 4. Consequently, in the next cell the process will start on vertex 6 and the space falling between vertices 4 and 6 will not be tessellated, giving rise to a hole in the mesh.

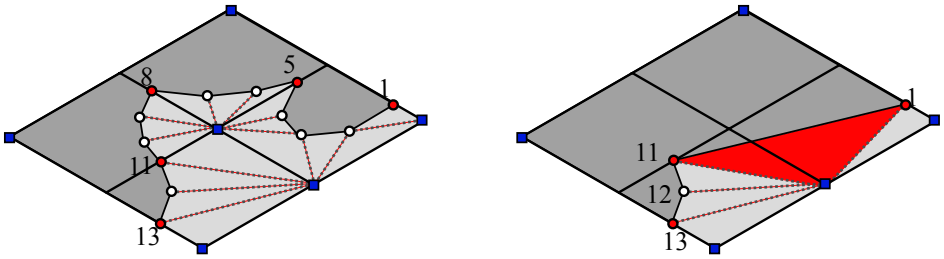
An additional problem caused when very different degrees of simplification are computed for the TIN and grid parts should be also considered. During visualization the multiresolution



(a) (b)

Figure 5.8: Example of elimination of cell border points: (a) original mesh; (b) generation of gaps between neighbor cells.

TIN may locally remove a large number of vertices in the TIN boundary causing active PC grid cells to be left empty of TIN vertices. In this case, linking triangles between both models should be large enough to cross cell boundaries, in order to tessellate the gap between borders. Nonetheless, our algorithm works locally to every cell, and thus operations involving more than one grid cell are not possible. An example is shown in Figure 5.9, where a large cross-cell triangle should be generated. Since in (b) all the TB points between vertices 1 and 11 have been simplified, including intersection vertices 5 and 8, a large triangle should be generated linking vertices 1 and 11 with the respective grid cell corner. Note that this is a fairly simple example, but when using real models with irregular borders, non-local triangles could be



(a) (b)

Figure 5.9: Example of elimination of cell border points: (a) original mesh; (b) generation of non local triangles.

much larger, connecting non neighbor cells and possibly leading to complex boundaries easily overlapped.

To avoid the non-local generation of triangles and holes, a new proposition is derived: cell intersection TB vertices have to be preserved in any view-dependent refinement of the TIN. In our case TIN meshes are considerably more detailed than the base regular grid and many TIN vertices fall in the same grid cell, so this requirement is barely noticeable. Furthermore, TIN meshes are not rendered when very coarse grid LODs are selected for the area of the regular model covered with the TIN, as this implies that the highly detailed geometry added by them is not needed in the current view conditions. Thus, the number of TB vertices needed to preserve the coherence of the final model is small, regardless of which LOD is active in the grid object.

EHM algorithm can be safely used with multiresolution TIN hierarchies fulfilling these two fairly simple preconditions. The conjunction of the real-time tessellation algorithm properties and multiresolution hierarchies built according to the preconditions presented in this section, is enough to guarantee that a well-formed hybrid mesh will be extracted and rendered. In the next section we shall analyze some results obtained in our tests with the EHM algorithm.

5.4 Implementation and results

A test application has been developed to validate the EHM algorithm and evaluate the quality of the hybrid models generated. The application has been programmed in C++ language, using OpenGL and OpenSceneGraph for managing the interactive rendering. OpenSceneGraph is a cross-platform high performance 3D graphics toolkit implemented in C++ and OpenGL. It provides high-level rendering features not found in the OpenGL API as bulk culling on the host CPU, state change minimization, cross-platform access to input devices and file I/O with support for many 2D and 3D model file formats.

The application works as an interactive hybrid terrain models viewer. It loads the preprocessed data structures and the TIN and grid models at startup, and then generates an interactive visualization of the resulting hybrid model. Since the component meshes are rendered by a multiresolution method and both meshes are joined at real-time by the EHM algorithm, the displayed model shows an adaptive level-of-detail depending on the point of view and a visually pleasant appearance.

Table 5.2: Sample models information.

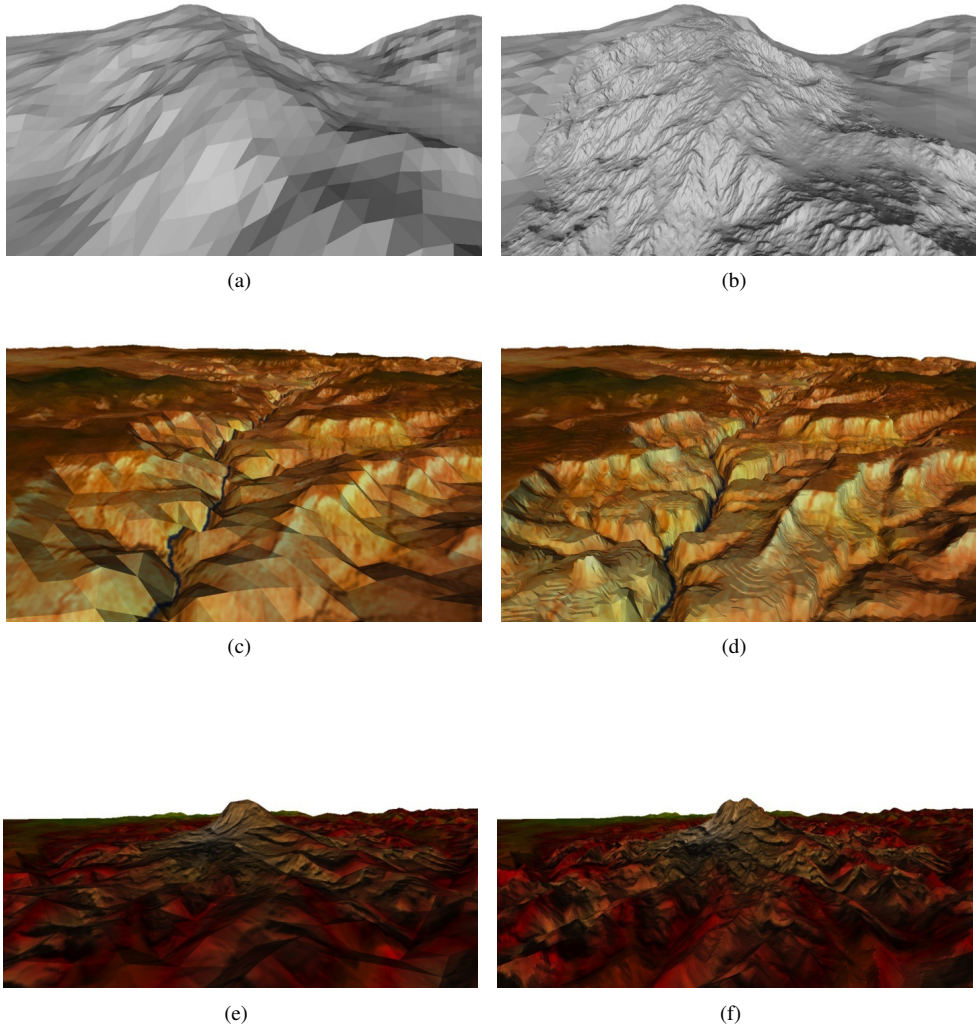
Model	Grid #vertices	TIN #vertices	TIN #triangles
Alpine	4225	193586	385339
GCanyon	16641	34331	68032
PSound	16641	54462	108009

This implementation involves an important complexity in both the preprocessing and rendering phases. Thus, only the TB and GC list management functions of previous HM implementations could be reused. In the preprocessing phase, the application generates the data structures used by the two multiresolution methods employed in the grid and TIN meshes, besides the computation of the EHM data structures. The modification of the preprocessing phase of the TIN multiresolution algorithm to enforce the preconditions detailed in previous sections was specially complex and required a deep understanding of the multiresolution method.

Three sample models, depicted in Figure 5.10, have been used in the tests. These models, synthetically generated from high resolution terrain DEMs, present the typical scenario considered in our approach: a hybrid model formed by a base regular grid and a very highly detailed TIN mesh, covering a extension around the 20-30% of the total model area. The original datasets for the GCanyon and PSound models were obtained from the Georgia Tech repository [79] and the Alpine dataset from Viewfinder Panoramas DEM site [17]. The number of vertices in the grid and TIN parts of each model can be found in Table 5.2. Note that all the TIN meshes have a much higher number of vertices than their respective base grids.

All the tests were performed on a standard PC system equipped with a Pentium IV Quad Core 3 GHz processor, 4GB of RAM and a Nvidia GeForce GTX280 graphic card with 512 MB of video memory.

The multiresolution method used in the grid mesh is a simple quad-tree based approach derived from [72]. Regarding to the TIN mesh, our application employs the View-Dependent Progressive Meshes (VDPM) framework [31]. VDPM is an effective, well known approach for the LOD rendering of irregular meshes with several different implementations, such as [35] or [56]. Furthermore, the VDPM method has been adapted to parallel graphic hardware in [33]. Since a GPU implementation of the EHM algorithm is planned as future work, this was the main reason for choosing the VDPM method. The implementation of the method is

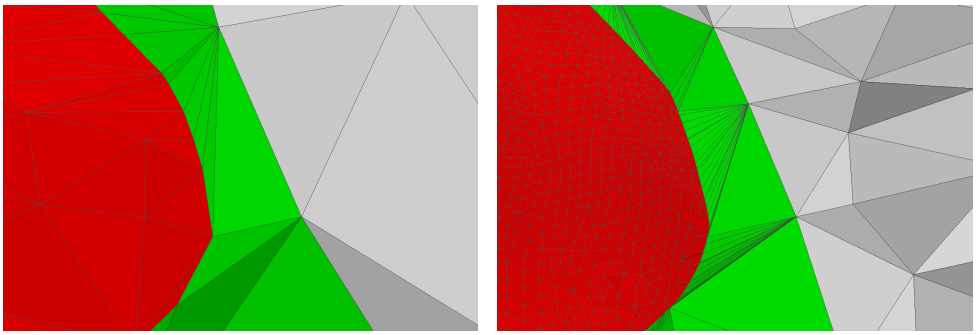


(a) (b)

(c) (d)

(e) (f)

Figure 5.10: Sample models used in the tests. (a) and (d) Alpine dataset grid and hybrid models. (b) and (e) GCanyon dataset grid and hybrid models. (c) and (f) PSound dataset grid and hybrid models.



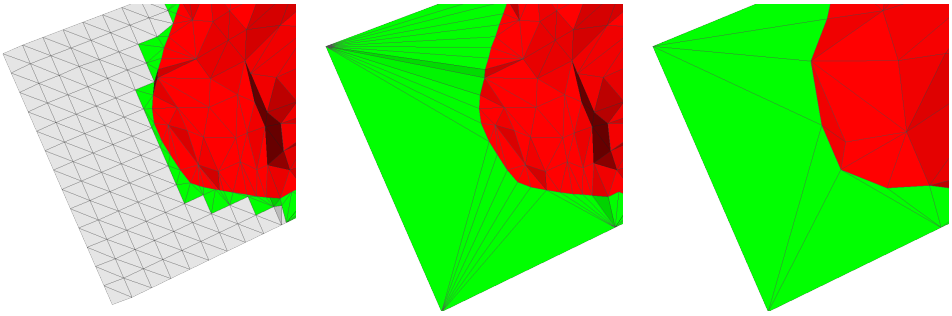
(a) (b)

Figure 5.11: Detail of the tessellated area joining TIN and grid meshes: (a) coarse detailed model; (b) finer detailed model.

based on the approach followed by Kim and Lee [35] since it achieves more abrupt changes between different parts of the refined mesh.

Good quality results were attained in all our tests. The EHM algorithm generated an adaptive tessellation of the area between both meshes boundaries without any visible problems, such as holes or cracks. By way of example, Figure 5.11 shows a detail of the dynamic tessellation (colored in green) generated during the interactive rendering of a hybrid model at two different resolutions. As can be seen in the figure, the tessellation is well adapted to the meshes and, moreover, there is no evidence of reduced quality in the border of the TIN mesh, caused by the modifications introduced in the multiresolution data generation to comply with the EHM preconditions.

The EHM approach is used to dynamically refine both grid and TIN meshes. The obtained tessellation is generated according to the view conditions of the scene and to the overall level-of-detail of the model. In this way, the level-of-detail of the tessellation is coherent with the resolution of both meshes, presenting a tessellation with an adequate density and proportionally sized triangles. By way of example, Figure 5.12 shows the variation in the size of the tessellation triangles depending on the level-of-detail selected on the grid and TIN meshes. Much larger and thinner triangles are generated when using a high resolution version of a TIN with a coarse detailed grid in Figure 5.12(b) (HM approach), while more reasonable sized triangles are obtained with the view-dependent rendering of the TIN in Figure 5.12(c) (EHM approach).



(a) (b) (c)

Figure 5.12: Reduction of tessellation triangles using multiresolution TINs: (a) detailed grid and TIN meshes; (b) coarse grid and fine TIN meshes; (c) coarse grid and TIN meshes.

Table 5.3: Render primitives used during the visualization of sample models using the EHM algorithm.

Model		grid #tri	TIN #tri	EHM #tri
Alpine	Full	8192	385339	2114
	High	5803	271994	1705
	Med	2016	96609	1223
	Low	396	20740	1109
GCanyon	Full	32768	68032	980
	High	22588	48100	791
	Med	7909	17067	628
	Low	1495	3473	569
PSound	Full	32768	108009	1200
	High	23646	74412	1021
	Med	8023	28355	786
	Low	1913	5248	682

Table 5.4: Performance results obtained during the visualization of sample models using the original HM and the EHM algorithm.

Model		FPS	TIN gen time %	EHM gen time %
Alpine	Full TIN	25.54	–	–
	High	4.0	88.87%	0.14%
	Med	13.5	89.65%	0.44%
	Low	87.5	83.81%	2.07%
GCanyon	Full TIN	131.8	–	–
	High	42.8	89.65%	0.72%
	Med	115.8	83.30%	1.28%
	Low	469.4	71.62%	2.62%
PSound	Full TIN	89.0	–	–
	High	23.6	93.17%	0.48%
	Med	68.8	86.99%	1.20%
	Low	373.8	75.32%	2.48%

The results obtained during our tests are shown in Tables 5.3 and 5.4. Table 5.3 summarizes the number of primitives generated during the rendering. Each sample model has been tested in three different view configurations, labeled as High, Medium and Low, using around 70%, 25% and 5% of the original model triangles, respectively. The three columns in the table contain the average number of triangles used in the grid, TIN and EHM component meshes. Here, we should stress that using the EHM method it is possible to render the whole hybrid model using the 5% of the original grid and TIN triangles. The small overhead claims of the EHM algorithm are also verified in this table. Note that only around 2100 new primitives were needed by the EHM algorithm to join the grid and TIN meshes in the worst case, which corresponded to the fully detailed configuration of the Alpine model. Compared to the complete size of the model, this number is almost negligible and is a reasonable trade off for obtaining well-formed hole-free hybrid meshes.

Table 5.4 shows the frames-per-second (FPS) obtained during the visualization of the test models using the EHM algorithm. The performance of the EHM algorithm in the High, Medium and Low view configurations are compared with the performance obtained using the HM algorithm with the full resolution TIN. The first column shows the average FPS obtained during visualization. The second and third columns show the fraction of time spent in the generation of the view-dependent TIN mesh, and in the tessellation of the PC cells. It is patent visible that the performance overhead introduced by the EHM tessellation algorithm in

our experiments is small, representing only a 2.62% of the frame rendering time, in the worst case. The overall gain of the EHM algorithm is highly significant when a low resolution TIN is extracted, e.g., a 4x speed up is obtained for the PSound model in the best case.

The table also shows that the TIN refinement operation is the bottleneck in the visualization phase, since is the most time consuming stage in the rendering pipeline. As mentioned above, the multiresolution rendering of the TIN was implemented following [35], which achieves a more flexible view-dependent refinement than most of the similar methods. However, it is clear that this additional flexibility comes with a cost in performance. Pajarola and DeCoro's implementation, for example, is reported to be five times faster than Kim and Lee's one on a slower machine [56]. Thus, although the general performance of the viewer application allows interactive rendering of large hybrid meshes, using a faster multiresolution rendering implementation for the TIN part would greatly speed up the overall performance, since the specific overhead of the EHM tessellation is almost negligible.

5.5 Contributions of the EHM Algorithm

In this chapter we have presented a new method for rendering multiresolution hybrid terrain models. Our work is based on the original HM algorithm, which supports a multiresolution method only in the regular base mesh of the model, while the finely detailed TIN parts are statically rendered. This new EHM algorithm succeeds in enhancing the original algorithm by adding view-dependent rendering of the TIN meshes, in a simple and lightweight manner.

The method we present works by constraining the simplification order of the TIN boundary vertices, so that tessellation of the region between the meshes boundaries can be performed, regardless of the simplification step of the TIN. These constraints are fairly straightforward to meet, and are enforced during the construction of the view-dependent refinement data structures. Since all these computations take place in the preprocessing phase, no additional time-demanding tasks are performed during real-time visualization. Thanks to this feature, the benefits of our approach are clear, as the whole multiresolution hybrid model can be rendered without incurring in supplementary performance penalties to deal with the varying geometry of multiresolution models. Thus, we have developed and tested a complete framework for the full rendering of multiresolution hybrid terrain models. Using this approach high-quality meshes are obtained, while performance allows for practical applications, as has been presented in the results section. Those results show that the EHM algorithm

performs high quality rendering of complex multiresolution hybrid models involving regular and irregular meshes, in an efficient way and without a heavy performance penalty.

Chapter 6

Hybrid terrain rendering based on the External Edge Primitive

The previous HM and EHM methods use the TB vertices as the key pieces to perform the adaptive tessellation between the regular and the irregular meshes. Therefore, convexification triangles are encoded using just a connectivity value associated to each vertex, while the triangles linking the boundaries of the convexified TIN and the grid cell vertices are generated using the PC list information. Although this technique is efficient in terms of memory space, it presents some inconveniences. For instance, the TB vertices array must be processed sequentially during the tessellation of a grid cell to avoid incoherences in the mesh. This approach also requires the computation of the VC and GC lists during the preprocessing phase, and the creation of vertices in the TIN boundary at the intersection points of the TIN with the edges of the grid cells. Adding these intersection vertices does not modify the overall shape of the tessellation or its quality, but it has a performance penalty in the algorithm, since there are more TB elements to process during the generation of the tessellation, and involves complex calculations in the preprocessing phase to introduce the intersection vertices.

In the External Edge Primitive (EDP) method, the TIN boundary vertices are also consecutively numbered in clockwise order and the convexification process is performed progressively from the finest LOD to the coarsest one. However, TIN boundary vertices are not longer the operational units in the process. Instead, we used the *external edges* of the convexified TB triangles as the fundamental pieces of the procedure to avoid the aforementioned inconveniences of using single vertices.

This chapter begins with an overview of the EDP proposal introducing the key concepts of the new method: the External Edge Primitive, the procedure to link the grid and the TIN boundaries based on the orientation of the external edges, and the culling grid strategy. Next section explains the most conflictive cases that may arise during the visualization as well as the techniques developed to solve them. Both the codification of the precomputed data in the preprocessing and the decodification during the visualization step are explained in the following sections. The results of our experiments with the EDP implementation are shown at the end of the chapter.

6.1 EDP Algorithm overview

The EDP proposal is a new hybrid terrain model visualization method built on a new key primitive, the *external edge*, and new strategies for the creation of the corner triangles and the computation of the overlapped parts of the meshes. This proposal shares some common aspects with the HM and EHM methods, but achieves better rendering performance and also reduces the cost of the preprocessing step due to the changes introduced in the linking primitive used and in the strategy employed to generate the triangles in the visualization phase.

The *external edge* primitive is used as the basic unit for the generation of an adaptive tessellation between the convexified TIN and the grid mesh. For the creation of the corner triangles, a new strategy has been developed, based on the orientation of the external edges of the convexified TIN mesh. Finally, the computation of the overlapped parts of the meshes is performed in a simple yet effective way, using a new culling grid approach. These feature are described in more detail in the following subsections.

6.1.1 External Edge Primitive

The EDP proposal introduces the *external edge* primitive as the base element for the encoding and reconstruction of the adaptive tessellation between the TIN and grid meshes. The use of this primitive simplifies the generation of the triangles at rendering and, as mentioned above, overcomes some limitations of the HM and EHM algorithms, as the limited parallelization of the decoding step. Moreover, using a external edge based encoding for the convexification triangles, minimizes the preprocessing step.

The strategy presented in this chapter requires the precomputation of the local 2D convexification of the TIN boundary for every grid cell at every LOD. This means that the external

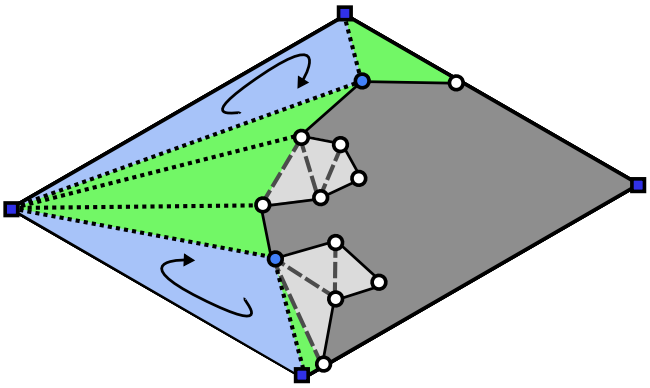


Figure 6.1: Convexification triangles (light gray), corner triangles (green) and shift triangles (blue) used in the local tesellation of a grid cell.

boundary of the TIN mesh is projected over the grid base plane to obtain a closed 2D polygon. According to their position on the plane, the TIN boundary vertices belong to different cells in the grid. Thus, each grid cell contains a polygonal chain formed by the vertices and edges belonging exclusively to the cell, which is an open subchain of the closed TIN boundary polygon. Since the EDP method requires the local convexification of these chains, additional triangles are computed during the preprocessing step in the exterior part of the chain until they become strictly convex polygonal chains. Since these convexification triangles are combinations of the existing vertices in the chain, the convexification does not introduce new points in the mesh.

An *external edge* of a convexification triangle is labeled as a *boundary* edge when it belongs to the locally convexified TIN boundary for a certain LOD, i.e., the edge is not hidden by another convexification triangle. Thus, an edge is external with regard to the triangle and to the boundary at the same time. The EDP method uses the external edges of the locally *convexified TB* (CTB) as the basic encoding and building elements to generate the local tessellation between the convexified TIN and the grid cells.

The information of the external edges is stored in the CTB data array, as it is explained below in Section 6.3. This structure uses a representation of each convexification triangle based on its middle vertex element and its opposite edge. Since every convexification triangle is formed by three vertices belonging to the TB array, they can be defined by the TB indexes of these three vertices, such as $< i-m, i, i+n >$ (where m and n can be different than 1).

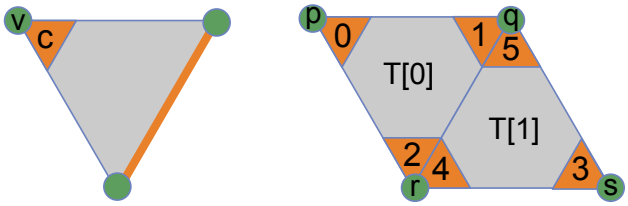


Figure 6.2: Example of a triangle mesh Corner-Table representation from [29].

Consequently, each triangle can be represented solely by vertex i and its opposite edge $|i-m, i+n|$. This representation is encoded in the CTB data array by storing the edge $|i-m, i+n|$ in the position i corresponding to the middle vertex.

This encoding of the convexified TB is both simple and bijective, as each convexification triangle has exactly one representation and each represented element corresponds to one convexification triangle. In next sections, it will be shown how this representation allows an important simplification of the preprocessing and rendering phases.

This representation, based on associating each convexification triangle with its middle vertex and its opposite edge, follows a similar strategy to the Corner Table [69, 70] approach, also called Vertex Opposite Table [29], to define a triangle mesh. The Corner Table structure represents the connectivity of a manifold triangle mesh by defining the association of a triangle with one of its bounding vertex as a *corner*. For each corner c of each triangle, the Corner Table stores the references to the corresponding vertex and to the opposite corner in an adjacent triangle, if one exists. Figure 6.2 shows an example of two triangles, $T[0]$ and $T[1]$, encoded using this technique. In triangle $T[0]$, the orange edge $e(c)$ is the opposite edge of corner c. In triangle $T[1]$, corners 0 and 3 share the same opposite edges, $e(0)$ and $e(3)$, meaning that they are opposite corners.

An example of a locally convexified TB inside a grid cell is shown in Figure 6.3. The following list represents a simplified version of the CTB array including only the vertices in each triangle of the TIN depicted in the figure:

$$CTB = \left\{ \ldots, \frac{|1,4|}{2}, \frac{|2,4|}{3}, \frac{|1,7|}{4}, \frac{|4,6|}{5}, \frac{|4,7|}{6}, \frac{|7,9|}{7}, \frac{|7,9|}{8}, \ldots \right\}$$

where the lower index indicates the position of the opposite edge in the CTB data list. In this case, two different kinds of edges appear in the convexification triangles: *interior edges* of triangles a, b, d and e and *external edges* of triangles c and f.

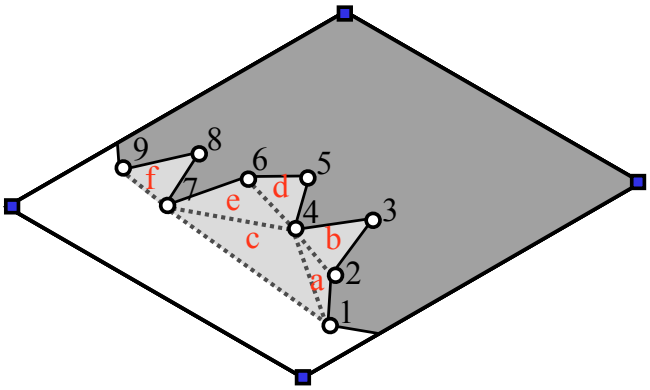


Figure 6.3: Locally convexified TB inside a grid cell.

It is essential to differentiate which triangles are external at each level, because only the external edges of the convexified TB are linked to the grid cell corners in order to generate the linking tessellation of the grid and TIN meshes.

The convexification process is performed level by level, from the finest detailed LOD to the coarsest one, preserving the generated triangles between levels, as in the HM and EHM algorithms. Using this incremental approach, the convexification triangles generated in each cell of a given LOD exist in the local convexifications of the cells in coarser LODs. However, in coarser LODs they will probably not have the external edge in the convexified TIN boundary, since they will be covered by larger convexification triangles of coarser LODs. Therefore, it is required to distinguish which edges are external at each particular LOD, as it will be explained in following subsections.

In conclusion, the result of the preprocessing step in the EDP algorithm is the convexification of the TIN boundary for each LOD of the grid, encoded with the opposite edge representation. For a given vertex i only one triangle $< s,i,e >$ contains i as its middle vertex. Consequently, a given edge $|s,e|$ appears only external edge of the CTB item i. Besides the identification of the s and the e vertices, some additional data is needed for each triangle, for example the activation LOD or the LOD where the triangle is no longer an exterior triangle of the TB. The data structures and the rendering phase will be described with more detail in Section 6.3.

6.1.2 Linking triangles based on orientation edge

In a hybrid terrain model the TIN mesh partially overlaps the base regular grid. In the HM and EHM methods, grid cells were classified according to the covering condition as *Completely Covered* (CC) cells, those falling in the interior of the the TIN projection; *Partially Covered* cells, those intersected by the boundary of the TIN mesh; and *Non-Covered* (NC) for every other cell in the grid. In the EDP proposal, the PC cells are tessellated to effectively link the grid and TIN boundary vertices. This section introduces the cell tessellation algorithm that generates the linking triangles between the grid cell corners and the convexified TIN vertices.

Being C_c a non-covered corner of the cell, and $|s,e|$ an external edge of the TB with s and e vertices as starting and ending points, both elements can be linked together by a triangle $< s,e,c >$ if it is a well-formed triangle that does not overlap the TIN nor intersect any other corner triangles. In a general case, this condition may be complex to evaluate since its result depends on the global shape of the TB and on the position and orientation of the edge related to the grid corner. In this case, however, the global shape of the TB is irrelevant since the potentially overlapping area is the space inside the PC cell, that is, the area between a convex polygon (the TB fragment in the cell) and a rectangular container (the grid cell). Thus, the previous condition can be reduced to a much simpler problem: the selection of the appropriate cell corner to be used in a corner triangle linking a TB edge to a grid cell. The technique used in this proposal only requires the evaluation of the edge orientation.

The orientation of a TB edge can be determined using the normal vector of its 2D footprint on the base grid plane. Comparing this normal vector with the oriented cell diagonal vectors, the minimal angle difference between them happens when both vectors are placed on the same quarter of the unit circle divided by the x and y axis, that its, on the same quadrant. Linking each edge to this cell corner ensures a non-overlapping tessellation while favoring the generation of compact triangles. Note that this simple strategy works when the TIN boundary is properly convexified, since the orientation of the boundary edges in convex polygons always evolves in clockwise direction.

An illustration of the process is depicted in Figure 6.4. At left, Figure 6.4(a) shows how the orientation of the edge normal vector may change in counterclockwise direction in non-convex TB fragments, while in convex fragments it is guaranteed to be clockwise. At right, Figure 6.4(b) shows how the edge normal vector $\vec{n}$ forms the minimal angle difference with the oriented diagonal of the corner placed in the respective circle quadrant of the edge, $\vec{C_0}$ in the example.

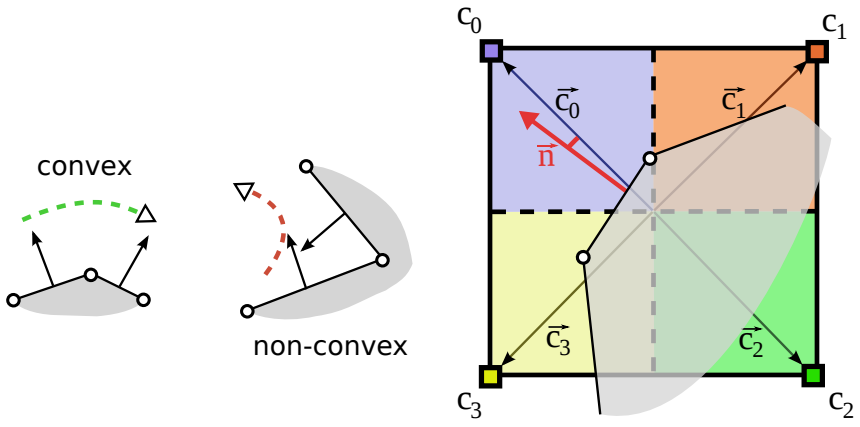


Figure 6.4: Linking TIN boundary edges and grid cell corners based on the edge orientation: (a) change in the orientation of edge normals is only guaranteed to be clockwise orientation in convex TB fragments; (b) the cell corner used to link each external edge of the convexified TIN boundary is defined by the circle quadrant of the edge normal vector.

Using this criteria is easy to decide which cell corner must be used in the linking triangle between the TIN and grid meshes during visualization. In the implementation, the actual computation of the edge normal vector is not required since the orientation can be directly inferred from the signs of the horizontal and vertical components of the oriented edge. The computational cost of the implementation is thus minimal. Additionally, to avoid the generation of holes in the tessellation when the grid cell corner used for the tessellation changes between consecutive TIN boundary edges, some *shift* triangles should be generated at this external edge of the TIN boundary. Shift triangles are formed by the two consecutive cell corners implicated in the *shift* and the TB vertex shared between the two TB edges with different orientations. Linking triangles and shift triangles are the complementary pieces that together build the local tessellation of a grid cell. Thus the number of shift triangles in each cell is fixed by the number of uncovered cell edges.

Figure 6.5 depicts three cells at different LODs which have been tessellated using a different number of shift triangles. In Figure 6.5(a) there are three uncovered edges in the PC grid cell and, consequently, three shift triangles (depicted in blue) would be generated at the start point of edge $|7,8|$. Since edges $|6,7|$ and $|7,8|$ are inter-cells edges, both are oriented toward

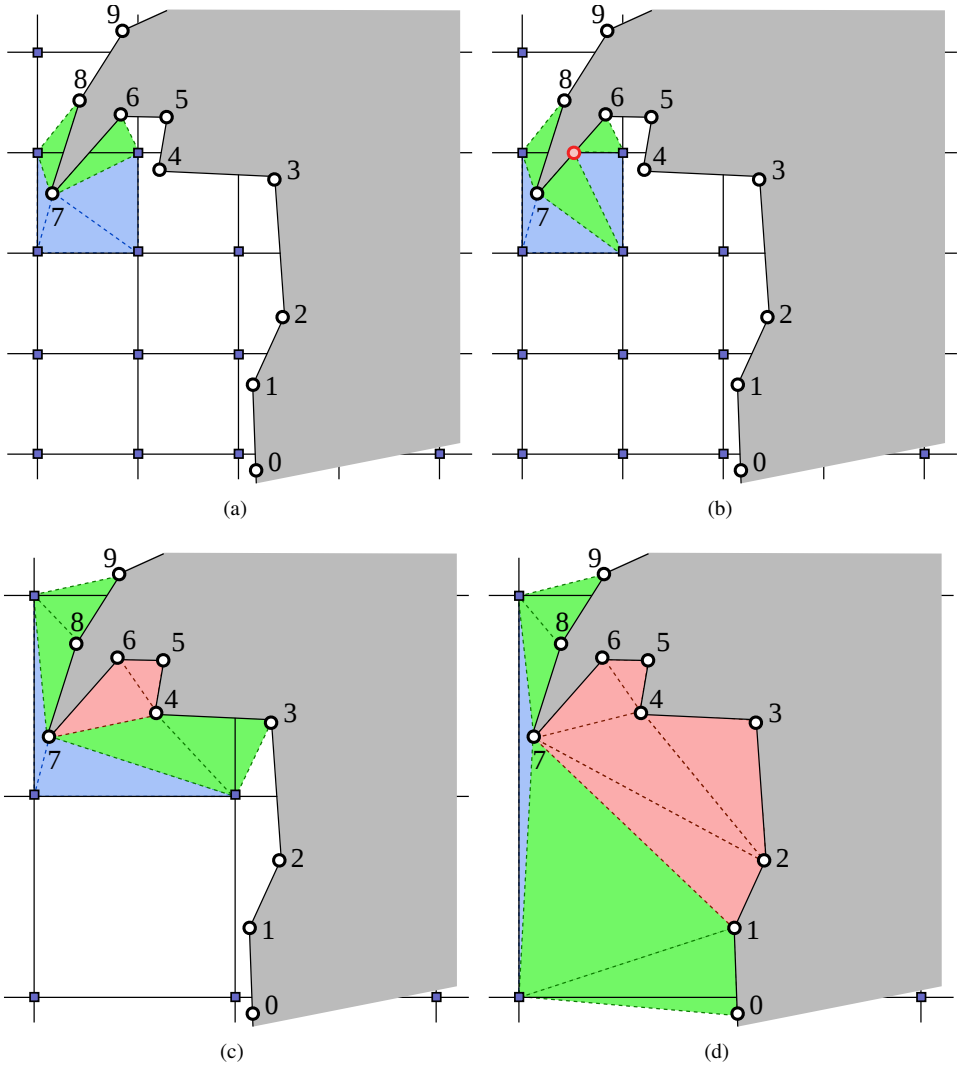


Figure 6.5: Generation of shift triangles in the local tessellation of a grid cell: (a) three shift triangles are needed; (b) previous case solved using only two shift triangles; (b) two triangles at coarser level; (d) single triangle generation at the coarsest level.

the shared cell corner (see Section 6.2.3 for additional details) and thus the orientation change from C_1 (NE direction) to C_0 (NW direction). Therefore, three shift triangles would be linked to the vertex 7, the vertex shared between both implied TB edges. Actually, this very particular case where none of the grid cell corners are covered is the only situation where three shift triangles would be required. To reduce the unnecessary complexity introduced by this unusual case, we have opted to eliminate this case at the preprocessing stage, by introducing an intercells vertex to convert it to a conventional case as shown in Figure 6.5(b). In this way, any linking triangle is associated to a maximum of two shift triangles.

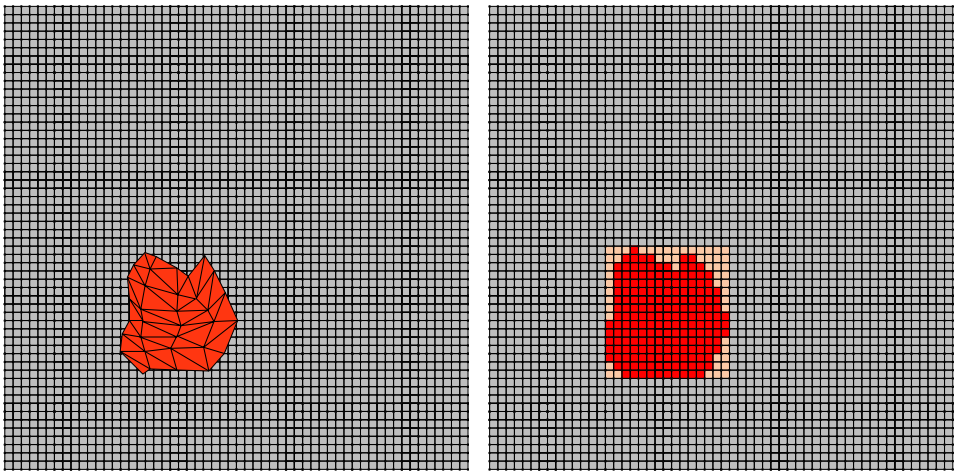
The local tessellation of the cell in Figure 6.5(c) uses only two shift triangles, also connected to vertex 7. In this case, the TB edges orientation changes from C_2 (SE) at edge $|4,7|$ to C_0 (NW) at edge $|7,8|$. The simplest local tessellation is exposed in 6.5(d): since the orientation changes from C_3 (SW) direction at edge $|1,6|$ to C_0 (NW) at edge $|7,8|$, one single shift triangle is needed to link the C_3 and C_0 cell corners to TB vertex 7.

Different implementation strategies can be used to store the preprocessed information about the shift triangles. For example, our implementation uses an additional tag associated to each TB edge in the CTB can be used to indicate how many shift triangles (from 0 to 2) need to be generated prior to linking the edge to the cell corner. Another option, prioritizing the spatial cost over the efficiency, would be to compute the shift triangles during visualization by evaluating the orientation of the neighbor edges at the same LOD.

Therefore, the information needed to generate the tessellation triangles associated to any vertex of the TIN boundary is contained in the CTB structure or may be directly derived from it without using additional data lists, unlike the HM and EHM methods which use several lists. This important property not only improves cache and memory coherence by using a more localized data structure but also makes the rendering process embarrassingly parallel since each external edge may be processed in truly independent way, and not grouped in cell batches as in the HM method.

6.1.3 Culling Grid

In the EDP proposal, the use of additional data lists have been reduced as much as possible. Therefore, the overlapping information between the grid and TIN meshes is not longer encoded in a similar way of the HM algorithm, as a list (GC list) indicating the coverage type (PC, CC or NC) for each grid at each level, computed during the preprocessing for every



(a) (b)

Figure 6.6: Overlapping between the TIN and grid meshes: (a) the original meshes (b) the overlapping table tagging the grid cells overlapped by the TIN.

LOD. Instead, our proposal is based on using a culling method [1] to compute efficiently the grid areas covered by the TIN mesh which is called *Culling Grid*.

Furthermore, due to the massive amount of data pressumably contained in the grid, our proposal employs a multi-resolution rendering algorithm for the visualization of the grid. The Geometry Clipmaps algorithm renders a regular surface mesh as a collection of nested regular square patterns, with decreasing detail from the center of the clipmap, usually choosen as the viewpoint position. Once the center of the clipmap is known, is easy to determine the LOD of any cell in the grid mesh using the 2D distance in the mesh base plane.

Our culling method exploits this property to cull the grid cells overlapped by the TIN before they are sent to the rendering pipeline. Our approach uses a binary mask that encodes the 2D footprint of the TIN to render only the non-covered cells of the grid.

The 2D footprint of an irregular TIN mesh may be a really complex polygon, which makes impossible to find a simple implicit way of encoding the exact correspondence between the grid and the TIN mesh. The bounding box of the TIN is useful to mark the limits of the overall conflicting area between the models but an *overlapping table*, computed only once at the preprocessing phase, is definitely needed to label the grid cells covered by the TIN.

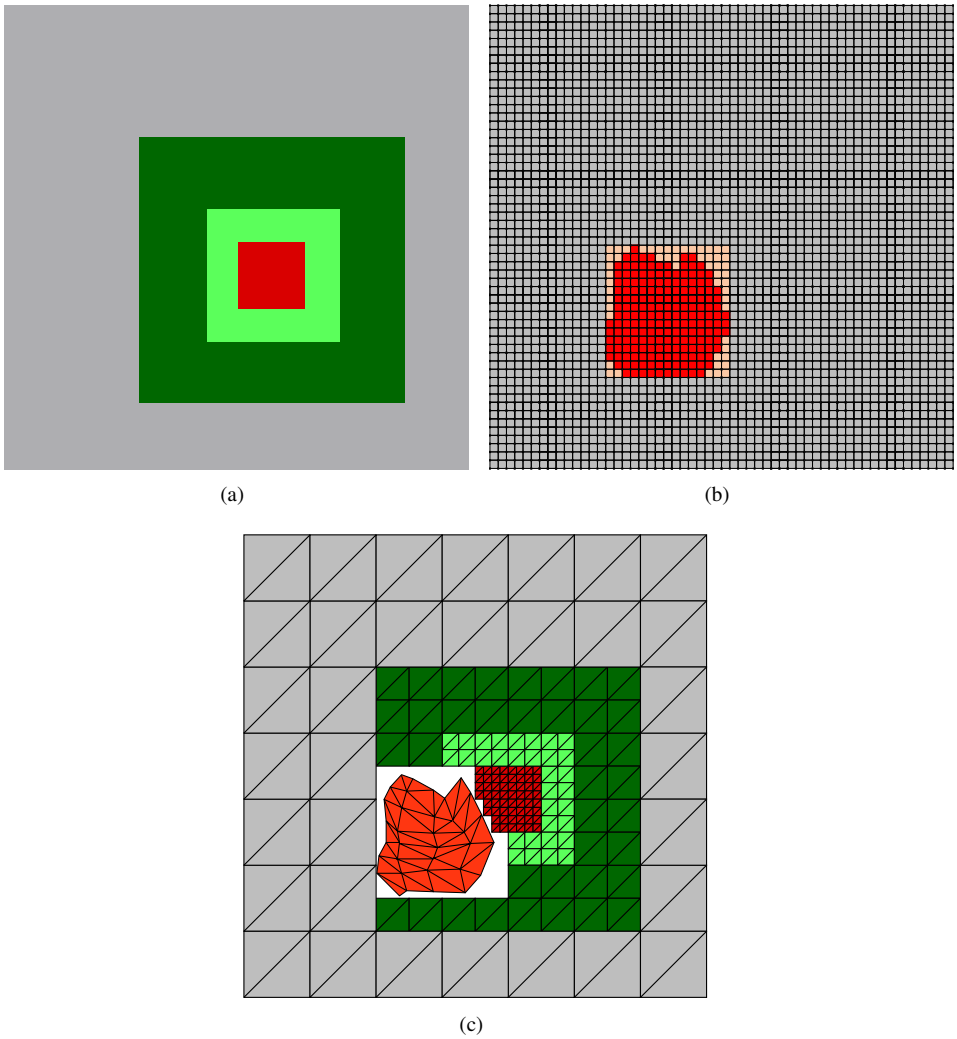


Figure 6.7: Culling of the clipmap against the overlapping table during rendering: (a) active clipmap to be rendered; (b) overlapping table; (c) TIN and grid meshes after culling the overlapping grid cells.

This table only needs to store the overlapping of the grid cells contained by the TIN boundary box, as it is obvious that external cells will not be covered. Moreover, it is not necessary to discriminate between partially and completely covered cells in the grid, since the table is just used to cull away the overlapped grid triangles from the render patterns generated by the geometry clipmap in the rendering phase.

An example of this culling technique is shown in Figure 6.6. The original TIN and grid meshes are shown in Figure 6.6(a). The overlapping table encoding the footprint of the TIN model is shown in Figure 6.6(b). The grid cells where the TIN cover the grid and consequently the grid cells must be culled during rendering are colored in red. The non-overlapped cells included in the mask are labeled in light orange.

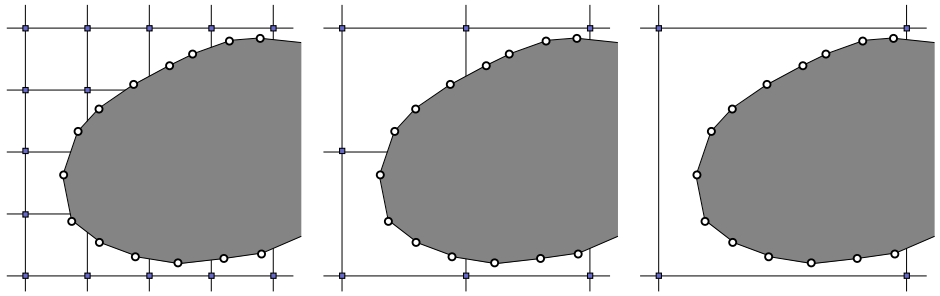
Finally, the rendered result of our multiresolution culling procedure is shown in Figure 6.7: the clipmap is matched against the overlapping mask (in Figure 6.7(b)) before rendering the current clipmap (shown in Figure 6.7(a)) and thus the culled clipmap does not overlap with the TIN mesh, as it is finally shown in Figure 6.7(c). Since the overlapping mask does not depend on the active grid LOD and remains constant during the rendering, the culling process can be performed in a simple and efficient way.

6.2 Fussy cases

The previous sections explain the strategy followed by the EDP algorithm to link the convexified TIN boundary with the grid vertices efficiently. This section explains how the EDP algorithm deals with some cases originated by the convoluted union of the TIN and the multiresolution grid minimizing the added complexity to the decoding and rendering process.

Two different kind of complications may arise during the tessellation of the grid and TIN boundaries. The first one is related to the generation of the linking triangles during the local tessellation of the grid cells and involves two different subcases. Sections 6.2.1 and 6.2.2 explain in detail these cases.

The second complication is related to the union of adjacent locally tessellated cells, since some holes may appear in the frontiers between them. Subsections 6.2.3, 6.2.4 and 6.2.5 explain how the EDP algorithm avoid this holes without introducing additional vertices.



(a) (b) (c)

Figure 6.8: A globally convex fragment of the TIN boundary which remains convex at any level-of-detail: (a) finest LOD; (b) medium LOD; (c) coarsest LOD.

6.2.1 Global convex fragment

A globally convex fragment of the TIN boundary is defined when all of the TB edges are external for every LOD. An example is shown in Figure 6.8 where the fragment of the TIN boundary is convex in every cell at each of the three different LODs represented.

Thus, an already convex TIN boundary fragment poses a problem since our proposal uses the external edges of the convexification triangles as building elements for the adaptive tessellation between the grid and TIN models. Each external edge is associated to the inner vertex of the TIN convexification triangle which it belongs to. Therefore, no convexification triangles are generated in a globally convex fragment and the grid cell corners should be linked to the original edges of the TIN boundary.

Our proposal manages to solve this case at no cost using the normal CTB data structure: degenerated convexification triangles $< i, i, i+1 >$ are created in the global convex fragment. This degenerated triangles provide the required external edges $|i, i+1|$ to link both meshes without adding new geometry in the mesh, since during rendering the degenerated triangles are automatically discarded by the GPU. Note that degenerated triangles are completely valid and regular triangles from the point of view of the EDP method; they do not introduce side effects or exceptions to the general case and are only used to generate the appropriate linking triangles between the TIN boundary to the grid cell corner.

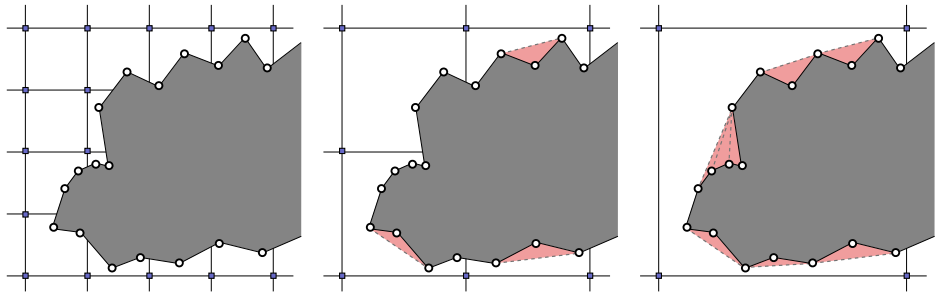


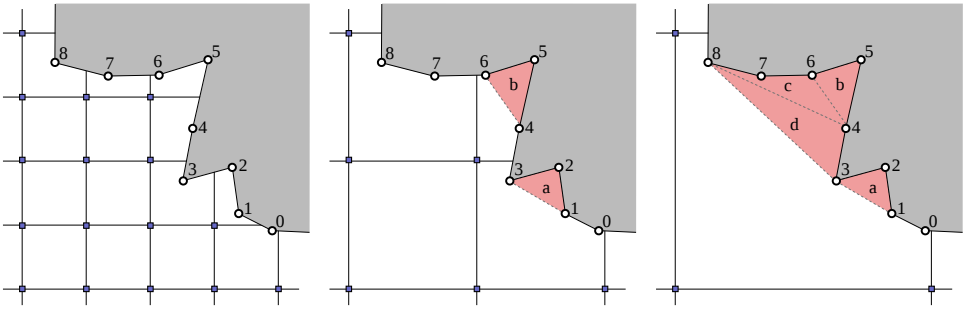
Figure 6.9: A locally convex fragment of the TIN boundary does not remain convex at any level-of-detail: (a) finest LOD; (b) medium LOD; (c) coarsest LOD.

6.2.2 Local convex fragment

Another complicated case appears when fragments of the original TIN boundary are convex only at the finest LODs but become not convex at coarser levels. Although it may seem similar to the issue exposed in the previous section, this case is different and it needs to be specifically addressed in a different way.

A locally convex part of the boundary can be defined as a subset of the TIN boundary edges forming a locally convex polyline inside a grid cell at a specific LOD. When considering the same TB edges at coarser LODs, the polyline does not remain convex. Figure 6.9 contains an example of a TIN boundary with several fragments of the TB being locally convex at the finest LOD, as depicted at Figure 6.9(a), and becoming non-convex at coarser LODs as shown in Figure 6.9(b) and Figure 6.9(c), where the convexification triangles inside the non-convex caves are colored in red.

In the general case, a convexification triangle $< i-m, i, i+n >$, is formed by a external edge $|i-m, i+n|$, and two inner edges $|i-m, i|$, $|i, i+n|$. In the case that $m > 1$ and $n > 1$, the inner edges do not belong to the original TIN boundary and thus they are, at the same time, external edges of other convexification triangles previously generated. At levels where the convexification triangle is being rendered and external, the external edge $|i-m, i+n|$ is linked to the correspondent cell corner. On the opposite case, edges $|i-m, i|$, $|i, i+n|$ are the elements linked to the grid during the processing of the convexification triangles. In locally convex fragments of the TIN boundary, these edges are not external edges of any previous



(a) (b) (c)

Figure 6.10: Local convex TIN boundary fragment represented at several levels of detail with the corresponding convexification triangles: (a) finest LOD; (b) medium LOD with associated convexification triangles; (c) coarsest LOD with associated convexification triangles.

convexification triangles as seen in the previous section. However, degenerated convexification triangles can not be used in this case since real convexification triangles are needed at coarser levels. Therefore, this specific situation has to be considered at the rendering phase when the associated convexification triangle is not active.

The proposed solution does not increase the complexity of the EDP data structures. A simple evaluation of the TB indices of the edge endpoints can detect if the edge is an original TIN boundary edge, that is, if the endpoints are consecutive vertices of the boundary, such as $|i, i+1|$. In the case that a convexification triangle containing one or two original edges as inner edges is not active, the original edge or edges are considered to be external and thus used to generate the appropriate linking triangles. Note that this will not result in overlapping linking triangles if both inner edges are original edges, since in that case one of them is an inter-cells edge, or the convexification triangle would be active.

Figure 6.10 illustrates this case and the presented solution. The TIN boundary is locally convex at the finest LOD as shown in Figure 6.10(a), meaning that every edge of the TIN boundary is labeled as external although there are not associated convexification triangles and then linked to the grid boundary using the degenerated triangles technique exposed in Section 6.2.1. However, at the next coarse level depicted in Figure 6.10(b), the edges are still external but now two convexification triangles, a and b, are also active and thus their external edges $|1,3|$ and $|4,6|$ are used in the linking process. Finally, at the coarsest LOD shown in

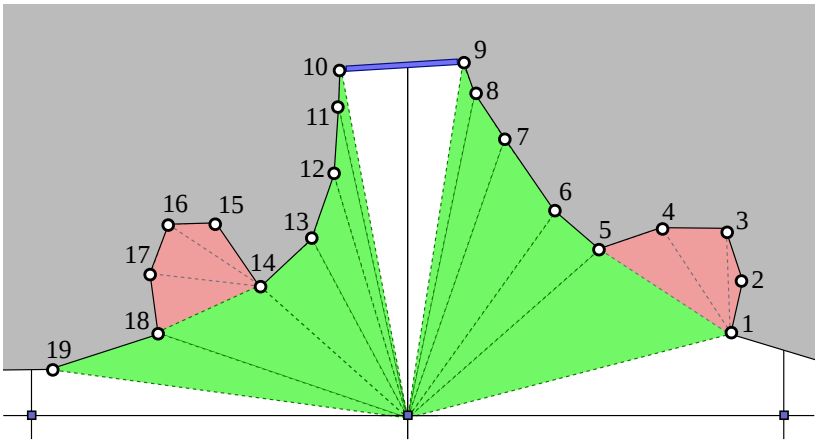


Figure 6.11: Holes appearing between two adjacent grid cells when only the local tessellations of the cells are used. The inter-cells edge is labeled in blue.

Figure 6.10(c), more convexification triangles are active covering the external edge of triangle *b*, which is not longer labeled as external.

6.2.3 Monotone TB fragment between adjacent cells at the same LOD

This section presents the fussy cases involving inter-cell edges, that is, edges crossing cell boundaries, since the EDP algorithm does not introduce new intersection vertices as the original HM algorithm did.

Inter-cells edges are TIN boundary edges whose endpoints belong to adjacent cells. These edges connect the locally convexified TIN boundaries in both cells. For example, Figure 6.11 depicts an inter-cells edge, in blue, between two convexified cells. In the figure, the convexification triangles appear in red and the linking triangles in green. A hole in the tessellation is clearly visible between the two locally convexified TB fragments in each cell. Since those edges do not fully belong to none of both cells, a slightly different strategy is employed to link them to the cell corners adjusted to the particular conditions of both cells.

The detection of the inter-cells edges during rendering is a simple test operation to check if the edge endpoints are placed at two different cells. Moreover, since the identification of

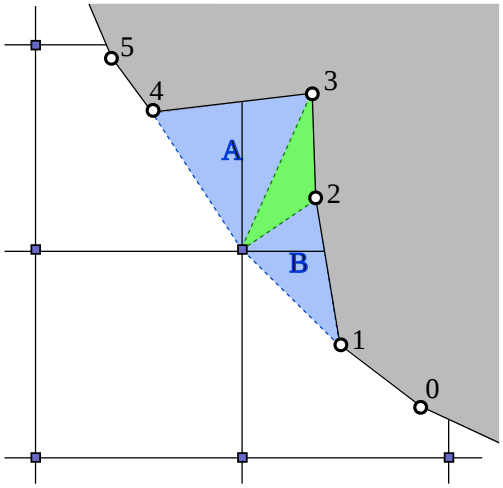


Figure 6.12: Inter-cells triangles filling the holes around the local convexification of a grid cell.

the cell positions and the currently active LODs of the edge endpoints are performed at the beginning of the edge processing, the inter-cells edges are detected right from the beginning.

Once an inter-cells edge has been identified, it is processed in different ways depending on the active LOD of the two involved cells. In the most simple case, when both adjacent cells are rendered at the same LOD, the inter-cells edge is linked to the grid in the render phase by generating a new triangle $< i, i+1, C_{nearest} >$, similarly to any other external TB edges. The $C_{nearest}$ vertex selected to connect the grid to the inter-cells edge is the non-covered corner shared by both cells, that is, the nearest one to the edge. In this case, the orientation of the edge is not taken into account, since only the relative positions of the two involved cells are relevant.

An example of the inter-cells triangles generated with this procedure is depicted in Figure 6.12, where two inter-cells triangles A, B are generated to fill the holes around the grid cell with the linking triangle generated from the edge $|2,3|$. In this case, inter-cells triangle A links the $|3,4|$ edge with the uncovered corner at the bottom of the shared edge between the horizontally adjacent cells, while triangle B links the $|1,2|$ edge to the same corner, since it is also the uncovered corner in the shared edge between the vertically adjacent cells.

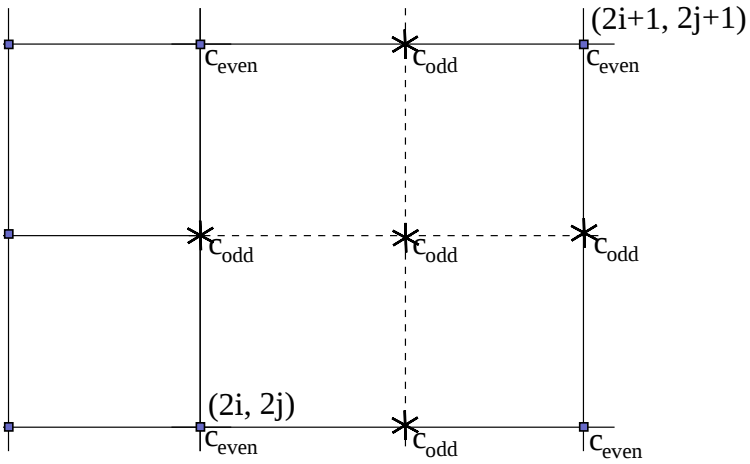


Figure 6.13: Inter-cells tessellation between adjacent cells with different even-odd position.

6.2.4 Monotone TB fragment between adjacent cells at a different LOD

A more complex case involving inter-cells edges appears in the boundaries between different LODs. In our method, the difference between adjacent LODs is exactly of one level as in most of the grid multiresolution rendering methods [48]. This one-level difference, fairly usual in regular multiresolution methods, limits the number of exceptional cases and allows a straightforward solution.

Thus, a cell at level l having (i, j) as its lower index coordinates, generates four subcells at the following finer level $l+1$ with indices $(2i,2j),(2i,2j+1),(2i+1,2j),(2i+1,2j+1)$. The corner vertices placed in a position with odd indexes are not present at level l, e.g. $(2i,2j+1)$, but vertices with even indexes, for example $(2i,2j)$ and $(2(i+1),2(j+1))$, remain present on both levels. Figure 6.13 depicts two adjacent grid cell at two different resolution levels; in the figure even vertices appear at both levels while odd vertices only appear at the finer level.

When the cell at the finer LOD is placed on a even position, this cell and the cell at the coarser LOD share one non-covered even vertex. Therefore, this vertex is used as a connection element, analogously to the previous case explained in Section 6.2.3. A more serious conflict arises when the connection element is placed in an odd position. Blindly using the previous

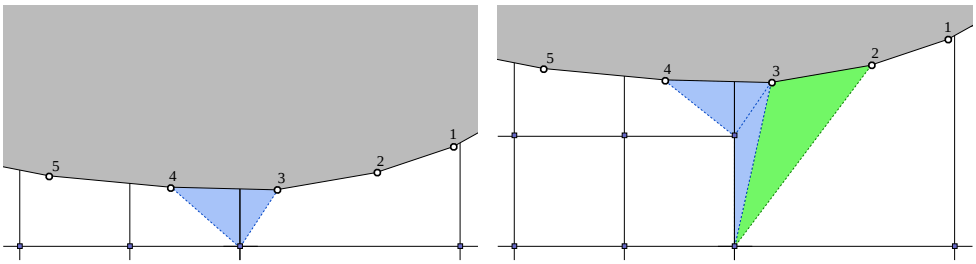


Figure 6.14: Inter-cells tessellation between adjacent cells with different LOD: (a) even position (b) odd position.

strategy in this situation would led to a tessellation with a hole in one of the cells. In order to avoid holes in the result, two additional triangles —instead of only one— are generated by the algorithm to connect the endpoints of the inter-cells segment with the appropriate corners of both cells:

$$< i, C_{odd}, i+1 >$$

$$< i+1, C_{odd}, C_{even} >$$

where C_{odd} is the odd vertex and C_{even} is an even vertex at the finer LOD. Each one of these triangles tessellates the hole in its respective cell in a coherent way with the rest of the tessellation and following the same pattern of the normal case.

Figure 6.14 depicts an example of this technique comparing the case where the connection element is an even position (see Figure 6.14(a)) and the case where the connection element is an odd position (see Figure 6.14(b)). In this case, the two inter-cells triangles, shown in blue, manage to tessellate the hole between the local respective local convexifications by linking the TIN boundary with the corners of both cells.

6.2.5 Non-monotone fragments between adjacent cells

Previous subsections presented the general strategy to fill the inter-cells gaps in the adaptive tessellation of the meshes boundaries and its application in different subcases. The strategy is based on generating linking triangles from the two endpoints of each inter-cells edge and the

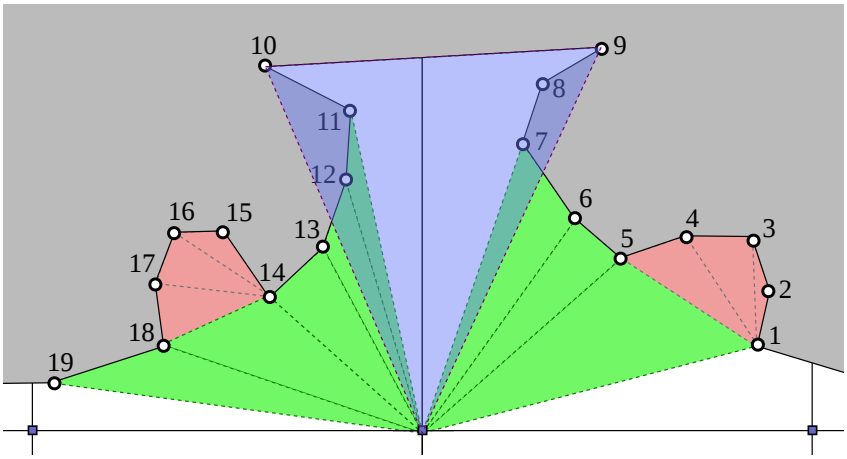


Figure 6.15: Overlapping conflict between a non-monotone TIN boundary inter-cells fragment and the local convexification of the grid cells.

selected cell corners. However, this solution does not consider the possibility that triangles generated to fill the inter-cells gap overlap other parts of the TIN, as it is shown, for example, in Figure 6.15, where the linking triangle generated from edge $|9, 10|$ to the shared corner between the adjacent cells overlaps partially the convexified TIN boundary.

In the example depicted in Figure 6.15, it is visible a break in the monotonicity of the TIN boundary with respect to the parallel grid cells border between the two locally convex cells. The reason is that despite the fragments are locally convex in their respective cells, they are not longer monotone with respect to the grid cells borders. The monotonicity of the TIN boundary breaks whenever the orientation of the local convex hull advances too much in clockwise direction and surpasses the orientation of the inter-cells edge. This monotonicity break means that a line perpendicular to the cell borders has more than one intersection point with the TIN.

Actually, examining in detail the configuration of the tessellation, it becomes clear that the monotonicity is only broken when the final edges of the TIN boundary are oriented towards an uncovered corner of the cell, as it also happens in Figure 6.15. A new condition is thus introduced to avoid this problem in a convenient and straightforward way: the TIN boundary fragment contained in adjacent grid cells must be *monotone* with respect to the grid border

which is linked to. This monotonicity condition is a convenient and straightforward way to avoid potentially conflictive cases.

A common definition of monotonicity for a connected sequence of line-segments or polygonal chain C, extracted from [64] says that C is monotone with respect to a line L if there is at most one intersection point between C and any *sweep line* L_s perpendicular to the *sweeping direction* determined by L. The more common definition of monotonicity is related to closed polygons, but in this case it is not used since only the fragment of the TIN boundary contained by the two adjacent cells is involved and not the entire TIN boundary. Thus, establishing the sweeping direction L as the boundary grid edge parallel to the TIN boundary fragment, then it is guaranteed by the monotonicity requirement that two important conditions are true: every linking triangle connected to the grid corner shared by the two cells would be a proper triangle not intersecting the TIN, and every external edge of the TIN boundary is oriented towards not covered grid cell corners. The first condition is obviously true since, by definition, there are not other TIN boundary edges in the area between the inter-cells edge and the grid boundary. The second condition is also true because the monotonicity requirement restricts the orientation of the external TIN boundary edges avoiding precisely to be oriented toward a covered corner.

In case that the inter-cells fragment joining two local cell convexifications is not already monotone with respect to the grid boundary, the strategy to enforce the monotonicity of the TIN boundary consists in a reconstruction of the monotonicity by generating additional convexification triangles. First, the monotonicity is checked by examining the orientation of the external TIN boundary edges in the two involved cells, starting from the edges placed nearest to the inter-cell. The monotonicity is broken when the checked edge is not oriented toward the same corner than the inter-cells edge or a previous one, clockwise ordered. Whenever an edge is found that fulfills the monotonicity requirement of being oriented to the inter-cells corner, the search is stopped, as the convex nature of the TIN boundary guarantees that any previous edge will also point to this corner or a previous one.

Once the non-monotone edges have been detected, additional convexification triangles involving the non-monotone edges are generated until the inter-cells chain becomes monotone with respect to the grid boundary. The additional convexification triangles would be formed in any case at a coarser LOD, since both TIN fragments would belong to the same coarse cell convexified TIN boundary. Hence, a simple encoding for this solution consists in labeling these triangles as active at the finer level where they are required to preserve the monotonicity.

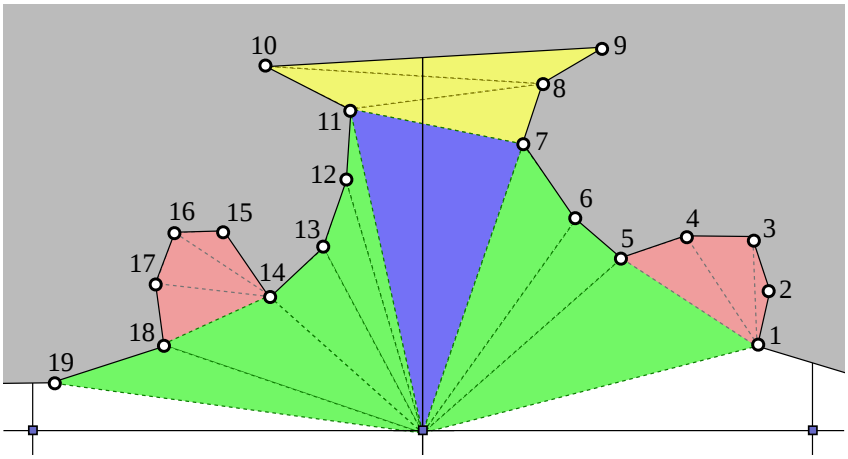


Figure 6.16: Correct tessellation of an inter-cells hole with a non-monotone fragment.

Note that this simple technique manages to fix the problem in the preprocessing phase, without increasing the run time load at all.

Figure 6.16 shows the proper tessellation of the inter-cells holes previously presented in Figure 6.15. The new triangles, pictured in yellow, have been generated to rebuild the monotonicity of the TIN boundary. Although those extra convexification triangles would be generated anyway at a coarser LOD, they are not strictly needed for the local convexification of the cell at the current LOD, but for preserving the monotonicity property and thus simplifying the tessellation of the inter-cells holes. In the rendering phase, the triangles are decoded and rendered as usual, relying only in the precomputed data of the selected edges. With this strategy, the new inter-cells edge $|7,11|$ is monotone with respect to the direction toward the cell corner and thus can be easily connected with the adequate grid corner.

6.3 EDP representation

The hybrid terrain model is defined in the EDP proposal by the geometry of the TIN and grid meshes, and the supplementary data structures used during rendering to rebuild the adaptive tessellation between the models boundaries. The terrain surface of the TIN model is defined, as usual, by the geometry and connectivity data of the surface mesh, i.e., the vertices positions and triangles definitions.

The data structures used by the EDP algorithm to generate the adaptive tessellation joining the grid and TIN meshes, are simple and regular structures, as previously explained. They are computed only once during the pre-processing phase and do not require a computing intensive process. In the visualization phase the tessellation algorithm reads the precomputed data to rebuild the triangles of the adaptive tessellation adapted to the currently active LOD on the grid model.

6.3.1 Convexified TIN boundary data

This data structure encodes the local convexification triangles in the TIN boundary. It contains not only the reference of the vertices forming the convexification triangle but also the information associated to the LOD dependent tessellation, such as the activation and linking levels, and the required extra shift triangles.

The CTB data is an array of items with the convexification information associated to each of the N boundary vertices in the TIN model. Since the TIN is a closed surface mesh, the CTB is a circular array (in clockwise order, due to the orientation of the mesh faces) where the last element $N-1$ is followed by the first one 0.

As mentioned in previous sections, the *external edge* primitive associates each convexification triangle with a boundary vertex, meaning that each element i in the array actually stores the information of the convexification triangle associated to the TB vertex i. The CTB array encodes simultaneously the incremental convexification of the TIN boundary and the linking between both models boundaries. Thus, this single piece of information contains all the information dealing with the tessellation connecting the grid and TIN meshes.

A convexification triangle is defined by two different pieces of information, both of them encoded as a pair of integer values: the two additional TB vertices in the triangle (corresponding to the external edge endpoints), and the range of LODs where the triangle is active and external. Moreover, since the CTB array also collects the linking information, one extra item is require to encode the range of LODs where shift triangles are used. Shift triangles are generated when needed at the starting vertex of the TB edge in the linking triangle. They join the current linking triangle to the previous one, closing the hole in the cell tessellation. Consequently, they are only required when consecutive linking triangles are oriented toward different cell corners.

Let us examine the data encoded in the CTB array. Each item i, corresponding to the convexification triangle whose inner medium vertex is the i-th vertex in the TB, has the following

structure:

$$CTB = \left\{ \cdots, \underbrace{|s,e|(A,B)[shift^1, shift^2]}_{\text{i}}, \cdots \right\}$$

The s (*start*) and e (*end*) fields encode the positions in the TB of the external edge endpoints associated to this convexification triangle. As it was said, the middle vertex corresponds to the vertex in the same position i of the CTB item and thus it is already implicitly encoded. Note that s is a previous vertex in the clockwise ordered TB boundary ring while e is a subsequent one.

To understand the meaning of the A (*active*) and B (*boundary*) fields, let us examine first the usual level ranges in a general case. The generation of the convexification triangles follows an incremental level-by-level approach from the finest level $(L-1)$ towards the coarsest one (0). Therefore, triangles generated in a level A will stay active in any level coarser than A, since they form part of the convexification. On the other hand, when a convexification triangle is external, its $|i-m, i+n|$ edge is part of the external TB, and thus it is used to link the convexified TIN boundary to the appropriate cell corner by a linking triangle. A external convexification triangle may remain external for several coarser levels from the same level A at which it was activated. However, if a covering triangle is activated at a coarser level C, making the previously external triangle to stop being in the TIN boundary, it can not be external again for coarser levels, since the covering triangle will also remain active.

Thus, there are two different indications associated to a range of LODs in a convexification triangle: the range of levels at which the triangle is active and the range of levels at which the triangle is external. A convexification triangle activated at level A remains active for any coarser level; thus the range of activation levels is $[0,A]$. A regular convexification triangle can only be external if is already active and not covered by another convexification triangle. For instance, if a triangle activated at level A is covered by a coarser triangle at level C, the range of valid external levels for the triangle is $(C,A]$. Naming the next finer level to C as B, that is, $B = C+1$, the inclusive interval of levels where the triangle is placed at the boundary is $[B,A]$. Thus, the criteria to establish if a triangle $< i-m, i, i+n >$ is active and part of the boundary at a LOD l can be formalized as:

$$l > A_i \Rightarrow \textbf{inactive}$$

$$B_i \leq l \leq A_i \Rightarrow \textbf{active } \textit{and} \textbf{ boundary}$$

$$l < B_i, (B_i \leq A_i) \Rightarrow \textbf{active } \textit{and} \textbf{ } \textbf{\textit{not}} \textbf{ boundary}$$

Therefore, the A field encodes the range of levels where the convexification triangle is active, that is, the levels where the triangle should be recreated in the rendering step. Due to the incremental convexification approach followed in the preprocessing step, convexification triangles computed at a certain LOD remain valid and active at every coarser LOD. Therefore, storing only the finest level of the range, LOD A, is enough to encode the whole range since it goes from the coarsest level (LOD 0) to the level indicated by A. During the visualization phase, the convexification triangle is generated when the LOD at that cell corresponds to the level stored in A or a coarser one.

The B (*boundary*) field encodes the range of levels where the convexification triangle forms part of the convexified TIN boundary, that is, it is not covered by any other convexification triangle and thus it should be linked to the grid during the rendering.

Note that both the inner edges and the external edge of the convexification triangle can be used to join the grid and TIN meshes. In the most usual case, when the convexification triangle is active, the external edge of the triangle $|s,e|$ will be linked to the correspondent grid cell corner if the active cell LOD is not coarser than the LOD stored in the B field. In the particular case that the convexification triangle is not active and the inner edges of the triangle are original edges of the TIN boundary, they need to be linked to the grid, as it was explained in Section 6.2.2.

Finally, the shift fields $[shift^1, shift^2]$ indicate at which levels the tessellation algorithm needs to generate 1 or 2 additional shift triangles at the starting vertex of the associated linking triangle to avoid holes between the current linking element and the previous one. Since the generation of shift triangles is also an incremental process, at coarser levels, more convexification triangles are active and thus the orientation of consecutive edges changes in a less abrupt manner. Therefore, some shift triangles required at finer levels are not longer needed. On the other hand, a switch triangle used at some specific LOD is also needed at every finer level. The $shift^1$ field indicates the coarsest level where generation of the first extra triangles is necessary, and the $shift^2$ field indicates the same for a second switch corner triangle in case it is needed. Note that those extra triangles are not needed for every CTB item, in which case these fields will tagged with an invalid level value.

A complete overview of the relationships between the CTB data fields and the LODs is depicted in Figure 6.17. Only one endpoint of the range of levels is explicitly stored as the other always agrees with the coarsest or finest level. Note that the *active* level affects both to the active and external ranges, since it is an endpoint of both at the same time. On the other

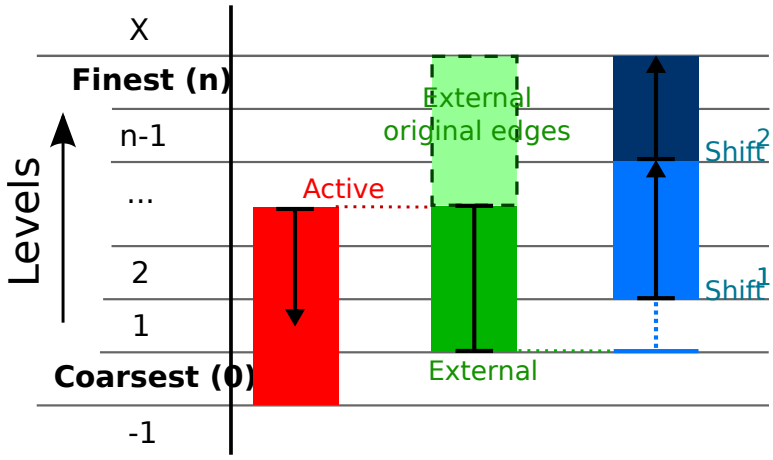


Figure 6.17: Relationships between the different ranges of levels stored in the CTB data items.

hand, when the shift triangles exist, they are not limited by the convexification range. Since they are only affected by the change in orientation between consecutive linking triangles, regardless of whether the linking triangles are generated by original or external edges.

As an example, let us examine the codification of the convexified TIN boundary fragment depicted in Figure 6.10, reproduced again fully tessellated in Figure 6.18.

$$CTB = \left\{ \underbrace{|0,1|(2,0)[0,2]}_{0}, \underbrace{|1,1|(-1,X)[X,X]}_{1}, \underbrace{|1,3|(1,0)[X,X]}_{2}, \underbrace{|3,3|(-1,X)[X,X]}_{3}, \right\}$$

$$\left\{ \underbrace{|3,8|(0,0)[1,2]}_{4}, \underbrace{|4,6|(2,1)[2,X]}_{5}, \underbrace{|4,7|(0,X)[X,X]}_{6}, \underbrace{|4,8|(0,X)[2,X]}_{7}, \cdots \right\}$$

As it is shown in the figure, triangles a and b are associated to CTB items 2 and 5. Therefore, the item number 2 in the CTB array encodes the starting and ending vertices of triangle a as $|1,3|$, that is, vertices 1 and 3 respectively. The middle vertex of the triangle corresponds to its own CTB index 2. The activation and boundary ranges are encoded as $(1,0)$, meaning that it gets active at level 1 and remains active and present in the convexified TIN boundary until level 0. This encoding is verified in Figure 6.18(b) and Figure 6.18(b), where triangle a

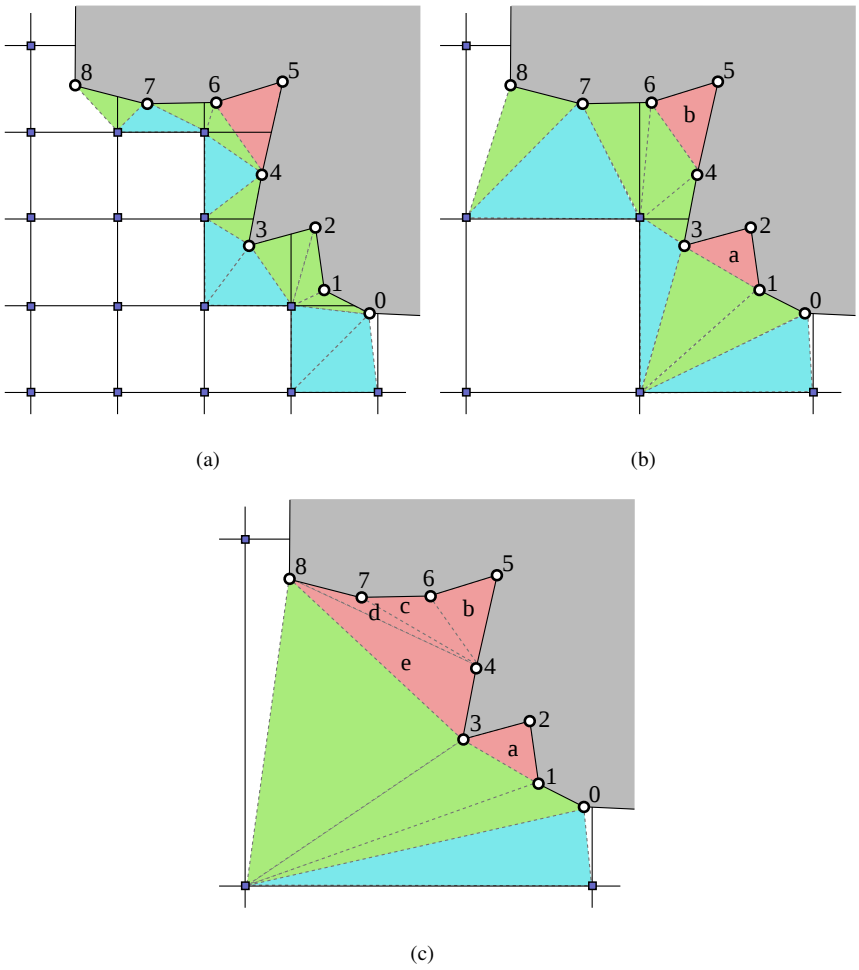


Figure 6.18: Fully tessellated TIN boundary fragment represented at several levels of detail: (a) level 2 (finest); (b) level 1 (medium); (c) level 0 (coarsest).

is both active and external in levels 1 and 0. Finally, the encoding $[X,X]$ indicates that shift triangles are not required at the starting vertex 1, as it is again visible in the figure.

Triangle b, encoded in CTB item number 5, is formed by vertices $< 4,5,6 >$, that is, the middle vertex corresponds to its own CTB index and the starting and ending vertices are indicated by the external edge encoded as $|4,6|$. Regarding the activation and external ranges, the encoding $(2,1)$ shows that the convexification triangle gets active at level 2, which seems wrong since at this level the triangle spreads over two different grid cells. However, the example is right: the convexification triangle is activated on a finer level on purpose to solve the non-monotone intercells egde problem explained in Section 6.2.5. The encoding also exposes that triangle b remains external only until level 1 and, as Figure 6.18(c) shows, at level 0 triangle b is covered by several other triangles and is not longer external. Shift triangles are encoded as $[2,X]$ meaning that only one shift triangle is required at the starting vertex, in this case vertex 4, at level 2 or finer levels. Actually, only the CTB item number 4 (corresponding to triangle e at the coarsest level) requires multiple shift triangles: two shift triangles at level 2, one at level 1, and none at the coarsest level, as encoded in the pair $[1,2]$.

6.4 EDP visualization

This section presents the main algorithm steemps to be performed for the basic algorithm used in the EDP proposal to render a crack-free hybrid terrain model. This phase includes the rendering of the TIN and grid meshes as well as the reconstruction of the adaptive tessellation linking the boundaries of the meshes from the precomputed data structures.

The render phase of the EDP algorithm takes place during the interactive visualization of the hybrid model. For each rendered frame, the algorithm updates the models according to the scene and camera conditions and then generates the tessellation of the space between the boundaries of the regular and the irregular meshes. The general sequence of steps followed by the method is:

1. The data required by the multiresolution method of the regular grid model —Geometry Clipmaps— is updated according to the new position of the viewpoint.

2. The grid cells covered or overlapped by the TIN mesh are culled from the rendering pipeline since they will be substituted by the finely detailed TIN meshes. The remaining cells of the grid are rendered following the normal procedure of the multiresolution method.

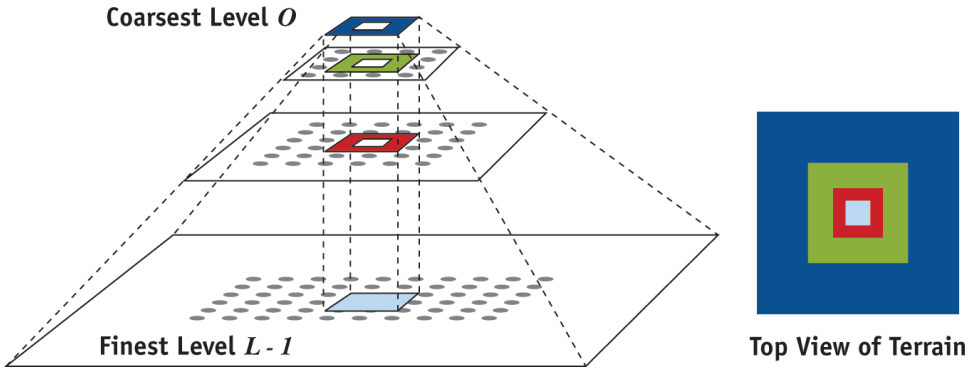


Figure 6.19: Terrain geometry data loaded on a set of nested grid levels in Geometry Clipmaps from [3].

3. TIN meshes are sent to the pipeline and rendered.

4. Finally, the tessellation between the boundaries is rebuild by decoding the active convexification triangles at each grid cell and linking the external boundary edges to the corner grid cells.

In the HM and EHM algorithms the multiresolution method was not enforced by the algorithm, but selectable by the user as long as it permitted to identify the non-covered, partially covered and completely covered cells. The EDP approach specifically uses Geometry Clipmaps [46], since it follows a regular approach easily leveraged by the EDP algorithm to simplify the data structures and the tessellation algorithm during the visualization phase. Geometry Clipmaps uses a set of nested regular grids centered on the view position which are incrementally updated when this position changes. All data levels contains the same number of samples but, since the resolution doubles on each consecutive level, the terrain extension covered by the data is also larger, and the coarsest level contains a very coarse version of the whole terrain. Thus, coarser levels work also as lower resolution fallback versions of finer levels terrain data in case that the update procedure is not finished on time for the frame rendering. Figure 6.19 depicts the extension of the geometry data loaded on every resolution level, on the left, and the final composition of the rendered mesh using using increasingly coarser geometry data from the viewpoint position on the right.

After updating the data structures required by the multiresolution method, the first step in the process is rendering the non-overlapped parts of the grid mesh and the whole TIN. In this step the overlapping table information is used to cull the grid cells overlapped by the TIN mesh from the hierarchy of nested regular grids of the geometry clipmap. Except for this culling step, the procedure for rendering the geometry clipmap is identical to the original procedure described in [46] and thus not detailed here.

Once the multiresolution grid has been updated, culled and rendered, the TIN mesh is sent to the graphic pipeline without additional processing. Then, the adaptive tessellation between boundaries of the meshes is generated to avoid holes in the final hybrid model. The CTB array is accessed to generate all the convexification, linking and shift triangles. Each item i in the CTB array contains precomputed information about vertex i, the convexification triangle $< s,i,e >$, and the external edge $|s,e|$. The CTB array also includes precomputed information about the grid levels, which is compared with the current active LOD of the related grid cells to determine if the precomputed elements are active.

A high level outline of the algorithm is exposed in Figure 6.20. The first step of the tessellation for a CTB vertex i is to identify the grid cell which the vertex belongs to and the active LOD (lines 4–7). Using Geometry Clipmaps this can be immediately computed from the x,y coordinates of the point. The grid cell is derived from the distance of the CTB vertex to the origin of the grid model and the sampling resolution, while the current cell LOD is defined by the distance from the TB vertex to the current clipmap center and by the nested level resolution.

Once the grid cell containing the CTB vertex has been identified, it is time to test if the conditions for the generation of the new triangles are met. To determine if the convexification triangle is active and, therefore, it should be rendered, the current LODs of the cells containing the s, i and e vertices are compared with the A level saved in the CTB array (line 10). If the coarser level of the three vertices is not finer than the A level, then that convexification triangle is actually needed for the convexification of the TIN boundary, and thus it is generated and sent to the render pipeline.

The next step is to check if some linking triangles are also required. If the convexification triangle has been generated, then the external edge of the new triangle may be a boundary edge and thus the active LOD of the grid cell it is compared with the precomputed B level stored in the CTB array (line 12). On other case, only the original TB edges (such as $|i-1,i|$) may originate a linking triangle (line 16).

Require: $N =$ *total number of items in CTB*

1:
2: **for all** i such that $0 \leq i < N$ **do**
3: {– *Compute the grid cell and the current level* –}
4: $start\_level \leftarrow$ GetLOD(vertex s - $clip\_center$, $clip\_size$)
5: $middle\_level \leftarrow$ GetLOD(vertex i - $clip\_center$, $clip\_size$)
6: $end\_level \leftarrow$ GetLOD(vertex e - $clip\_center$, $clip\_size$)
7: $current\_level \leftarrow$ Min($start\_level$, $middle\_level$, end_level)
8:
9: {– *Generate convexification triangle* –}
10: **if** $current\_level \leq A_i$ **then**
11: MakeTriangle(s, i, e)
12: **if** $current\_level \geq B_i$ **then**
13: $link\_edges \leftarrow |s,e|$
14: **end if**
15: **else**
16: $link\_edges \leftarrow inner\_edges$ which are $external\_edge$
17: **end if**
18: {– *Generate initial shift triangles* –}
19: **if** $current\_level \geq shift_i^1$ **then**
20: MakeShiftTriangle(i, 1)
21: **if** $current\_level \geq shift_i^2$ **then**
22: MakeShiftTriangle(i, 2)
23: **end if**
24: **end if**
25: {– *Generate linking triangles* –}
26: **for all** $edge$ in $link\_edges$> **do**
27: MakeLinkTriangle($edge$)
28: **end for**
29: **end for**

Figure 6.20: Pseudo-code of the EDP tessellation algorithm during the rendering phase.

Model	Grid #verts	TIN #verts	TIN Δ	TB #verts
Alpine	4225	193586	385339	1831
GCanyon	16641	34331	68032	628
PSound	16641	54462	108009	913
Coruña	1000K	878K	1739K	13361

Table 6.1: Terrain sample models statistics.

Finally, the initial shift triangles are also tested and generated at the appropriate levels (lines 19–24), and in case that some linking triangles have been considered active, they are also generated (line 27).

6.5 Results

An implementation of the EDP algorithm described in this chapter has been included in a terrain visualization application to evaluate the results produced with several sample terrain models. The implementation, developed in C++ language, uses OpenGL for the hardware accelerated rendering and the Qt Framework as cross-platform user interface layer. For the non real-time geometric preprocessing of the terrain models, the GEOS Geometry Engine and several polygon tessellation libraries were used. Our visualization application uses a CPU implementation based on the original work by Lossaso and Hoppe [46]. We have decided to use a CPU implementation instead of a more capable GPU version to avoid issues with the GPU during the initial development phase of the EDP algorithm and also to simplify the debugging and testing.

The sample terrain models employed in the testing are depicted in Figure 6.21. These models have been specifically selected to compare the results with the HM-based approaches, as they have been also used in previous chapters of this thesis. As it was documented in Chapters 3, 4 and 5, the datasets for the *GCanyon* and *PSound* models are publicly available through the Georgia Tech repository [79], the *Alpine* dataset was obtained from the Viewfinder Panoramas DEM site [17] and the *Coruña* model from the Spanish GIS database (IDEE) [34]. Figure 6.21 shows a basic wireframe view of the models. Their respective sizes and data elements counts are compiled once again in Table 6.1.

The hybrid models generated by the EDP method are perfectly closed and the adaptive tessellation between the TIN and grid models do not contain any cracks or holes. In Fig-

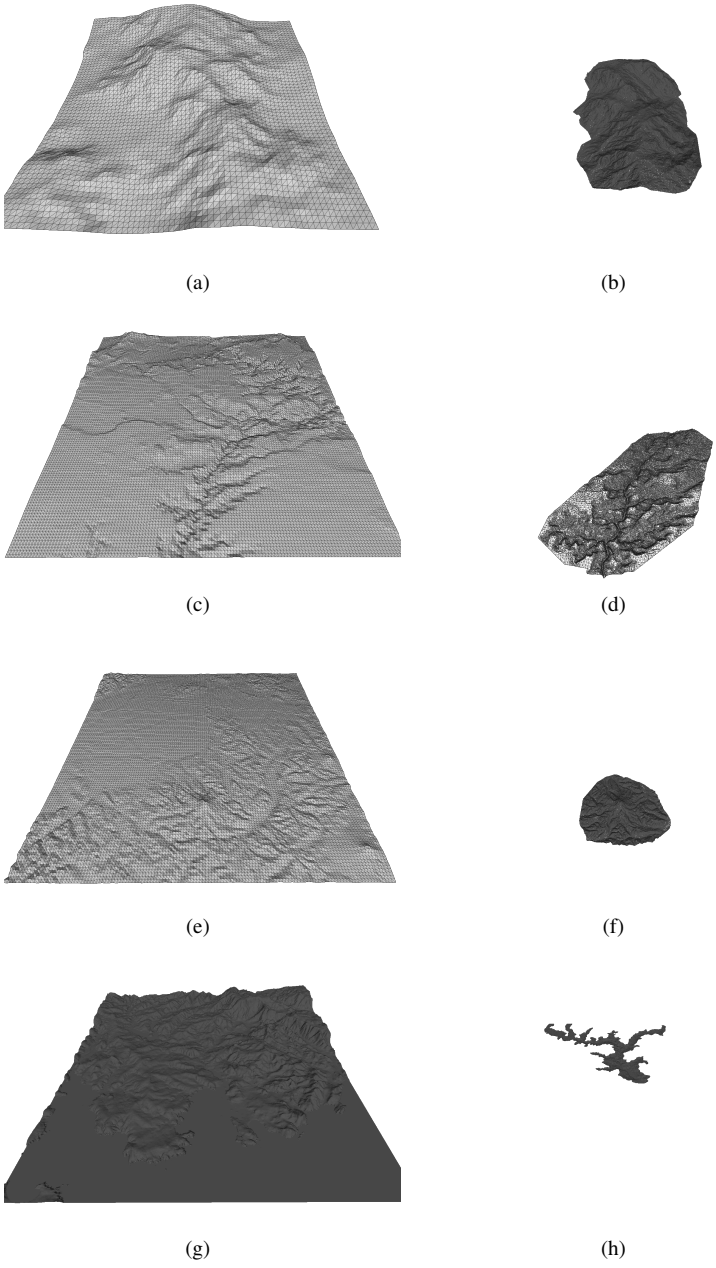


Figure 6.21: Terrain sample models used in the tests: (a) and (b) *Alpine* dataset, grid and TIN meshes; (c) and (d) *GCanyon* dataset, grid and TIN meshes; (e) and (f) *PSound* dataset, grid and TIN meshes;(g) and (h) *Coruña* dataset: grid and TIN meshes.

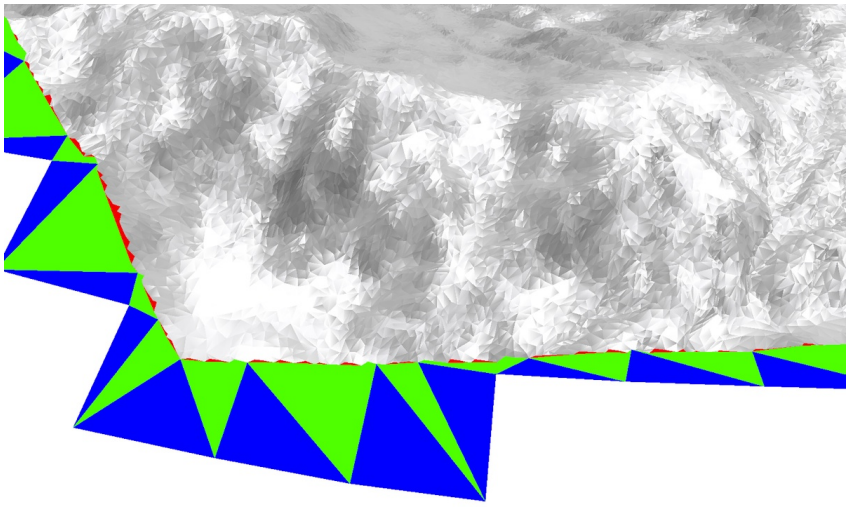
Model	Original	HM		EDP	
	TB Verts. #	TB Verts. #	TB increase %	TB Verts. #	TB increase %
Alpine	1831	1967	7.43 %	1833	0.11 %
Gcanyon	628	801	27.55 %	645	2.71 %
PSound	913	1052	15.22 %	919	0.66 %
Coruña	13361	17307	29.53 %	13361	0 %

Table 6.2: Increase in the number of the TIN boundary vertices using the HM and the EDP algorithms.

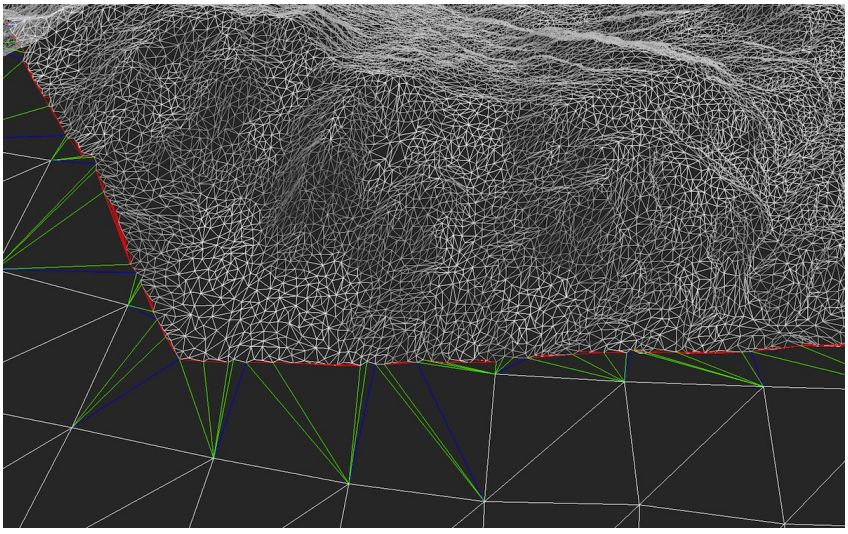
ure 6.22 is shown an example of the adaptive tessellation obtained using the EDP method in a hybrid terrain model. In the figure, the different types of triangles are marked with different colors: convexification triangles are colored in red, corner link triangles in green and corner shift triangles in blue. Note that although the TIN triangles are much finer than the grid cells, the adaptive tessellation between the borders is perfectly closed. Furthermore, the tessellation remains coherent even in the boundaries between adjacent cells, using the convexified TIN external edges as link elements between the meshes without including new vertices in the intersections.

In our tests, the visualization application uses the EDP algorithm to dynamically tessellate and render the hybrid terrain models according to the conditions of the scene and the viewpoint. The involved steps are not very different of the HM algorithm but the overall results differ greatly in the number of rendered primitives used in the adaptive tessellation between the models boundaries. This difference is mainly a direct consequence of the reduction in the number of TIN boundary elements generated by the EDP algorithm compared with the HM algorithm. The EDP algorithm does not need to create a TIN boundary vertex in the intersection point between two adjacent grid cells, as the HM algorithm does. Thus, using the EDP algorithm, new TIN boundary vertices are only inserted in the case that the TIN boundary footprint crosses a grid cell which does not contain any TIN boundary points. Table 6.2 shows a comparison between the original number of TIN boundary vertices of the sample models and the number of elements generated by the HM and the EDP algorithms. The HM algorithm inserts a number of additional TIN boundary vertices directly proportional to the number of PC grid cells in the hybrid model.

Test results also show the differences in the number of graphic primitives rendered during the visualization of the models depending on the algorithm used. Table 6.3 collects the



(a)



(b)

Figure 6.22: Detail of the tessellation between the TIN and grid meshes in a hybrid model generated by the EDP method. (a) Solid mode. (b) Wireframe mode.

Model	HM Δ	EDP Δ	EDP reduc. %	Shift Δ	real EDP reduc. %
Alpine	2111	1977	6.35 %	144	6.81 %
Gcanyon	977	821	15.97 %	176	19.48 %
PSound	1202	1069	11.06 %	150	12.64 %
Coruña	21281	17335	18.54 %	3974	22.80 %

Table 6.3: Triangles used in the adaptive tessellation of the sample models at the finest LOD using the HM and the EDP algorithms.

numbers of triangles used in the adaptive tessellation between the meshes boundaries. The second and third columns contain the total number of triangles needed by each algorithm, while the fourth column explicits the triangle reduction achieved with the EDP algorithm. However, note that the number of *shift* triangles (shown in the fifth column) depends only in the uncovered grid vertices of the PC cells and thus remains constant no matter the algorithm used for the tessellation. Hence, the real reduction in the number of triangles obtained by the EDP algorithm is larger, since it should be measured without taking into account this constant triangles, as shown in the sixth column of the table.

The exposed results clearly confirm that the EDP algorithm introduces less unnecessary points in the TIN mesh and consequently a smaller number of triangles is used during rendering. This reduction is mainly due to the change in the precomputed data encoding/decoding strategy during the precomputation and visualization phases.

In the original HM algorithm, each item in the TB array contained a vertex *handle* — the index of the vertex in the vertex buffer— and the *connectivity* value. An optimization was later introduced in the GPU-HM and the EHM algorithms which managed to remove the vertex handle from the TB data item, by reordering the mesh vertices buffer. Thus, the position of a vertex in the TB array is the actual index of the vertex in the buffer, making the *handle* field redundant and unnecessary. In addition to the TB array, the HM algorithm used two lists, the PC and the GC lists, which encoded the tessellation information for each PC cell and the correspondence between models, respectively.

Table 6.4 exposes the storage requirements of the specific HM algorithm data structures, that is, the TB array and the PC list since the GC list is equivalent to the overlapping table used in the EDP algorithm. It is important to note that the number of the TB items corresponds with the TB size, and the number of PC cells is the sum of the PC grid cells for all the grid levels, and not only the finest one. Table 6.5 shows the storage requirements of the equivalent data

HM Algorithm		
Structure	Field	Size
PC cell	A	unsigned int (32 bits)
	L	unsigned short (16 bits)
	C	char (8 bits)
	I	char (8 bits)
	$Total$	64 bits
TB item	connectivity	unsigned short (16 bits)
	$Total$	16 bits

Table 6.4: HM algorithm data storage requirements.

EDP Algorithm		
Structure	Field	Size
CTB array	s	unsigned short (16 bits)
	e	unsigned short (16 bits)
	A	char (8 bits)
	B	char (8 bits)
	$shift\ [+1,+2]$	2× char (8 bits)
	$Total$	64 bits

Table 6.5: EDP algorithm data storage requirements.

structure in the EDP algorithm: CTB data array. Although smaller data types could be used to reduce space requirements for small models, for the sake of generality, the table shows fields sizes encoded with regular data types typically used in an efficient implementation supporting large models. Therefore, 32-bit *integers* are used for encoding absolute indices and 16-bit *integers* for relative indices, since related elements will never be so far to require a 32-bit *integer* to encode the offset between them. For the rest of the elements will suffice with a 8-bit *char*.

In the EDP algorithm, the CTB array, which can be arguably seen as an equivalent structure of the TB array, stores much more information and thus is clearly larger than the HM structure. However, there are two particularities of the EDP algorithm to consider before assuming it requires more memory storage. The first one is that, as it has been previously shown, the EDP algorithm introduces a minimum number of points in the TIN boundary, and thus the TB is consistently smaller in the EDP algorithm, sometimes by a large factor (see Tables 6.2 and 6.3). The second one is that the EDP algorithm does not need any equivalent structure to the PC list, since the CTB array contains all the required data to generate the adaptive tessellation during the visualization.

The size of the PC list is related to the number of PC cells in the grid. Although this number is typically be small in comparison to the size of the TB array at the finest LOD, the total number of PC cells in the model is not negligible. Moreover, when the resolution of the TIN mesh is only marginally better than the grid resolution (around 4-6 TIN vertices by grid cell, for example) the additional cost introduced by the HM algorithm as additional intersection vertices and PC cells data structures is much larger than the cost of the reduced CTB array.

The EDP and the HM algorithms have moderate storage requirements, although it greatly depends on the number of PC cells in the model, that is, in the shape of the TIN mesh. In typical cases, the EDP approach present a marginally larger but roughly equivalent memory cost for the data structures than the HM. For example, the *Coruña* model using the HM approach contains 17307 TB items, 4233 PC cells at the finest level, and a total number of 7950 PC cells adding up PC cells across all levels. The total required space for the HM data structures includes 2 bytes (16 bits) per item in the TB array and 8 bytes (64 bits) by item in the PC cell list. Therefore:

$$\text{HM data structures size} = 17307 \times 2 \text{ bytes} + 7950 \times 8 \text{ bytes} = 98214 \text{ bytes}$$

On the other hand, the required space used by the EDP approach only includes the CTB array, with a cost of 8 bytes (64 bytes) by item. In this case, the total space used would be:

$$\text{EDP data structures size} = 13361 \times 8 \text{ bytes} = 106888 \text{ bytes}$$

In this example, the EDP data structure is just a 9% larger than the equivalent data structures in the HM algorithm. However, this small difference is negligible compared to the much larger size of the meshes. The EDP algorithm uses the original TIN mesh of the *Coruña* model without any additional vertex, while the HM algorithms requires the introduction of 3946 new inter-cell vertices, as shown in Table 6.2. Assuming a minimum cost of at least 24 bytes per mesh vertex (using three *32-bit floats* for the position and three *32-bit floats* for the normal vector), the extra space cost required by the TIN mesh used in the HM algorithm would be:

$$\text{HM additional mesh size} = 3946 \times 24 \text{ bytes} = 94704 \text{ bytes}$$

And thus, the total memory cost of both approaches is:

$$\text{HM total memory cost} = 98214 + 94704 = 192918 \text{ bytes}$$

$$\text{EDP total memory cost} = 106888 \text{ bytes}$$

The HM algorithm presents a total memory cost about an 80% larger than the EDP algorithm. Therefore, the EDP algorithm is clearly superior to the HM algorithm, since the HM algorithm introduces a considerable number of vertices in the boundary of the TIN mesh, which becomes larger and heavier.

In summary, the EDP algorithm it is an effective method to generate high quality hybrid models without holes between the boundaries of the regular and irregular meshes. Compared to the the several implementations of the HM algorithm it becomes clear that uses less triangles to obtain an adaptive tessellation of the hybrid model. Additionally, the storage requirements of the EDP algorithm are lower than the HM algorithm, since the particular data structures are in both cases very light, but the meshes used in the HM algorithm are heavier due to the added vertices in their boundaries.

Chapter 7

Conclusions

With the increasing availability of geographic datasets, a large number of systems, in several application domains, demand for interactive manipulation and visualization of terrain models. The massive size of these models and their particular data representation scheme poses significant challenges for their interactive visualization. Hence, this field has been broadly studied in the literature and several techniques exist to deal with the representation and visualization of terrain models. Most of them using some kind of multiresolution scheme over the original data. Multiresolution methods organize the contents of a dataset at different resolutions or level-of-details, normally using some kind of hierarchical data structure, so as to select the most suitable representation at rendering time, depending on the quality requirements and performance constraints in the particular view conditions.

Interactive terrain rendering methods typically use a terrain representation based on irregular meshes of triangles or on regularly sampled height fields defining the surface elevation. Recent methods tend to focus on regular grids, since multiresolution approaches perform better with regular grids due to the efficient indexing of their regular data structures. However, grids are not efficient representing flat areas or regular surfaces with much redundancy, since the same sampling density is used all over the model. Furthermore, height fields can not model some geographic features as caves or overhangs, since they represent terrain surfaces as a parametrization of the terrain elevation over a base plane and not as surface in a 3D space. Models formed by irregular triangle meshes, on the other hand, are less efficient in terms of performance but can represent any geometry and also excel at removing redundancies in homogeneous areas of the surface.

A hybrid terrain model combining the strengths of both approaches can be a valuable addition to the field, but this possibility has been hardly considered in the literature until recently. The HM algorithm is one of the few proposals exploring this path, but in its original form, it is mostly a theoretical work without a real implementation. Hence, the objective of the research developed during this thesis was to develop enhanced methods for the real-time visualization and manipulation of hybrid terrain models, by either extending previous methods on the field or developing new approaches.

This dissertation has presented a detailed analysis of the original HM algorithm, arguably the first serious step in the interactive visualization of hybrid terrain models. The HM algorithm generates a hybrid model by the combination of a regular base grid with high-resolution TIN meshes, which are used to add fine-grained detail to specific areas of the terrain or to represent geophysical features impossible to model with a regular grid. An adaptive tessellation procedure is applied, locally, between the boundaries of the regular and irregular meshes to obtain a watertight hybrid model which does not contain holes or cracks. This tessellation is partially pre-computed and encoded in lightweight data lists that are consulted during the visualization to recreate the tessellation. Since the algorithm is compatible with a multiresolution rendering for the large grid, the partially encoded tessellation must be independent of the grid LOD used in rendering. Thus, the encoding is solely based on the boundary of the TIN. Furthermore, the HM algorithm works with different tessellation procedures and multiresolution rendering methods, which makes the algorithm a powerful and versatile approach.

Directly based on the strategy of the HM algorithm —a local adaptive tessellation process between the boundaries of the regular and irregular meshes— this thesis introduces two different proposals of a hybrid terrain model visualizer. Both proposals are capable of generate a coherent crack-free hybrid model due to the tessellation of the meshes boundaries part and the irregular TIN parts. The first proposal combines a standard polygon triangulation algorithm, the well-known Incremental Randomized Triangulation algorithm developed by Seidel [74], to generate the adaptive tessellation between the models. In this proposal, it is not necessary to precompute and encode any information since the whole tessellation between the grid cell corners and the TIN boundary vertices is computed during the visualization. The second proposal is a faithful implementation of the HM algorithm using the standard graphic pipeline. Therefore, the most demanding tasks of the tessellation process are precomputed during the preprocessing, and then this information is used to rebuild the tessellation triangles in the interactive visualization phase.

These methods have been tested and results indicate that both methods generate high quality water-tight meshes at interactive frame-rates. However, the proposal based on the faithful implementation of the HM algorithm is much faster than the proposal using a standard polygon triangulation strategy, proving that the HM algorithm strategy of preprocessing and encoding part of the tessellation is not only valid, but extremely efficient.

This thesis also introduces a new parallel method for hybrid terrain model rendering: the GPU-HM method. Based on the original HM algorithm, GPU-HM exploits the computational power and strong parallelism of GPUs by implementing a considerably optimized version of the HM algorithm using the geometry shader unit. This geometry shader based architecture brings two important consequences: first, its inclusion as a standard additional step in rendering engines pipelines is quite simple, provided that rendering codes is compatible with the now well established Shader Model 4 specification; secondly, it can be used with a wider range of GPU hardware as it does not require the more recent Tessellator unit, which is not present in low performance hardware, as some integrated CPUs or GPUs for mobile devices.

The method also follows a two phases approach similar to the HM algorithm, preprocessing and encoding the most expensive computations. In the interactive rendering phase, data lists are decoded and recreated on the GPU. This process is performed in parallel for multiple boundary cells at the same time, since the GPU features multiple processing units. Furthermore, the tessellation algorithm employed in the visualization for generating the GPU-HM method presents several important optimizations over the original proposal mainly due to the reorganization of the instructions in the geometry shader and the reduction of the data lists for attaining a more efficient processing. Finally, conventional strategies to enhance parallelism gainings in the GPU have been also applied, as loop unrolling or the elimination of bifurcations in the shader code.

The GPU-HM has been implemented and tested with several hybrid models with satisfactory results in terms of quality and performance. The implementation manages to render models of several millions of triangles, without geometric discontinuities or overlapping, at interactive frame rates. This is, to our knowledge, the first hybrid terrain model rendering algorithm implemented on the GPU.

The EHM algorithm is another important enhancement of the HM algorithm also developed in this thesis. This new method succeeds in adding view-dependent rendering to the generated hybrid model. Therefore, not only the regular grid mesh is rendered depending on the view conditions while the finely detailed TIN parts are statically rendered, as in the

previous approaches, but the irregular TIN meshes are also dynamically adapted to the scene. The EHM algorithm may work with any multiresolution TIN approach based on successive refinements of a hierarchical arrangement of triangles, but requires to perform a minor modification to constrain the simplification order of some TIN boundary vertices. This small requirement guarantees that the adaptive tessellation between the meshes boundaries can be performed, regardless of the simplification step of the TIN. The requirement is implemented by enforcing some preconditions during the construction of the view-dependent refinement data structures. Since all these computations take place in the preprocessing phase, no additional time-demanding tasks are performed during real-time visualization.

The benefits of this approach are clear, as the whole multiresolution hybrid model can be rendered without incurring in supplementary performance penalties to deal with the varying geometry of multiresolution models. This approach has been implemented and tested with satisfactory results regarding the high-quality of obtained meshes and the interactive performance shown by the algorithm, although mainly dependent on the performance of the multiresolution algorithms used.

Finally, this thesis also introduces the EDP algorithm, a completely new proposal for the interactive visualization of hybrid terrain models. This method, which is not based on the HM algorithm, presents several important enhancements to the previous approaches.

The most relevant feature of the EDP algorithm is the ability to perform the adaptive tessellation on the boundaries of the regular and irregular meshes without the inclusion of new additional points in the TIN boundary. HM-based algorithms treat the partially covered cells as perfectly closed polygons which can be tessellated independently. However, to fulfill this requirement, additional vertices were inserted in the intersections between the grid and the TIN boundary edges. Although the additional vertices do not modify the geometry of the TIN mesh, they do increment the complexity of the tessellation, with a larger number of triangles and vertices involved in the process. The EDP algorithm is able to join the regular and irregular meshes boundaries in a cell based way but uses a different basic primitive in the process —the external edge of the meshes— which presents a smaller complexity in the regular case. Moreover, it allows to leverage the spatial orientation of the edges to simplify the process. This proposal also deals with some particularly fussy cases by using efficient and simple techniques.

This thesis proposes several techniques for the interactive visualization and manipulation of hybrid terrain models. Since this research field has been barely studied in the literature,

plenty of opportunities exist to explore the problem and provide alternative solutions. Moreover, continuous advances in graphics hardware will provide further chances for the development of new techniques to exploit hardware features.

Regarding the research developed in this thesis, the most promising path to continue with the work consists in the extension of the EDP algorithm. One planned extension is to include a multiresolution rendering algorithm in the TIN mesh. Using an adequate method for handling the rendering of the irregular mesh, the whole hybrid model will be rendered depending on the conditions of the scene, as the EHM already manages to do, but preserving all the advantages of the EDP proposal. Another planned extension is to migrate the tessellation process of the boundaries to the GPU, to rely as much as possible in the GPU for the rendering process.

Another interesting line of research for the future is the study of the non geometric attributes of the hybrid model, to find appropriate methods to handle the visual attributes such as texture and color, and even non-visual attributes representing different kinds of data associated to the terrain, such as land use, temperature or other physical properties.

A long-term research line will certainly consider a change of paradigm, to include approaches capable of generating a visually pleasant representation of a hybrid model even if the model is not perfectly tessellated or contains crack. For situations where the geometry of the model does not need to be watertight, image domain techniques could be used in the fragment shader to mix the images from the regular and irregular meshes and mask the artifacts in the boundaries with a really low cost in performance.

Bibliography

[1] T. Akenine-Möller, E. Haines and N. Hoffman. *Real-Time Rendering*, 3rd ed. A. K. Peters, Ltd., 2008.

[2] M. Amor and M. Bóo. A New Architecture for Efficient Hybrid Representation of Terrains. *Journal of Systems Architecture*, 54(1-2):145–160, 2008.

[3] A. Asirvatham and H. Hoppe. Terrain Rendering using GPU-based Geometry Clipmaps. In *GPU Gems 2*, chap. 2, pp. 27–46. Addison-Wesley, 2005.

[4] K. Baumann, J. Döllner and K. Hinrichs. Integrated Multiresolution Geometry and Texture Models for Terrain Visualization. In *Proceedings of Joint Eurographics–IEEE TVCG Symp. on Visualization*, 2000, pp. 157–166.

[5] K. Baumann, J. Döllner, K. Hinrichs and O. Kersting. A Hybrid, Hierarchical Data Structure for Real-Time Terrain Visualization. In *Proceedings of the Int. Conf. on Computer Graphics (CGI '99)*, 1999, pp. 85–92.

[6] J. Blow. Terrain Rendering at High Levels of Detail, 2000. In *Proceedings of the 2000 Game Developers Conference*, 2000.

[7] D. Blythe. The Direct3D 10 System. *ACM Transactions on Graphics (TOG)*, 25(3): 724–734, 2006.

[8] M. Bóo and M. Amor. Dynamic Hybrid Terrain Representation based on convexity limits identification. *International Journal of Geographical Information Science*, 23(4):417–439, Apr. 2009.

[9] M. Bóo, M. Amor and J. Döllner. Unified Hybrid Terrain Representation based on Local Convexifications. *GeoInformatica*, 11(3):331–357, Feb. 2007.

[10] S. Burbeck. Applications Programming in Smalltalk-80: How to Use Model-View-Controller (MVC). Tech Report Smalltalk-80 v2. 5. ParcPlace. http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

[11] B. Carlisle. *Digital Elevation Model Quality and Uncertainty in DEM-Based Spatial Modelling*. Ph.D. dissertation, University of Greenwich, 2002.

[12] I. Castaño. Next-Generation Hardware Rendering of Displaced Subdivision Surfaces. In *Exhibition Tech. Session at the SIGGRAPH'08 Conf.*, 2008.

[13] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio and R. Scopigno. BDAM - Batched Dynamic Adaptive Meshes for High Performance Terrain Visualization. *Computer Graphics Forum*, 22(3):505–514, Sep. 2003.

[14] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio and R. Scopigno. Planet-Sized Batched Dynamic Adaptive Meshes (P-BDAM). In *Proceedings of the 14th IEEE Conf. on Visualization (VIS'03)*, 2003, pp. 147–155.

[15] J. H. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM*, 19(10):547–554, Oct. 1976.

[16] L. De Floriani, P. Magillo and E. Puppo. Efficient Implementation of Multi-Triangulations. In *Proceedings of the Conf. on Visualization (VIS'98)*, 1998, pp. 43–50.

[17] J. DeFerranti. Digital Elevation Models in Viewfinder Panoramas, 2005. http://www.viewfinderpanoramas.org.

[18] M. Duchaineau, L.M. Hwa and K. I. Joy. Real-Time Optimal Adaptation for Planetary Geometry and Texture: 4-8 Tile Hierarchies. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):355–68, 2005.

[19] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. PC. Miller, C. Aldrich and M.B. Mineev-Weinstein. Roaming Terrain: Real-Time Optimally Adapting Meshes. In *Proceedings of the 8th Conf. on Visualization (VIS '97)*, 1997, pp. 81–88.

[20] J. Dykes, A. M. MacEachren and M.-J. Kraak. *Exploring Geovisualization*. Elsevier, 2005.

[21] J. El-Sana and A. Varshney. Generalized Vew-Dependent Simplification. *Computer Graphics Forum*, (18): 83–94, 1999.

[22] I. S. Evans. *An Integrated System of Terrain Analysis and Slope Mapping.* Dept. of Geography, Univ. of Durham, Tech Report on Grant DA-ERO-591-73-G0040, 1979.

[23] I. S. Evans. General Geomorphometry, Derivatives of Altitude and Descriptive Statistics. In *Spatial analysis in Geomorphology*, chap. 2, pp. 17–90, Methuen Co., 1972.

[24] A. Fournier and D. Y. Montuno. Triangulating Simple Polygons and Equivalent Problems. *ACM Transactions on Graphics*, 3(2):153–174, Apr. 1984.

[25] M. Garland. Multiresolution Modeling: Survey & Future Opportunities. In *Proceedings of Eurographics*, State of the Art Report, pp. 111–131, 1999.

[26] G. Gröger, T.H. Kolbe, A. Czerwinski and C. Nagel. *City Geography Markup Language (CityGML)*, OpenGIS Encoding Standard, 2008. http://www.citygml.org/.

[27] A. Gueziec. Surface Simplification with Variable Tolerance. *2nd Annual Int. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, 1995, pp. 132–139.

[28] A. Guéziec. Surface Simplification inside a Tolerance Volume. IBM Research Division. T.J. Watson Research Center, Technical Report, 1997.

[29] T. Gurung and J. Rossignac. SOT: Compact Representation for Triangle and Tetrahedral Meshes. School of Interactive Computing. Georgia Institute of Technology, Technical report, 2010.

[30] H. Hoppe. View-Dependent Refinement of Progressive Meshes. In *Proceedings of the 24th annual Conf. on Computer Graphics and Interactive Techniques ()SIGGRAPH '97)*, 1997, pp. 189–198.

[31] H. Hoppe. Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. In *Proceedings of the Conf. on Visualization (VIS '98)*, 1998, pp. 35–42.

[32] H. Hoppe. Progressive Meshes. In *Proceedings SIGGRAPH '96*, 1996 pp. 99–108.

[33] L. Hu, P. V. Sander and H. Hoppe. Parallel View-Dependent Refinement of Progressive Meshes. In *Proceedings of the 2009 Symp. on Interactive 3D Graphics and Games*, 2009, pp. 169–176.

[34] Infraestructura de Datos Espaciales de España (IDEE). Digital Elevation Models, 2002. http://www.idee.es.

[35] J. Kim and S. Lee. Truly Selective Refinement of Progressive Meshes. In *Proceedings of Graphics Interface*, 2001, pp. 101–110.

[36] D. Kirk. NVidia CUDA Software and GPU Parallel Computing Architecture. In *Proceedings of the 6th Int. Symp. on Memory Management*, 2007, vol. 21, pp. 103–104.

[37] M. Kraus and T. Ertl. Cell-Projection of Cyclic Meshes. In *Proceedings of the Conf. on Visualization (VIS '01)*, 2001, pp. 215–559.

[38] M. Kraus and T. Ertl. Simplification of Nonconvex Tetrahedral Meshes. In *Hierarchical and Geometrical Methods in Scientific Visualization*, pp. 185–196. Springer-Verlag, 2000.

[39] R. Lario, R. Pajarola and F. Tirado. Hyperblock-QuadTIN: Hyper-Block Quadtree based Triangulated Irregular Networks. In *Proceedings of IASTED Visualization, Imaging and Image Processing Conference (VIIP '03)*, 2003, pp. 733–738.

[40] D. T. Lee and B. J. Schachter. Two Algorithms for Constructing a Delaunay Triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, Jun. 1980.

[41] J. Levenberg. Fast View-Dependent Level-of-Detail Rendering using Cached Geometry. In *Proceedings of the Conf. on Visualization (VIS '02)*, 2002, pp. 259 – 265.

[42] P. Lindstrom, D. Koller and L. F. Hodges. Level-of-Detail Management for Real-Time Rendering of Phototextured Terrain. Georgia Institute of Technology, Technical Report, 1995.

[43] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust and G. A. Turner. Real-Time, Continuous Level of Detail Rendering of height fields. In *Proceedings of SIGGRAPH '96 Conf.*, 1996, pp. 109–118.

[44] P. Lindstrom and V. Pascucci. Visualization of Large Terrains Made Easy. In *Proceedings of the Conf. on Visualization (VIS '01)*, 2001, pp. 363–371.

[45] P. Lindstrom and V. Pascucci. Terrain Simplification Simplified: A General Framework for View-Dependent Out-of-Core Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254, 2002.

[46] F. Losasso and H. Hoppe. Geometry Clipmaps: Terrain Rendering using Nested Regular Grids. In *Proceedings of the Int. Conf. on Computer Graphics and InteractiveTechniques*, 2004, pp. 769–776.

[47] D. Luebke. A Developer's Survey of Polygonal Simplification Algorithms. *Computer Graphics and Applications*, (Jun.):24–35, 2001.

[48] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson and R. Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2002.

[49] D. M. Mark. Geomorphometric Parameters: a Review and Evaluation. *Geografiska Annaler. Series A. Physical Geography*, pp. 165–177, 1975.

[50] W. R. Mark, R. S. Glanville, K. Akeley and M. J. Kilgard. Cg: A System for Programming Graphics Hardware in a C-like Language. In *ACM Transactions on Graphics*, 22: 896–907, 2003.

[51] M. Natali, E. M. Lidal, J. Parulek, I. Viola and D. Patel. Modeling Terrains and Subsurface Geology. In *EuroGraphics 2013 State of the Art Reports (STARs)*, pp. 155–173, 2013.

[52] T. Ni, I. Castaño and J. Peters. Efficient Substitutes for Subdivision Surfaces. In *SIGGRAPH '09 Courses*, 2009, Article no. 13.

[53] OpenGL.org. OpenGL Geometry Shader 4 Extension Specification, 2008. http://www.opengl.org/registry/specs/ARB/geometry_shader4.txt.

[54] OpenGL.org. OpenGL Texture Buffer Object Extension Specification, 2009. http://www.opengl.org/registry/specs/EXT/texture_buffer_object.txt.

[55] J. O'Rourke. *Computational Geometry in C*, 2nd ed. Cambrige University Press, 1998.

[56] R. Pajarola and C. DeCoro. Efficient Implementation of Real-Time View-Dependent Multiresolution Meshing. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):353–68, 2004.

[57] R. Pajarola and E. Gobbetti. Survey of Semi-Regular Multiresolution Models for Interactive Terrain Rendering. *The Visual Computer*, 23(8):583–605, 2007.

[58] R. Pajarola. Large Scale Terrain Visualization using the Restricted Quadree Triangulation. In *Proceedings of the conference on Visualization (VIS '98)*, pp. 19–26, 1998.

[59] R. Pajarola, M. Antonijuan and R. Lario. QuadTIN: Quadtree Based Triangulated Irregular Networks. In *Proceedings of the Conference on Visualization (VIS '02)*, pp. 395–402, 2002.

[60] E. G. Paredes, C. Lema, M. Amor and M. Bóo. Hybrid Terrain Visualization based on Local Tessellations. In *Proceedings of the Int. Conf. on Computer Graphics Theory and Applications (GRAPP '09)*, pp. 64–69, 2009.

[61] E. G. Paredes, M. Bóo, M. Amor, J. D. Bruguera and J. Döllner. Extended Hybrid Meshing Algorithm for Multiresolution Terrain Models. *International Journal of Geographical Information Science*, 26(5):771–793, 2012.

[62] E. G. Paredes, M. Bóo, M. Amor, J. Döllner and J. D. Bruguera. GPU-Based Visualization of Hybrid Terrain Models. In *Int. Conf. on Computer Graphics Theory and Applications (GRAPP 2012)*, pp. 254–259, 2012.

[63] E. G. Paredes, M. Amor, M. Bóo, J. D. Bruguera and J. Döllner. Hybrid Terrain Rendering based on External Edge Primitive. *IEEE Transactions on Visualization and Computer Graphics (To be submitted)*.

[64] S. C. Park and H. Shin. Polygonal Chain Intersection. *Computers & Graphics*, 26(2):341–350, Apr. 2002.

[65] S. Patidar, S. Bhattacharjee, J.-M. Singh and P. J. Narayanan. Exploiting the Shader Model 4.0 Architecture. Technical report, Center for Visual Information Technology, IIIT Hyderabad, 2007.

[66] A. A. Pomeranz. *ROAM using surface triangle clusters (RUSTiC)*. Ph.D. dissertation, University of California, 2000.

[67] E. Puppo. Variable Resolution Triangulations. *Computational Geometry*, 11(3-4):219–238, Dec. 1998.

[68] M. Reddy, Y. Leclerc, L. Iverson and N. Bletter. TerraVision II: Visualizing Massive Terrain Databases in VRML. *IEEE Computer Graphics and Applications*, 19(2):30–38, 1999.

[69] J .Rossignac. 3D Compression made Simple: Edgebreaker with Zip&Wrap on a Corner-Table. In *Proceedings of the Int. Conf. on Shape Modeling and Applications (SMI '01)*, pp. 278–283, 2001.

[70] J .Rossignac. Surface Simplification and 3D Geometry Compression. In *Handbook of Discrete and Computational Geometry (Discrete Mathematics and Its Applications), 3rd ed.* Chapman and Hall, 2004.

[71] R. J. Rost, B. Licea-Kane, D. Ginsburg, J. M. Kessenich, B. Lichtenbelt, H. Malan and M. Weiblen. *OpenGL Shading Language*, 3rd ed. Addison-Wesley Professional, 2009.

[72] S. Röttger, W. Heidrich, P. Slusallek and H. P. Seidel. Real-Time Generation of Continuous Levels of Detail for Height Fields. In *Proceedings of the Int. Conf. on Central Europe Computer Graphics, Visualization and Computer Vision (WSCG '98)*, pp. 315–322, 1998.

[73] J. Schneider and R. Westermann. GPU-Friendly High-Quality Terrain Rendering. In *Proceedings of the 14th Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision 2006 (WSCG '06)*, pp. 49–56, 2006.

[74] R. Seidel. A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and for Triangulating Polygons. *Computational Geometry*, 1(1):51–64, 1991.

[75] J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. *Lecture Notes in Computer Science*, 1148:203–222, 1996.

[76] R. Sivan and H. Samet. Algorithms for Constructing Quadtree Surface Maps. In *Proceedings of the 5th Int. Symposium on Spatial Data Handling*, pp. 361–270, 1992.

[77] S. St-Laurent. *The Complete Effect and HLSL Guide*. Paradoxal Press, 2005.

[78] J. E. Stone, D. Gohara and G. Shi. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in science engineering*, 12(3):66–73, May 2010.

[79] G. Turk and B. Mullins. Large Geometric Models Archive at Georgia Institute of Technology, 1998. http://www.cc.gatech.edu/projects/large_models/.

[80] T. Ulrich. Continuous LOD Terrain Meshing using Adaptive Quadtrees, Feb. 2000. Gamasutra article. http://www.gamasutra.com

[81] P. van Oosterom, S. Zlatanova, F. Penninga and E. Fendel. *Advances in 3D Geoinformation Systems*. Lecture Notes in Geoinformation and Cartography. Springer Berlin / Heidelberg, 2008.

[82] L. Velho and J. Gomes. Variable Resolution 4-k Meshes: Concepts and Applications. *Computer Graphics Forum*, 19(4):195–212, 2000.

[83] R. Weibel and M. Heller. *Digital Terrain Modelling*. Oxford Univ. Press, 1993.

[84] R. Westerteiger, T. Compton, T. Bernadin, E. Cowgill, K. Gwinner, B. Hamann, A. Gerndt and H. Hagen. Interactive Retro-Deformation of Terrain for Reconstructing 3D Fault Displacements. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2208–2215, Dec. 2012.

[85] J. D. Wood. *The Geomorphological Characterisation of Digital Elevation Models*. Ph.D. dissertation, University of Leicester, UK, 1996.

[86] R. S. Wright, N. Haemel, G. Sellers and B. Lipchak. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addison-Wesley Professional, 5th ed., 2010.

[87] Q. Wu and H. Xu. An Approach to Computer Modeling and Visualization of Geological Faults in 3D. *Computers & Geosciences*, 29(4):503–509, May 2003.

[88] B. Yang, W. Shi and Q. Li. An Integrated TIN and Grid Method for Constructing Multi-Resolution Digital Terrain Models. *International Journal of Geographical Information Science*, 19(10):1019–1038, Nov. 2005.