Visualización 3D de Terrenos Multiresolución basada en Shader Model 5

Lucas Borrelli[1], María J. Abásolo[1,2]

[1] Universidad Nacional de La Plata, Facultad de Informática
calle 50 y 120, 1900 La Plata, Argentina
[2] Comisión de Investigaciones Científicas de la Pcia. de Bs.As.
febolucas@gmail.com, mjabasolo@lidi.info.unlp.edu.ar

Resumen. Este artículo presenta un nuevo algoritmo de visualización de terrenos multirresolución que emplea los últimos avances de la GPU. En su diseño se aplica un criterio de selección de nivel de detalle que considera la percepción que tendrá el usuario de la rugosidad particular de cada zona del terreno. Además, se incorporan las capacidades de teselado por hardware de la GPU correspondientes a *Shader Model 5*, y se implementa un mecanismo de *geomorphing* creando transiciones suavizadas entre distintos niveles de detalle. La técnica realiza una representación multirresolución eficiente y escalable. Los resultados obtenidos posibilitan su utilización en aplicaciones interactivas que requieren de la visualización de terrenos en tiempo real para aplicaciones 3D.

Keywords: Terrenos, Gráficos 3D, Multirresolución, GPU, Teselado.

1 Introducción

Los algoritmos relacionados con la representación y visualización de terrenos se encargan de administrar los datos de modelos de elevación para representar escenarios basados en terrenos, y son importantes en un gran número de aplicaciones, como ser simuladores de vuelo o de entrenamiento, sistemas de información geográfica, realidad virtual, videojuegos, etc.

La principal dificultad a resolver por los algoritmos de visualización de terrenos es obtener una visualización eficiente en tiempo real y lograr terrenos que luzcan realistas. Por esta razón dichos algoritmos utilizan el concepto de multirresolución para lograr una adaptación dinámica de la complejidad del modelo visualizado generando representaciones simplificadas o de menor nivel de detalle o LOD (*Level Of Detail*).

Los últimos avances en las capacidades de las unidades de procesamiento gráfico o GPU (*Graphics Processing Unit*) han introducido la posibilidad de realizar teselados de geometría directamente por hardware y con una mínima intervención por parte de la CPU. Estos avances son muy relevantes respecto de la multirresolución, ya que proponen el desarrollo de sistemas más flexibles y eficientes para la generación de representaciones simplificadas en tiempo real.

Este artículo presenta el diseño de un algoritmo de visualización 3D de terrenos multirresolución basado en la última características de la aceleración de gráficos por hardware, y se muestran los resultados de las pruebas realizadas sobre el mismo. El objetivo ha sido lograr un algoritmo eficiente en el uso de los recursos de procesamiento y con buenos resultados visuales.

El resto del artículo se organiza de la siguiente forma: la sección 2 presenta algunos de los algoritmos multirresolución más relevantes de la bibliografía, la sección 3 detalla los aspectos de diseño del algoritmo desarrollado, y la sección 4 muestra los resultados de las pruebas realizadas sobre el mismo. Finalmente la sección 5 presenta las conclusiones del caso.

2 Trabajos anteriores

Con el advenimiento de la GPU a comienzos de 2000, una serie de nuevos algoritmos de visualización de terrenos han sido desarrollados principalmente para aprovechar al máximo las capacidades de cómputo de ese hardware. A continuación se mencionan las principales características de los algoritmos más relevantes. Una lista exhaustiva incluyendo algoritmos previos a la GPU, puede consultarse en [7].

En primer lugar, el algoritmo *Geometrical Mipmapping* [2] modela el terreno mediante una grilla regular de vértices la cual es dividida en bloques adyacentes para permitir que regiones del terreno sean visualizadas en distinto detalle. Para esto, cada bloque es simplificado realizando un submuestreo regular formando una secuencia de mallas de resolución decreciente denominadas *geomipmaps*. Así un *geomipmap* de nivel 0 corresponde a la mayor resolución, el nivel 1 posee la mitad y así sucesivamente hasta alcanzar el nivel de menor detalle de sólo 2 triángulos.

Otro algoritmo pionero es *Chunked LOD* [9], el cual se basa en una estructura de tipo *quadtree*, es decir un árbol cuaternario utilizado para subdivisión espacial. En este trabajo cada nodo del *quadtree* posee su propia malla denominada *chunk* para representar la zona del terreno que abarca. La malla de cada *chunk* es construida en una etapa de preprocesamiento utilizando un esquema de triangulación no regular, la cual es cargada en memoria de GPU a medida que es necesaria su utilización.

En *Geometry Clipmaps* [6], el terreno es representado mediante un conjunto de grillas regulares anidadas que se centran alrededor de la posición 2D del punto de observación sobre el plano del terreno. Cada grilla posee un nivel de detalle diferente, disminuyendo la resolución hacia las grillas externas. Los datos de elevación que conforman las grillas son almacenados en memoria de la GPU, de modo que cuando el punto de observación se traslada, los vértices deben ser actualizados de manera que las grillas se mantengan siempre centradas respecto del observador.

Por último, el algoritmo *CDLOD* [8] se estructura en un *quadtree* de igual modo que *Chunked LOD*, pero a diferencia de éste utiliza mallados regulares. Este algoritmo renderiza el terreno desde un *heightmap* almacenado en GPU gracias a las características de acceso a texturas presentes a partir del *Shader Model 3* [3]. Además, en el mismo se incorpora una técnica simple para la selección de nivel de detalle basada en un vector de distancias que a su vez posibilita realizar una transición suave entre LODs en la que no es necesario evitar grietas o T-junctions.

3 Descripción del algoritmo

3.1 Representación del terreno multirresolución

Para representar el terreno se conforma una grilla de dimensión 2n+1 de lado. La grilla se compone de vértices separados a intervalos regulares. La información de altura se toma de un *heightmap*, es decir de una imagen en tonos de grises que codifica la intensidad de cada píxel como la altura en ese punto. Un árbol cuaternario o *quadtree* de bloques organiza dicha grilla. El *quadtree* se utiliza como una estructura de subdivisión 2D de la superficie del terreno como se ilustra en la figura 1.

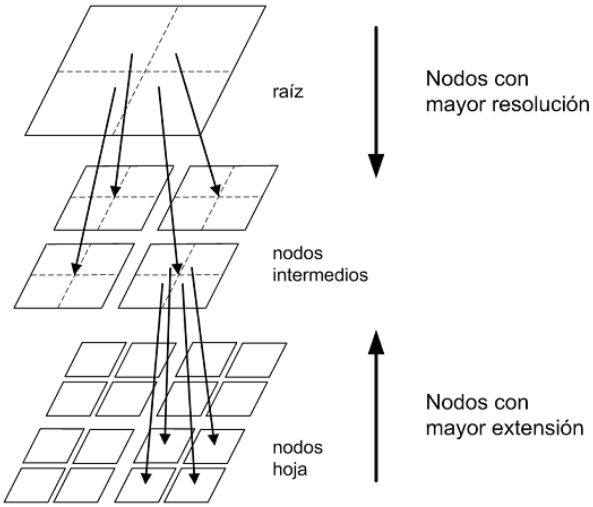

Fig. 1. Representación del terreno mediante *quadtree* de bloques.

La información de altura se modela en todos los nodos del *quadtree*. Así, los nodos del árbol representan una porción cuadrada del terreno a distinto nivel de detalle. El modelo multirresolución se logra aplicando distinto nivel de detalle según la altura de los nodos en el quadtree. La raíz abarca toda la superficie del mismo, mientras que el resto de los nodos abarcan un cuarto de la extensión que abarca su correspondiente nodo padre.

3.2 Selección de nivel de detalle

Algoritmos como [8] y [6] utilizan la distancia al punto de observación como único criterio de selección de LOD. Estos enfoques tienden a maximizar la percepción por parte del usuario de cambios geométricos. Para este trabajo se aplicó una métrica que

emplea el error de aproximación geométrica proyectado en espacio de imagen. Esto implica no sólo emplear la distancia como criterio de selección de detalle, sino que permite considerar la rugosidad propia de cada zona del terreno, asignando además mayor detalle a zonas más rugosas (picos o cimas) y menor detalle a zonas llanas.

El cálculo de la métrica consiste en dos etapas. Primero debe determinarse el error de aproximación geométrica δ para cada nodo del *quadtree*. Luego, el error δ es proyectado en espacio de imagen obteniéndose el error visual ε perceptible por usuario.

Como el error de aproximación geométrica δ es la diferencia de altura (en espacio de objeto) entre los vértices de dos niveles de detalle distintos [5], el cálculo de δ requiere el recorrido y análisis de las muestras del *heightmap*. El costo de este computo no permite ser llevado a cabo durante la visualización de tiempo real, por lo que se realiza en una etapa de preprocesamiento. Los δ_i obtenidos para cada nodo i del árbol son almenados en disco, y cargados posteriormente en memoria para su utilización.

Al momento de la visualización, un proceso de refinamiento recursivo es llevado a cabo para identificar el nivel de detalle para cada zona del terreno. Esto es, el *quadtree* es recorrido de forma descendente comenzando desde la raíz. Descendiendo por el mismo, se recolectan los nodos que satisfacen el nivel de detalle deseado. El error δ_i de cada nodo i visitado es entonces proyectado en espacio de imagen mediante proyección perspectiva como indica la ecuación 1: siendo h_{img} la altura en espacio de imagen del plano de proyección, d la distancia desde el punto de observación al bloque de terreno, y *fov* el ángulo del campo visual con el cual se realiza la proyección mediante la cámara virtual de la escena.

$$\varepsilon = \delta \frac{h_{img}}{2d\ tan\left(\frac{fov}{2}\right)} \quad \textbf{(1)}$$

Esta métrica asume que el punto de observación se encuentra sobre el plano horizontal y la dirección de visualización siempre es paralela al mismo. Esta simplificación del cálculo trae un beneficio adicional. Al ser independiente de la dirección de visualización, el nivel de detalle del terreno se mantiene estable cuando el usuario rota la cámara, variando sólo cuando el usuario traslada su posición, lo que permite minimizar la percepción de posibles cambios en la geometría.

Finalmente, ε es comparado contra un umbral de tolerancia τ configurable. Durante el recorrido del quadtree, si el error visual ε_i de un nodo i supera la tolerancia permitida τ, es necesario mayor nivel de detalle en esa zona del terreno por lo que el nodo debe ser refinado. En caso contrario se ha encontrado un nodo con un nivel de detalle aceptable para ser visualizado.

3.3 Renderización

La renderización se lleva a cabo utilizando las características de teselado de primitivas de *Shader Model 5* [10]. Una misma primitiva tipo *quad-patch* es utilizada

para generar en tiempo real las mallas poligonales correspondientes a la superficie del terreno para cada nodo seleccionado en el proceso de refinamiento y recorrido del *quadtree*.

Los vértices de cada malla son trasladados mediante *shaders*. Un desplazamiento vertical respecto del plano del terreno se realiza mediante el acceso a las muestras de altura almacenadas en un recurso de tipo textura que contiene el *heightmap*. Horizontalmente la posición de los vértices también es modificada según un valor configurable de separación entre muestras, y trasladados según la posición relativa de cada nodo en el *quadtree*. De esta manera puede aprovecharse al máximo todos los recursos de hardware, a la vez que se minimiza la intervención de la CPU en la renderización.

3.4 Prevención de grietas entre bloques

Un inconveniente a resolver por las técnicas basadas en bloques, es la disyunción de vértices entre bloques de distinto nivel de detalle. Estas discontinuidades o "grietas" pueden manifestarse por no emplear los mismos vértices a lo largo del borde que dos bloques comparten entre sí.

En [2] y [4], la solución propuesta ha sido realizar modificaciones a la topología del mallado mediante la reconexión de los vértices en el borde de uno de dos bloques de geometría adyacentes. Por otro lado, con la introducción de la última versión de *Shader Model* (SM5), hoy también es posible realizar la adaptación de bordes por GPU mediante los denominados factores de teselado externos. Sin embargo, esta técnica requiere conocer el nivel de detalle de los bloques vecinos de cada bloque. Esto implica realizar búsquedas adicionales por el *quadtree*, consumiendo ciclos adicionales de CPU y volviendo más complejo y menos performante el algoritmo de visualización.

En este trabajo se ha utilizado geometría adicional alrededor de cada bloque de terreno como en [9]. Esta geometría adicional es denomina “polleras” (*skirts*), ya que consisten en triángulos verticales que comienzan en los bordes de un bloque y se prolongan hacia abajo por una determinada extensión. Las polleras constituyen un método simple y efectivo de resolver la posible aparición de grietas, y son similares a los “triángulos de área cero” utilizados en [6]. Además, pueden utilizarse también primitivas *quad-patch* verticales para generar el mallado de las mismas, permitiendo una implementación limpia y por GPU al ser incluidas en el mismo proceso de renderización de bloques mediante *shaders*.

3.4 Geomorphing

Otro aspecto relacionado a la representación multirresolución de terrenos es el efecto denominado *popping*. Este consiste en la visualización de cambios repentinos de altura o *pops* en la superficie del terreno, que se producen al modificar el mallado de una zona, o al reemplazar un bloque de terreno por otro de distinto nivel de detalle. Con el objetivo de mejorar la performance a la vez de minimizar posibles apariciones de *pops*, se consideró aplicar la técnica denominada *geomorphing*, esto es, una

animación o transición suavizada entre las representaciones geométricas de dos niveles de detalle distintos.

Nuevamente, se han aprovechado las características de *Shader Model 5* para poder implementar esta técnica mediante el uso de los factores de teselado fraccionales. Su utilización ha permitido implementar transiciones suavizadas sin requerir el desarrollo de *shaders* específicos como en modelos de *shader* anteriores [11].

En la figura 2 puede observarse el mallado generado por el algoritmo utilizando distintos colores. La técnica de *geomorphing* es representada mediante la mezcla de los colores rojo y verde. El mallado de los bloques de terreno es coloreado hacia el rojo indicando un nodo a punto de subdividirse, y hacia el verde para indicar su colapso con nodos hermanos y reemplazo por el nodo padre.

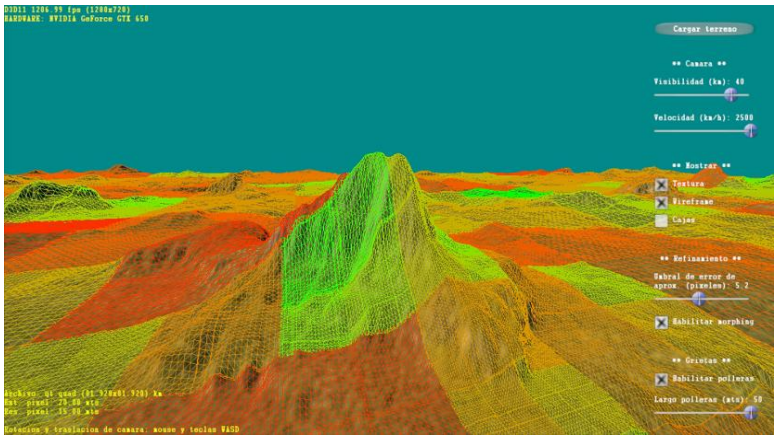

Fig. 2. Técnica de *geomorphing* representada mediante la mezcla de colores rojo y verde.

El *geomorphing* se realiza entre bloques de terreno que poseen niveles de detalle consecutivos, para lo cual se requiere conocer dos distancias para cada nodo del *quadtree*. En primer lugar es necesaria una distancia *s* que indica a qué distancia un nodo debería ser subdividido en sus cuatro nodos hijo de mayor detalle. A su vez, es necesaria la distancia *u* a la que un mismo nodo debe ser unificado junto a sus hermanos y reemplazados por el nodo padre.

$$f = \frac{d - s}{u - s} \tag{2}$$

La ecuación 2 (siendo *d* la distancia entre el centro del bloque y el punto de observación), es utilizada como peso para efectuar una interpolación lineal entre dos factores de teselado enteros. La relación *f* permite aplicar una transición suavizada entre dos niveles de detalle consecutivos, es decir entre un nodo del *quadtree* y sus nodos hijo. La técnica de *geomorphing* se logra aplicando un factor de teselado

fraccional (*Shader Model 5*) resultado de la interpolación. Cuando el punto de observación se traslada (variación de *d*), la transición entre el teselado del bloque actual y el teselado de los bloques de mayor detalle se realiza de forma incremental. Esto brinda una transición suavizada entre teselados para cualquier valor de *d*. Cuando *d* alcanza a *s*, el bloque actual es reemplazado por sus 4 bloques hijo de mayor detalle y el proceso de interpolación vuelve a comenzar sobre otro nivel del *quadtree*.

$$\tau = \varepsilon \quad \Rightarrow \quad \tau = \delta \frac{h_{img}}{2d\ tan\left(\frac{fov}{2}\right)} \quad \Rightarrow \quad d = \delta \frac{h_{img}}{2\tau\ tan\left(\frac{fov}{2}\right)} = s \tag{3}$$

Por último, es necesario calcular las distancias *s* y *u*. En primer lugar, como se ha mencionado la distancia *s* de un nodo es la distancia a la que el nodo es subdividido en sus cuatro hijos, y esa es la distancia a la que ε se equipara con τ, como se indica en la ecuación 3. Por otro lado, dado que la distancia a la que un nodo se divide es la misma a la que sus hijos se unifican, la distancia *u* es igual a la distancia *s* del padre del nodo actual. Así en el recorrido recursivo del quadtree, sólo es necesario enviar por parámetro la distancia *s* hacia los nodos hijo cuando un nodo es subdividido. Por último, la raíz carece de distancia *u*, la cual es considerada como 2*s*.

4 Resultados obtenidos

Para caracterizar el comportamiento del algoritmo, una serie de tests fueron realizados utilizando dos conjuntos de datos de zonas naturales conocidos como *Puget Sound* y *Hawaii*, y un tercer terreno *ad hoc* generado artificialmente, todos de 4097x4097 muestras de lado con separación de 10 metros. Los mapas de altura o *heightmaps* corresponden a imágenes PNG en escala de grises de 8 bits con una resolución vertical de 12.8 metros. Se obtuvieron terrenos de 40.96 x 40.96 km^2 de área, generando cada uno un mallado con resolución total de 33.6 millones de triángulos.

Para comparar performance respecto de la calidad de imagen obtenida, los tests fueron repetidos en cada terreno modificando el umbral de error visual τ = {1.0, 2.5, 5.0, 7.5, 10.0}. Los resultados se resumen en las tabla 1, 2 y 3, para *Puget Sound*, *Hawaii* y el terreno artificial respectivamente. Los valores de duración por cuadro se se estandarizaron en unidades de cuadros por segundo o FPS (*Frames Per Second*).

La implementación fue realizada utilizando la librería gráfica DirectX 11 [12]. El hardware utilizado en los tests fue una PC con procesador Intel® Core™ i7-3770 de 3.40GHz, memoria RAM de 8GB DDR3, y una placa gráfica NVIDIA GeForce GTX 650 con 2GB DDR5 de memoria de video. La resolución de pantalla fue de 1920x1080 píxeles (HD) y en pantalla completa.

Tabla 1. Resultados obtenidos con *Puget Sound*.

Umbral	#bloques (#tris) / seg.	FPS Mínimo	FPS Media	FPS Máximo
1.0	145K (672M)	183 (-29%)	257	363 (+41%)
2.5	108K (523M)	482 (-37%)	762	1106 (+45%)
5.0	61K (315M)	1057 (-21%)	1343	1599 (+19%)
7.5	40K (201M)	1356 (-17%)	1634	1878 (+15%)
10.0	28K (140M)	1598 (-12%)	1810	2000 (+10%)

Tabla 2. Resultados obtenidos con *Hawaii*.

Umbral	#bloques (#tris) / seg.	FPS Mínimo	FPS Media	FPS Máximo
1.0	139K (688M)	269 (-20%)	337	544 (+61%)
2.5	107K (535M)	634 (-16%)	756	1058 (+40%)
5.0	72K (346M)	1047 (-19%)	1296	1618 (+25%)
7.5	44K (225M)	1308 (-19%)	1608	2119 (+32%)
10.0	30K (164M)	1472 (-17%)	1779	2339 (+31%)

Tabla 3. Resultados obtenidos con el terreno artificial.

Umbral	#bloques (#tris) / seg.	FPS Mínimo	FPS Media	FPS Máximo
1.0	140K (694M)	201 (-6%)	214	227 (+6%)
2.5	124K (618M)	457 (-8%)	495	538 (+9%)
5.0	96K (481M)	872 (-7%)	934	1022 (+9%)
7.5	73K (367M)	1130 (-10%)	1249	1308 (+5%)
10.0	56K (284M)	1355 (-6%)	1449	1540 (+6%)

El resultado, es una variación de performance debido a la cantidad de geometría utilizada para mantener la calidad seleccionada. En todos los casos se observa que a medida que se relaja el umbral de error, es necesario un menor número de bloques para aproximar la calidad seleccionada, y por tanto el número de cuadros que es posible generar por segundo aumenta respectivamente. Además la media de FPS no registra grandes variaciones respecto de los extremos mínimo y máximo en cada test. Los tests que registran menor variación corresponden al terreno artificial debido principalmente a la alta frecuencia de zonas rugosas de su topología. Por la misma razón, también registra menores medias respecto de los otros terrenos.

Los mejores resultados son aquellos que logran un balance entre calidad visual y performance. Partiendo de un umbral de 1.0 y hasta 2.5 se registran mejoras en la performance y sin pérdida de calidad visual aparente. A partir de valores de 5.0 y mayores la calidad visual puede no ser muy apropiada. Un valor de umbral en el

rango 2.5 a 5.0 permite lograr el mejor balance entre calidad visual y performance. En este rango se han obtenido performances mínimas entre 500 y 1000 FPS. Sin considerar otros aspectos de las aplicación de visualización de terrenos (trabajo en red, simulaciones físicas, etc.), los resultados obtenidos superan ampliamente los 60 FPS necesarios para obtener una secuencia de imágenes fluida y de buena interactividad [1]. La performance lograda es muy satisfactoria.

5 Conclusión

El algoritmo de visualización 3D de terrenos multirresolución presentado se basa en un *quadtree*. Esta estructura regular ha permitido aprovechar las últimas características de la GPU (SM5), para realizar una representación multirresolución eficiente y escalable. Mediante las funcionalidades de teselado, fue posible controlar la resolución por hardware, facilitando a su vez la implementación de otras características como por ejemplo la técnica de *geomorphing*.

En su diseño se abordaron los inconvenientes provenientes de considerar como único criterio de selección de nivel de detalle la distancia al punto de observación. Se incorporó entonces un criterio más apropiado que considera la percepción que el usuario tendrá de las particularidades de cada zona del terreno. De esta manera es posible tener menos detalle en zonas llanas y tener mayor detalle en las zonas que poseen mayor rugosidad, obteniéndose una mejor distribución de triángulos para aproximar la superficie del terreno, logrando así mayor calidad visual disminuyendo la percepción de cambios de resolución.

Ajustando un parámetro de umbral de error, el algoritmo permite controlar la calidad visual/performance dependiendo de la necesidad o aplicación particular donde se desee implementar el algoritmo. Los tests de performance dieron como resultado un muy buen rendimiento de la técnica de visualización. El bajo consumo de recursos de procesamiento de CPU y GPU posibilita su utilización en aplicaciones interactivas que requieran de la visualización en tiempo real de modelos de terrenos para escenarios virtuales.

Referencias

1. Akenine-Möller, T., Haines, E., Hoffman, N.: Real-Time Rendering. 3ra edición, A.K.Peters Ltd. (2008)
2. De Boer, W.H.: Fast Terrain Rendering Using Geometrical MipMapping. World Wide Web, Octubre de 2000. Disponible en http://www.flipcode.com/archives/ article_geomipmaps.pdf.
3. Gerasimov, P., Fernando, R., Green, S.: Shader Model 3.0: Using Vertex Textures. NVIDIA Corporation Whitepaper (2004). Disponible en ftp://download.nvidia.com/developer/ Papers/2004/Vertex_Textures/Vertex_Textures.pdf.
4. Larsen, B.D., Christensen, N.J.: Real-time Terrain Rendering using Smooth Hardware Optimized Level of Detail. Journal of WSCG, 11(2), pp. 282–9 (2003)
5. Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L.F., Faust, N., Turner, G.A.: Real-Time, Continuous Level of Detail Rendering of Height Fields. Proceedings ACM SIGGRAPH ’96 Conference on Computer Graphics, pp. 109–118 (1996)

6. Losasso, F., Hoppe, H.: Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids. ACM Transactions on Graphics (SIGGRAPH), 23(3), pp. 769-776 (2004)
7. Pajarola, R., Gobbetti, E.: Survey on Semi-Regular Multiresolution Models for Interactive Terrain Rendering. The Visual Computer 23 (2007)
8. Strugar, F.: Continuous Distance-Dependent Level of Detail for Rendering Heightmaps. Journal of Graphics, GPU, and Game Tools, 14(4), pp. 57-74 (2009)
9. Ulrich, T.: Rendering Massive Terrains using Chunked Level of Detail Control. Course Notes of ACM SIGGRAPH '02, ACM Press (2002)
10. Valdetaro, A., Nunes, G., Raposo, A., Feijó, B.: Understanding Shader Model 5.0 with DirectX 11. Proceedings do SBGames (2010)
11. Wagner, D.: Terrain Geomorphing in the Vertex Shader. En Wolfgang E. Shader-X 2: Shader Programming Tips & Tricks With Directx 9. Wordware Publishing (2004)
12. Zink, J., Pettineo, M., Hoxley, J.: Practical Rendering and Computation with Direct3D 11. A.K.Peters Ltd. CRC Press (2011)