Controle do Nível de Detalhe em Mapas Tridimensionais Multi-Resolução

Thiago Bedin Frustaci[1], *Roberto Scalco*[2]

Abstract — *This work presents the development of a library for reading and data manipulation, allowing the graphical representation of three-dimensional multi-resolution maps. This library encapsulates a set of algorithms that allow the programmer to read heightmaps represented in greyscale images and to create outdoor environments. The data of the heightmaps are transformed into a list of triangles. The library makes the processing and returns to the application only the triangles being observed by the virtual camera, using the structure quadtree. Moreover, another important factor to improve the performance is the control of the level of detail, representing the region next to the observer with more triangles than those farther away. In this way, a huge portion of the vertices are eliminated, diminishing the time spent with graphical processing and the memory consumption as well.*

Index Terms — Level of detail, linked list, multi-resolution maps, quadtree, three-dimensional maps.

INTRODUÇÃO

A manipulação eficiente de mapas topográficos tridimensionais em tempo real é base para aplicativos destinados à simulação de processos físicos. Esta associação permite a realização de estudos científicos, desde simulações didáticas utilizadas em um laboratório virtual para o ensino de Física até a extração de resultados que podem ser utilizados em diversas áreas da Engenharia.

Outra aplicação comum se faz presente na indústria do entretenimento. Podem ser criados jogos ambientados em campo aberto, permitindo que o jogador esteja imerso em um mundo com características reais. Estes jogos são construídos a partir de *engines* [1] desenvolvidas especialmente para controlar a eficiente tonalização dos diversos ambientes, além de realizar o tratamento necessário por outras operações como a inteligência artificial e a simulação física dos diversos objetos que compõem o jogo.

Os mapas utilizados nestas aplicações possuem grande quantidade de dados. Desta forma, a manipulação e o processamento destas informações tornam-se um ponto fundamental para o bom desempenho dos aplicativos gráficos. Além disso, outras operações concorrentes também consomem recursos do processador.

Desta maneira, torna-se necessário aplicar um conjunto de técnicas para modificação dos dados do mapa de forma a minimizar seu custo computacional, sem que o usuário final note uma queda significativa na qualidade visual quando o terreno for tonalizado.

Para tal, foi desenvolvida uma biblioteca (*dynamic linked library – DLL*) que visa dar o suporte necessário ao programador que necessite criar aplicativos que utilizem regiões abertas de topografia conhecida.

Nesta biblioteca é possível realizar a leitura de um mapa de alturas (representado sob a forma de imagem *bitmap*) em tons de cinza e a criação de um ambiente tridimensional multi-resolução. Essa biblioteca utiliza as técnicas apresentadas a seguir.

ALGUMAS DEFINIÇÕES

As técnicas e definições apresentadas a seguir foram pesquisadas e implementadas no projeto para permitir a exibição de um terreno com uma grande quantidade de triângulos ou uma malha muito detalhada [2-5].

Nível de Detalhe e Partição do Espaço

As técnicas para armazenar e manipular modelos, em tempo real, consistem na redução do número de triângulos exibidos. O nível de detalhe de uma região diminui quando os objetos visualizados estiverem se distanciando, de forma que não haja piora perceptível para o observador. Além disso, podem ser utilizados em conjunto outros algoritmos para particionamento do espaço, como a *quadtree* ou *octree*, além do uso da técnica de *frustum culling*.

Volume de Visualização

Na Computação Gráfica, *viewing frustum culling* ou simplesmente *frustum culling*, é o processo de remoção dos objetos que estejam completamente fora do campo de visão fazendo com que não sejam enviados ao *hardware* gráfico.

Mesmo com a tecnologia atual, ainda não é possível exibir todo o terreno, com alto detalhamento, sem uma queda na taxa de atualização dos quadros. Entretanto, determinados cálculos podem ser omitidos para evitar a queda de desempenho.

É possível usar o *frustum culling* moderadamente, diminuindo o número de cálculos realizados e equilibrando o custo de processamento gráfico e computacional sem comprometer a qualidade visual percebida.

1 Thiago Bedin Frustaci, Centro Universitário do Instituto Mauá de Tecnologia – Escola de Engenharia Mauá, Praça Mauá, 1, 09.580-900, São Caetano do Sul, SP, Brasil, thiago.bfrustaci@ceun.maua.br
2 Roberto Scalco, Centro Universitário do Instituto Mauá de Tecnologia – Escola de Engenharia Mauá, Praça Mauá, 1, sala G-02, 09.580-900, São Caetano do Sul, SP, Brasil, roberto.scalco@maua.br

Mip-mapping

Algumas técnicas como as apresentadas por [4] geram grandes discrepâncias em relação aos métodos tradicionais [2, 3]. O terreno é recomposto usando o *mip-mapping*, técnica comumente utilizada para o processamento de texturas. Entretanto, o *mip-mapping* deixa de considerar a topografia do terreno e a posição do observador (utilizada na determinação do nível de detalhe) gerando regiões planas desnecessariamente detalhadas, além de montanhas distorcidas pela falta de detalhamento. Tais problemas não ocorriam quando as técnicas tradicionais eram utilizadas.

Vertex Popping

Outro problema comum que prejudica visualmente ambas as técnicas é conhecido como *vertex popping*. Sua ocorrência se dá nas situações em que o nível de detalhe de uma determinada área logo à frente do observador é subitamente substituído por outro. Entretanto, na técnica *mip-mapping*, o *vertex popping* é facilmente visualizado, pois a representação da superfície pode introduzir um erro maior do que nas técnicas tradicionais. Para minimizar *vertex popping*, [5] recomenda que a técnica *geomorphing* seja aplicada, interpolando linearmente a altura dos vértices entre um nível de detalhe e outro.

Desta maneira, quanto melhor a capacidade do algoritmo em representar a superfície, menos perceptível será o *vertex popping*.

TRABALHOS RELACIONADOS

A estrutura básica dos algoritmos apresentados por [2, 4, 5] utiliza a abordagem *top-down* partindo do nó raiz de uma *quadtree*. A Figura 1 mostra que a *quadtree* pode ser visualizada subdividindo-se um plano em quatro quadrantes e assim sucessivamente para cada novo quadrante. Avalia-se cada nó ou quadrante com o *frustum-culling*.

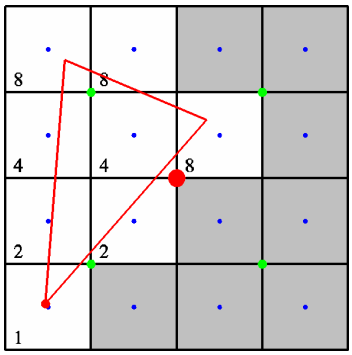

FIGURA. 1
AVALIAÇÃO DOS NÓS PELO *FRUSTUM CULLING*.

Em outras palavras, nas situações em que o nó pertence ao volume de visualização, analisa-se a necessidade de subdividi-lo em quatro nós filhos, considerando os parâmetros a seguir:

- nível máximo de subdivisão da *quadtree*;
- distância do observador ao centro do nó atual da *quadtree*;
- tamanho da aresta da *quadtree* que será utilizada na malha do terreno;
- erro em relação à um critério de qualidade, considerando a topografia do local.

Para cada nó, um nível de detalhe é calculado utilizando a distância entre a câmera e o centro do nó. O erro topográfico em relação à posição do observador e os diversos níveis de detalhe são considerados por [2, 5].

Como o nível de detalhe pode introduzir ou eliminar vértices, é comum encontrar áreas adjacentes com níveis de detalhe diferentes. Nestes casos, um ou mais vértices de fronteira podem ser introduzidos ou removidos sem que haja um correspondente no nível de detalhe da região vizinha. Este problema é conhecido como *triangle cracking*, pois a malha de triângulos nestes pontos não será contínua, apresentando buracos, como apresentado na Figura 2.

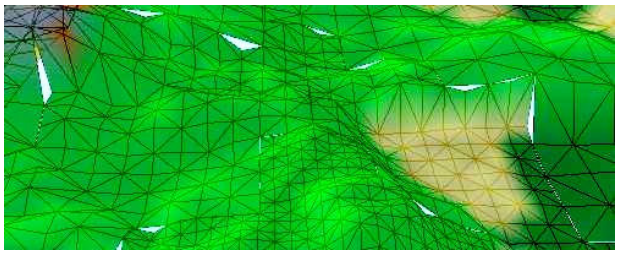

FIGURA. 2
TERRENO SEM A CORREÇÃO DO *TRIANGLE CRACKING*.

Para corrigir este problema, notificam-se os nós irmãos adjacentes para que removam um ou mais vértices, de forma a evitar o *triangle cracking*. Para que o processo de remoção de vértices seja simplificado, admite-se que o nível de detalhe entre nós adjacentes não ultrapasse uma unidade.

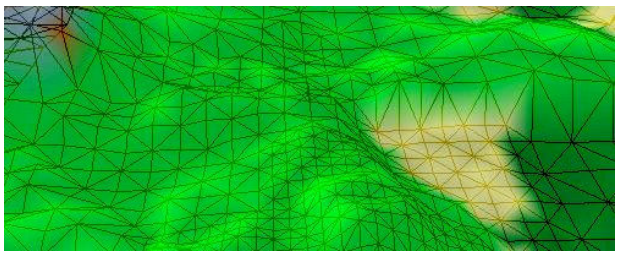

FIGURA. 3
TERRENO COM A CORREÇÃO DO *TRIANGLE CRACKING*.

O algoritmo apresentado por [2] possui as seguintes características:

- grande redução no número de polígonos tonalizados;
- transição suave entre diferentes níveis de detalhe;
- geração dinâmica de níveis de detalhe em tempo real;
- parâmetro de controle da qualidade da imagem.

Além disso, quaisquer algoritmos de nível de detalhe para mapas de altura devem atender aos seguintes critérios:

- a geometria e os componentes que a descrevem devem ser acessíveis diretamente e eficientemente, permitindo rápida indexação de polígonos e vértices;
- mudanças dinâmicas na geometria do terreno, resultando no recálculo de parâmetros da superfície ou geometria sem prejudicar o desempenho do sistema;
- dados com alta freqüência (como concavidades e convexidades localizadas, ou mudanças locais na geometria) não devem ter um efeito global na complexidade do modelo;
- pequenas mudanças nos parâmetros da câmera (como posição, direção de observação, campo de visão, etc.) geram pequenas mudanças na complexidade mantendo a taxa de atualização da imagem constante;
- controle da perda na qualidade de imagem pela redução do detalhe na geometria. Desta maneira, deve existir uma relação consistente e direta entre os parâmetros nível de detalhe e qualidade de imagem resultante.

O processo de simplificação da superfície é feito em duas etapas, denominadas *coarse-grained* e *fine-grained* [2].

A simplificação *coarse-grained* consiste em determinar quais níveis de detalhe serão necessários na geometria, seguida da simplificação *fine-grained* que considera cada vértice individualmente para remoção. Estes dois passos são executados para cada *frame* tonalizado. Todos os cálculos envolvidos na simplificação são realizados dinamicamente a partir da localização da câmera e da topografia do mapa.

O mapa de alturas é descrito por uma matriz de pontos elevados no plano *xy*, com um intervalo discreto de amostras. A superfície correspondente ao mapa de alturas é representada por uma malha de triângulos. A menor malha representável usada nesta triangulação é chamada de **malha primitiva** e tem dimensões de 3x3 vértices. As malhas maiores subseqüentes são formadas por agrupamentos de malhas menores em uma configuração matricial 2x2.

A triangulação obtida por [3] é semelhante à de [2], entretanto, apresenta estruturas mais consistentes para recriar a superfície, utilizando triangulação dinâmica fundamentada em árvore binária triangular. Além disso, são criados dois procedimentos para modificação desta árvore chamados de **divisão** (*split*) e **fusão** (*merge*). Estas operações são responsáveis por manter uma malha contínua, evitando o *triangle cracking*, enquanto adicionam ou removem vértices.

A técnica *quadtree* é utilizada por [5] para acelerar o processamento dos dados do mapa de alturas. Entretanto, uma matriz do mesmo tamanho que o mapa de alturas é utilizada para armazenar a *quadtree*. Desta maneira, são utilizadas $(2 \cdot n + 1)^2$ amostras, sendo $2 \cdot n + 1$ a quantidade de *pixels* em cada direção do mapa. Esta matriz é classificada como esparsa, pois a maior parte dos elementos não será utilizada. Os elementos utilizados são armazenados em variáveis lógicas, sendo que o valor **falso** é atribuído ao elemento (nó) que não foi subdividido já que o valor **verdadeiro** representa o centro de uma nova região em que o algoritmo realizou uma subdivisão.

A Figura 4 mostra uma matriz que contém os valores "**v**" e "**f**" como elementos armazenados da *quadtree*. O símbolo "**-**" indica elementos que não serão utilizados.

$$\begin{pmatrix} - & - & - & - & - & - & - & - & - \\ - & v & - & v & - & f & - & f & - \\ - & - & v & - & - & - & v & - & - \\ - & v & - & v & - & v & - & f & - \\ - & - & - & - & v & - & - & - & - \\ - & v & - & v & - & f & - & f & - \\ - & - & v & - & - & - & f & - & - \\ - & v & - & f & - & f & - & f & - \\ - & - & - & - & - & - & - & - & - \end{pmatrix}$$

FIGURA. 4
REPRESENTAÇÃO DA MATRIZ DA *QUADTREE*.

Analisando o requerimento de memória para tal matriz, caso fossem armazenados apenas os valores lógicos, a estrutura apresentada consumiria um byte por amostra do mapa de alturas. Esta estrutura de dados poderia ser utilizada desde que o consumo de memória permanecesse inalterado. Quando algoritmos como *geomorphing*, *frustum culling* e outros são aplicados, tornar-se evidente a necessidade de armazenar mais informações para cada nó da árvore, aumentando drasticamente o consumo de memória.

IMPLEMENTAÇÃO COM LISTA LIGADA

A solução proposta por [5] consome desnecessariamente a memória. A estrutura *quadtree* foi projetada para minimizar o consumo de memória nas técnicas de particionamento de espaço [6]. Desta maneira, a estrutura matricial não é conveniente para armazenar informações da *quadtree*.

Como a *quadtree* é uma estrutura em árvore, a melhor forma de armazená-la é utilizando uma lista ligada. Nela, cada nó armazena ponteiros para os nós filhos e para o pai, caso existam. A equação (1) e a Figura 5 mostram que, a partir dos filhos gerados para cada nó, é possível determinar o número de nós para um determinado nível de subdivisão. A soma da progressão geométrica dos nós gerados para cada nível resulta no número total de nós NN. Desta maneira, o número de nós gerados é função do número de subdivisões sd realizadas na árvore calculado a partir do tamanho do mapa de alturas.

$$NN = 4^0 + 4^1 + 4^2 + \cdots + 4^{sd} = \frac{4^{sd+1} - 1}{3} \quad (1)$$

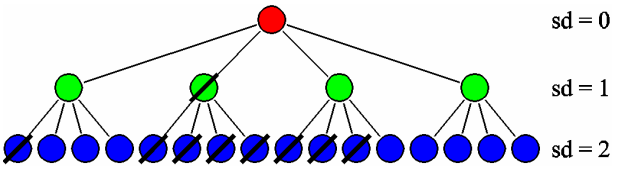

FIGURA. 5
SUBDIVISÃO DA ÁRVORE GERANDO 21 NÓS (*SD = 2*).

O consumo de memória pode ser exemplificado utilizando uma imagem em tons de cinza formada por $1\,025 \times 1\,025$ *pixels* ($s = 1\,050\,625$ elementos). A matriz que representa a árvore conterá esta quantidade de nós, porém a maior parte não será utilizada. Sob a forma de lista ligada, supondo o número máximo de subdivisões possíveis sd igual a seis, o número de nós criados é de $5\,461$. Ou seja, as arestas sm dos menores nós são 64 $\left(2^{sd}\right)$ vezes menores que o valor s.

Na prática, o parâmetro sd é controlado durante a execução do algoritmo tornando-se função do tamanho da aresta s do terreno e do tamanho das menores arestas sm. Esta relação é desenvolvida em (2).

$$\begin{aligned} \frac{s}{2^{sd}} &= sm \\ log\left(\frac{s}{sm}\right) &= sd \cdot log(2) \\ sd &= lg\left(\frac{s}{sm}\right) \end{aligned} \quad (2)$$

A *quadtree* é iniciada sem subdivisões, sendo que as únicas informações conhecidas são as coordenadas do centro da *quadtree* e o tamanho s do terreno, equivalente ao tamanho da *quadtree*.

A manipulação da *quadtree* é divida em um pré-processamento e o processo de manipulação.

Durante o pré-processamento, é realizada a criação completa da *quadtree* recursivamente até que sejam criados todos os nós finais. Para cada nó são armazenadas as respectivas coordenadas do centro e o comprimento da aresta, além de quatro ponteiros para possíveis filhos e outros quatro para nós adjacentes.

Os nós finais são armazenados temporariamente em uma lista não ordenada e uma busca é realizada para descobrir os nós irmãos (adjacentes) de cada elemento. Desta maneira, torna-se possível acessar informações como os níveis de detalhe dos adjacentes de quaisquer nós.

Uma vez preenchida, a etapa de tonalização é realizada a cada quadro (*frame*). Para tal, é necessário determinar se a câmera movimentou-se entre um quadro e seu antecessor. Em caso afirmativo, a lista de nós a serem tonalizados do quadro anterior é apagada. Recursivamente, o *frustum culling* é aplicado aos nós, incluindo na lista aqueles que pertencem ao volume de visualização. Além disso, o parâmetro de nível de detalhe de cada nó também é atualizado pelo cálculo da distância Manhattan entre a posição da câmera e as coordenadas de cada nó. Caso as características da câmera não sejam alteradas entre os quadros, a lista de nós é reutilizada.

Para cada nó da lista de tonalização verificam-se os níveis de detalhes entre nós adjacentes analisando a necessidade de codificação dos vértices a serem removidos na próxima etapa.

A partir das coordenadas do centro, do tamanho da aresta e do nível de detalhe do nó, os dados correspondentes ao mapa de alturas são enviados ao *hardware* gráfico.

RESULTADOS

A biblioteca está em contínuo desenvolvimento para o aperfeiçoamento das técnicas aqui apresentadas. O desenvolvimento é feito sobre o ambiente Visual Studio 2003 .NET utilizando a linguagem C++.

A OpenGL [7] é a API gráfica utilizada, servindo como interface entre o aplicativo e o *hardware* gráfico.

A biblioteca é capaz de ler o mapa de alturas a partir de um *bitmap* em tons de cinza, além de calcular e criar outro *bitmap* conhecido por mapa de vetores normais. Além disso, podem-se aplicar fatores de escala ao mapa e realizar o cálculo de sua área superficial.

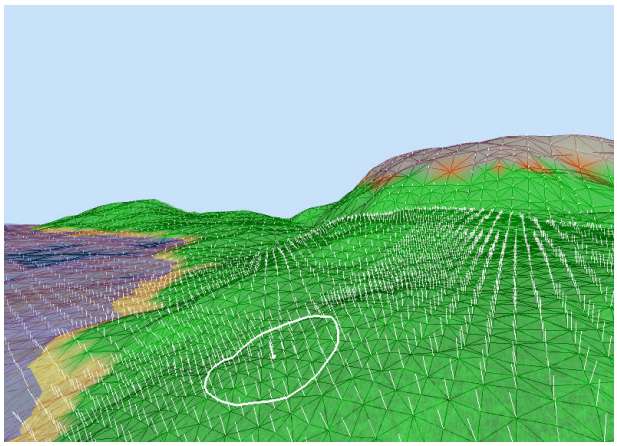

FIGURA. 6
REGIÃO TÍPICA.

A Figura 6 mostra um terreno de testes com tamanho de $1\,024 \times 1\,024$ *pixels* (vértices). Estão sendo exibidos $10\,552$ triângulos de um total de $2\,093\,058$. A área superficial do terreno é de $1\,205\,756\,u^2$. Existem apenas 392 nós finais (~9%) da *quadtree* no interior do volume de visualização e o

parâmetro *sd* foi automaticamente ajustado em 6 mantendo *sm* fixo. O círculo branco mostra a capacidade da biblioteca em calcular valores interpolados da altura em pontos não definidos no *bitmap*, além de interpolar os vetores normais.

A Figura 7 mostra a aplicação da técnica *frustum culling*, eliminando vértices não pertencentes ao volume de visualização, representado pelo tronco de pirâmide.

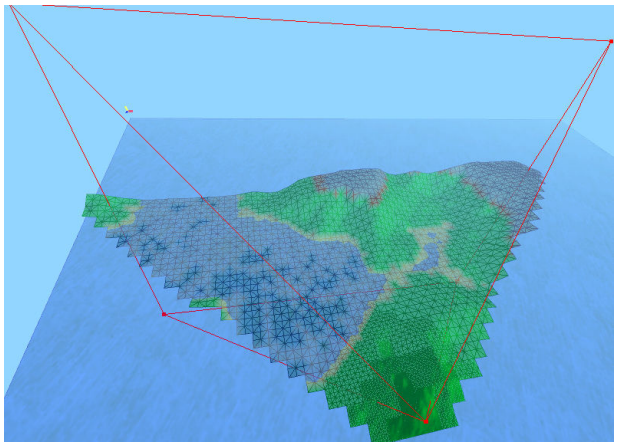

FIGURA. 7
VOLUME DE VISUALIZAÇÃO E TRIÂNGULOS RETORNARDOS COM A APLICAÇÃO DO *FRUSTUM CULLING*.

A Figura 8 ressalta os níveis de detalhe aplicados ao modelo com relação apenas à posição da câmera. Nota-se que, nas regiões de transição entre níveis de detalhe, são removidos diversos vértices evitando o *triangle cracking*.

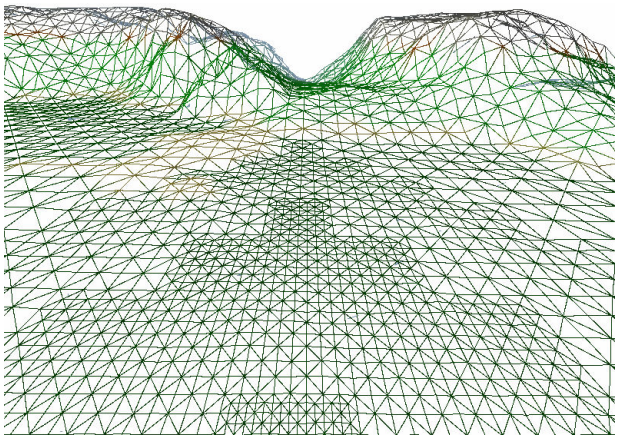

FIGURA. 8
REGIÕES COM DIFERENTES NÍVEIS DE DETALHES.

O desenvolvimento e testes foram realizados em um computador com processador AMD Athlon 1,0 GHz, 512 MB RAM DDR e uma placa de vídeo GeForce 3.

TRABALHOS FUTUROS

Podem-se aplicar métricas para refinar o cálculo do nível de detalhe, uma vez que apenas a posição da câmera é considerada. Novas métricas podem representar melhor a topografia do terreno. Desta forma, mais triângulos seriam utilizados nas regiões acidentadas e menos nas planas, melhorando a representação do terreno. Pode-se ainda associar um parâmetro de qualidade para que o controle da redução de nível de detalhe do terreno seja simplificado.

Além disso, para reduzir o *vertex popping* pode-se aplicar a técnica de *geomorphing*, quando houver mudança no valor do nível de detalhe das regiões.

CONCLUSÃO

Apresentou-se uma nova abordagem para manipulação de dados topográficos em tempo real utilizando o particionamento do espaço com a *quadtree*.

Sob a forma de lista ligada, a estrutura de dados mostra-se compacta e eficiente. Além disso, o suporte ao parâmetro de nível de detalhe está associado à redução do número de triângulos enviados ao *hardware* gráfico para a tonalização.

Embora a técnica que evita o *triangle cracking* realize algumas operações para codificar os vértices a serem removidos, não é significativa a queda de performance do aplicativo, mantendo a taxa de atualização da imagem (f.p.s.) constante.

Encapsulado sob a forma de uma DLL, pode ser utilizado em aplicações desde a indústria de jogos até as simulações de experiências de um laboratório de Física.

AGRADECIMENTO

Os integrantes do projeto agradecem ao Centro Universitário do Instituto Mauá de Tecnologia pelo apoio financeiro.

REFERÊNCIAS

[1] DevMaster.net – 3D Game and Graphics Engines Database. Disponível em <http://www.devmaster.net/engines>. Acesso em 10 out. 2005.

[2] LINDSTROM, P., *et al*. "Real-Time, Continuous Level of Detail Rendering of Height Fields". In: SIGGRAPH'96, New York, 1996. **Anais**. pp. 109-118.

[3] DUCHAINEAU, M. A., *et al*. "ROAMing terrain: real-time optimally adapting meshes". In: IEEE Visualization. 1997. pp. 81–88.

[4] LOSASSO, F., HOPPE, H. "Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids". In: SIGGRAPH'04, Los Angeles, 2004. Vol. 23, pp. 769-776.

[5] RÖTTGER, S., *et al*. "Real-Time Generation of Continuous Levels of Detail for Height Fields". In: 6th International Conference in Central Europe on Computer Graphics and Visualization. 1998. pp. 315-322.

[6] FOLEY, J. D., *et al*. "Computer Graphics – Principles and Practice". 2 ed. in C, Boston: Addison–Wesley System Programming Series, 1996, 1175p.

[7] OpenGL - The Industry Standard for High Performance Graphics, Disponível em <http://www.opengl.org>. Acesso em: 01 set. 2005.