Pontifícia **U**niversidade **C**atólica
do Rio de Janeiro

Visualização de terrenos em GPU

Leonardo Martins { lmartins@inf.puc-rio.br}

Disciplina: Visualização de Modelos Massivos
Professor: Alberto Raposo

Sumário

- Introdução
- Objetivos
- Visão geral
- Hierarquia de malhas
  - Balanceamento
- Renderização
- Gerenciamento de memória
- Resultados parciais
- A fazer
- Referências

Introdução

- Visualizar terrenos grandes de forma eficiente e com qualidade permanece sendo um grande desafio.
  - Imagens de satélite da ordem de bilhões de amostras estão disponíveis
  - A capacidade de processamento e armazenamento das máquinas atuais cresce na mesma proporção da capacidade de resolução dos *scanners* e *displays*
  - Terrenos estão presentes em um grande número de aplicações computacionais, tais como GIS, jogos e simuladores de vôo.
- Técnicas de visualização tais como LoD (Level of Detail) e *culling* ajudam a reduzir o número de primitivas e a aumentar a taxa de exibição de quadros.

Introdução

- Propriedades desejadas em algoritmos de visualização de terrenos (Bösch et al, 2009)
  - Suporte a LoD
  - Renderização de alta-perfomance
  - Exibição contínua
  - Recuperação rápida de dados
  - Armazenamento compacto
  - Acesso direto aos dados
  - Simplicidade
  - Pré-processamento rápido

Objetivo

- Implementar através da GPU um visualizador de terrenos
  - Taxas de exibição aceitáveis
  - Qualidade
  - Gerenciamento de memória
- "GPU-Friendly High-Quality Terrain Rendering"
  - Schneider and Westermann, 2006

Visão geral

- Pre-processamento
  - Divide-se o terreno em *tiles* e para cada um é gerado um conjunto discreto de LoDs através de uma hierarquia de malhas (quad-tree)
- Execução
  - LoDs contínuos podem ser gerados através de interpolação dos valores de altura dos vértices via GPU
  - Não é feita nenhuma re-triangulação de malha
  - Reduzida a necessidade de largura de banda (menos triângulos)
  - Garantia de erros pequenos com altas taxas de exibição

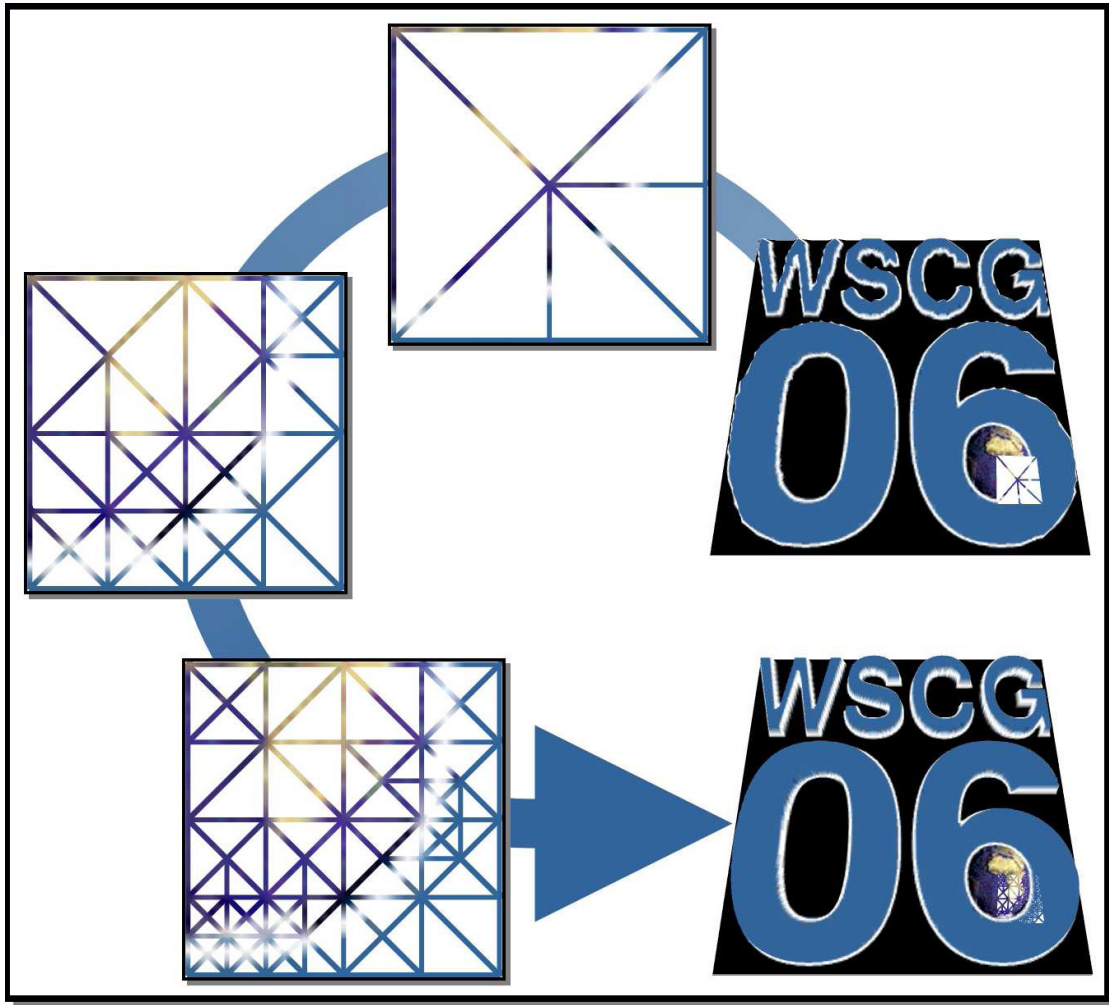
Hierarquia de malhas

- Um dado campo de alturas H: $N^2 \rightarrow Z$ pode ser aproximado por uma malha triangular sobre um domínio 2D.
- A superfície define uma reconstrução H' de H.
- A qualidade da reconstrução pode ser medida através de uma métrica de erro que se estendem por todo o domínio espacial δ: R x R → R
- A hierarquia é dita *aninhada* em relação à triangulação
  - *O triângulo no nível i está inteiramente contido no de nível i+1*
  - Dessa forma, tal hierarquia pode ser inteiramente gerada por uma quadtree

Hierarquia de malhas

- Define-se um vetor de erros (e_0, e_1, ..., e_{n-1}) para cada nível, tal que $e_i = 2^{n-1-i}$
- Partindo do nível 0, uma hierarquia $\{M_i\}$ $(0 \leq i \leq n-1)$ é construída tal que
  - V_i C V_{i+1}
  - $e_{i+1} \leq \delta(H'_i, H) \leq e_i$
- O refinamento continua até que $\delta(H'_{i+1}, H) \leq e_{i+1}$

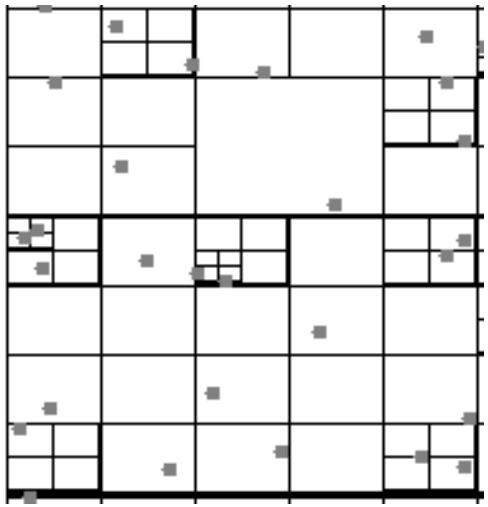
Hierarquia de malhas



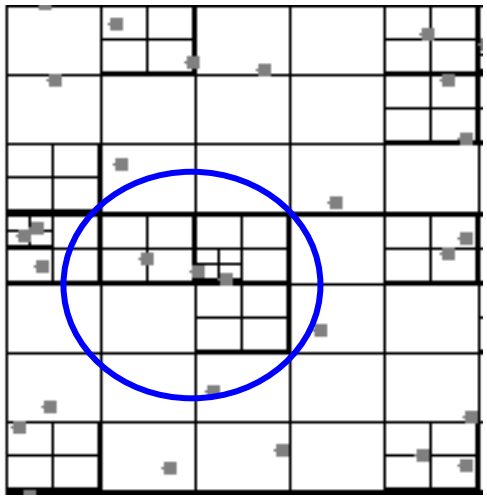
Balanceamento

Uma *quadtree* é dita balanceada se quaisquer dois quadrados vizinhos diferem no máximo de um fator dois; desta forma em uma *quadtree* balanceada, quaisquer duas folhas cujos quadrados são vizinhos, tem profundidades diferindo no máximo de 1.

Não balanceada



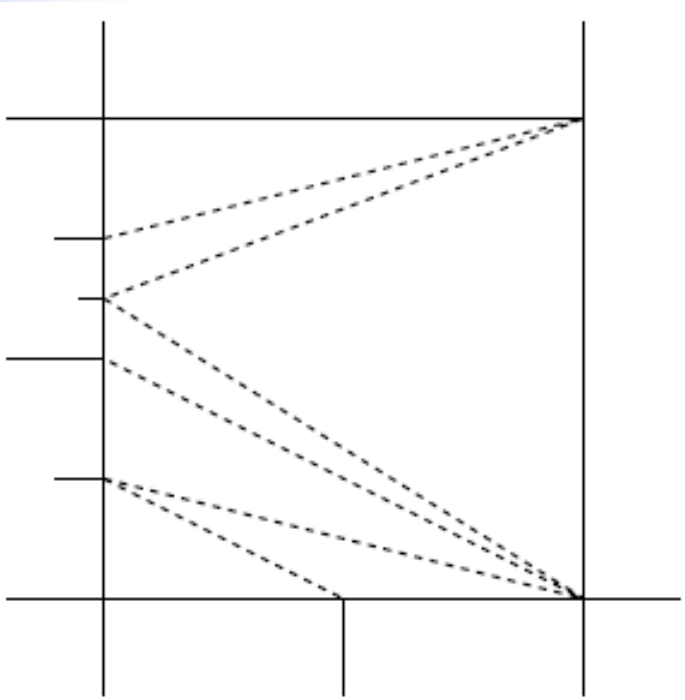
Balanceada



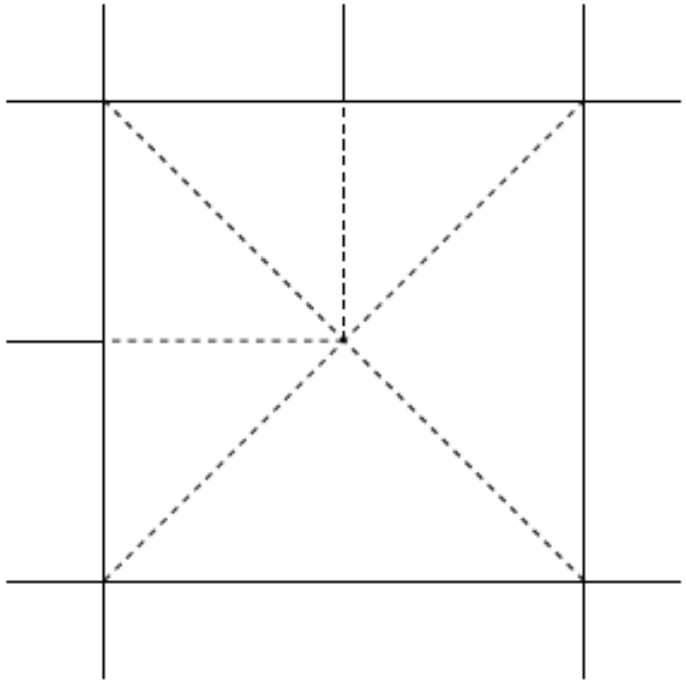
Balanceamento

Verificar se um quadrado s precisa ser dividido

1. Obter os vizinhos norte, sul, leste, oeste.
2. Dado um quadrado S verificamos seu vizinho norte por exemplo.
3. Verificamos se é folha
    3.1. Caso negativo, verificamos os filhos cujos lados sejam adjacentes ao lado do quadrado S e verifica se são folhas. Caso negativo, divide-se S e acrescentam os quatros filhos na lista para ser analisados.
4. Caso o vizinho norte não satisfaça as condições para divisão de S, tomam-se os demais vizinhos.



Malha obtida com quadtree não balanceada



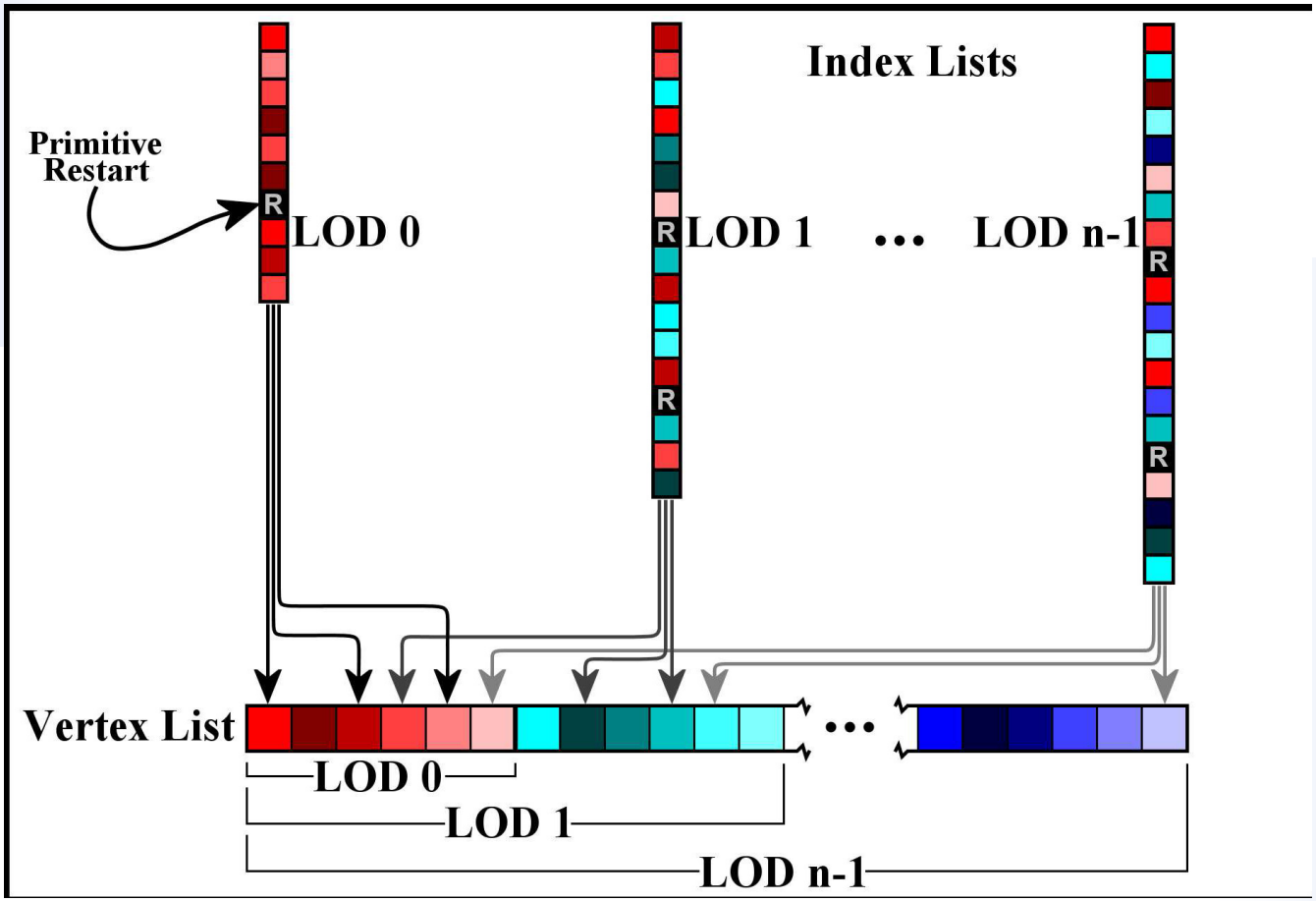
Malha obtida com quadtree balanceada

Geração de Malhas

Renderização

- A hierarquia de malhas permite que os tiles sejam enviados progressivamente a GPU
- Na GPU, rendering de tempo real em alta qualidade é realizado através de uma estrutura de dados apropriada
- Ao mesmo tempo, a CPU realiza o *frustum culling* e os cálculos de LoD para cada *tile*

Renderização



Renderização

- Para cada tile, calcula-se o bounding box para operações de *frustum culling*
- Para cada quadro, tiles visíveis são ordenados em ordem de profundidade para aproveitar o teste de profundidade anterior e evitar sobre-desenhos
- Um gerenciador de memória garante que todos os *tiles* visíveis podem ser renderizados pela paginação dos dados não residentes na GPU
- Para cada tile visível, o LoD apropriado é computado via CPU.

Renderização

- Para achar o LoD adequado para cada *tile* usa-se a matriz de projeção, que mapeia coordenadas no espaço do objeto $v = (v_1, v_2, v_3, 1)$ para o espaço de tela $s = (s_1, s_2, s_3)$

$$\rho = \sqrt{\frac{\sum_{i=1}^{3}\left(\frac{\partial v_i}{\partial s_1} ds_1 + \frac{\partial v_i}{\partial s_2} ds_2\right)^2}{ds_1^2 + ds_2^2}}$$

Renderização

- Na CPU, ρ_j é calculado para cada canto j da *bounding box* de cada *tile*
- O valor ótimo é dado por $\lambda_j = \lambda_{max} - \log2(\rho_j)$, onde $\lambda_{max} = n-1$ (número de níveis)
- A malha $M_{min}\{\lambda_j\}$ é selecionada para a renderização do *tile*.

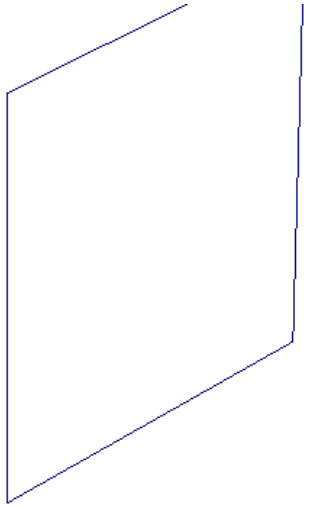
Gerenciamento de memória

- Aloca blocos de memória de tamanho variável exponencialmente, com meta-informações como *time-stamp*
- Paginação é implementada como mistura entre Last Recently Used (LRU) e Tighest Fit (TF)
  - *Ao carregar um tile A para a GPU, sempre procura pelo bloco B mais antigo e com tamanho mínimo para acomodar A*

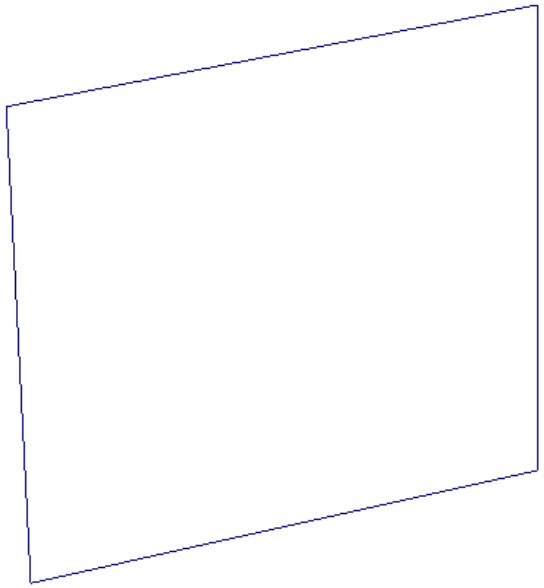
Resultados parciais

- Implementação da malha hierárquica através de uma quadtree balanceada
  - permite gerar uma malha de triângulos bem comportados
  - Dado um nível da hierarquia, retorna a malha correspondente

Resultados parciais



Resultados parciais



A fazer

- LOD adaptativo em relação à distância
- Divisão do dado em tiles
  - Cada tile representado por uma quadtree
  - Formato binário para armazenamento em disco
- Implementação em GPU
  - Usar uma linguagem de mais alto nível (CUDA)

Referências

- "GPU-Friendly High-Quality Terrain Rendering". Schneider and Westermann, 2006
- M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 1997.
- RASTeR: Simple and Efficient Terrain Rendering on the GPU. Bösch, Goswami and Pajarola, 2009.
- http://www.vterrain.org/LOD/Papers/
- http://lodbook.com/terrain/