Daniel Luiz Alves Madeira

Métodos de Adaptação de Grandes Terrenos para Visualização 3D

Juiz de Fora

08/02/2007

Daniel Luiz Alves Madeira

Métodos de Adaptação de Grandes Terrenos para Visualização 3D

Orientador:
Marcelo Bernardes Vieira

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Juiz de Fora

08/02/2007

Monografia submetida ao corpo docente do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como parte integrante dos requisitos necessários para obtenção do grau de bacharel em Ciência da Computação

Prof. Marcelo Bernardes Vieira, D. Sc.
Orientador

Prof. Marcelo Lobosco, D. Sc.

Profa. Jesuliana Nascimento Ulysses, M. Sc.

Sumário

Lista de Figuras

1 Introdução

Um dos grandes problemas da computação gráfica é o equilíbrio entre complexidade e desempenho. Sempre há a necessidade de se dosar realismo e velocidade, fidelidade e taxa de quadros, mundos bem detalhados e animação contínua. Como tentativa de ligar complexidade e desempenho, surgiu a pesquisa de adaptação por níveis de detalhe.

A pesquisa em torno da adaptação por nível de detalhe é importante pois, mesmo com o avanço do hardware gráfico, a complexidade dos modelos 3D utilizados aumenta a cada dia. Não importa quão poderosa é a plataforma, o número de polígonos enviados para processamento tende a ser maior que a capacidade máxima.

Para ilustrar a limitação quanto ao número de polígonos (medida comum para complexidade dos modelos 3D), tomemos como exemplo um objeto de 2048×2048 vértices. Temos um total de 8.380.418 triângulos $(2 \times 2047 \times 2047$ triângulos$)$ na malha completa. Em cada segundo, é necessário processar este objeto 60 vezes, o que gera 502.825.080 triângulos por segundo para serem processados pela GPU (*Graphic Processor Unit*, ou Unidade de Processamento Gráfico). Se levarmos em conta que numa cena são utilizados muitos objetos simultaneamente, é fácil visualizar a grande limitação do hardware disponível no mercado. Uma placa NVidia GeForce® FX5200, placa bastante comum em computadores pessoais hoje, processa somente cerca de 350 milhões de triângulos por segundo.

Temos então o conceito fundamental e simples da adaptação por nível de detalhe: utilizar objetos mais detalhados (e, portanto, com mais polígonos para serem processados) em áreas mais importantes da cena, e deixando áreas menos importantes com nível de detalhe menor.

Este trabalho se foca em uma subárea de métodos de adaptação geométrica que é a de adaptação de nível de detalhe de grandes terrenos. Este estudo tem particular importância em aplicações como SIG's (Sistemas de Informação Geográfica), aplicações de planejamento de missões militares, simuladores de vôo realistas e jogos baseados em terreno. Aparece ainda, por exemplo, em problemas como posicionamento de antenas de celular, em que a topologia da área, junto com outros fatores, influencia na área de cobertura do sinal da antena.

Fazendo um comparativo entre nível de detalhe de terrenos versus nível de detalhe tradicional, temos alguns pontos de diferença marcantes, sendo eles:

- **topologia mais simples**: terrenos tendem a ter uma topologia homeomorfa ao plano, diferentemente da área tradicional, que trata também de outras topologias, como a homeomorfa à esfera e ao toro.
- **algoritmos especializados e simples**: como a topologia do plano é mais simples de ser tratada, os algoritmos são simplificados. Além disso, em geral não há ocorrência de outras topologias sobre o terreno, o que torna os algoritmos de nível de detalhe ainda mais especializados.
- **modelos contínuos e grandes**: os modelos de terreno geralmente são muito grandes, tendo um custo muito alto para processamento.
- **simultaneamente próximo e distante do observador**: ao se olhar para a frente, vemos o mesmo terreno que está perto indo até o horizonte. Isso gera algumas dificuldades, pois precisamos de um modelo mais detalhado perto do ponto de visão (a câmera) e devemos diminuir a quantidade de detalhes no horizonte (visto que o número de polígonos processados é restrito).
- **adaptação dependente do observador**: é preciso observar as áreas mais detalhadas (áreas próximas ao observador, morros ou vales, etc.) com resolução diferente daquelas com menos detalhes (áreas planas, por exemplo).

Em resumo, o foco da adaptação por nível de detalhe para grandes terrenos é criar um modelo virtual de terreno com a maior fidelidade possível para aproximar o mundo real, respeitando as limitações impostas pela plataforma.

1.1 Objetivos

Muitos trabalhos já foram publicados sobre o tema e este trabalho tem por objetivo primário estudar, apresentar e implementar algumas soluções clássicas, sintetizando os fundamentos e os problemas relacionados à adaptação de terrenos. Como objetivo secundário, pretende-se implementar um visualizador de terrenos a fim de comparar algumas dessas soluções clássicas.

1.2 Organização do trabalho

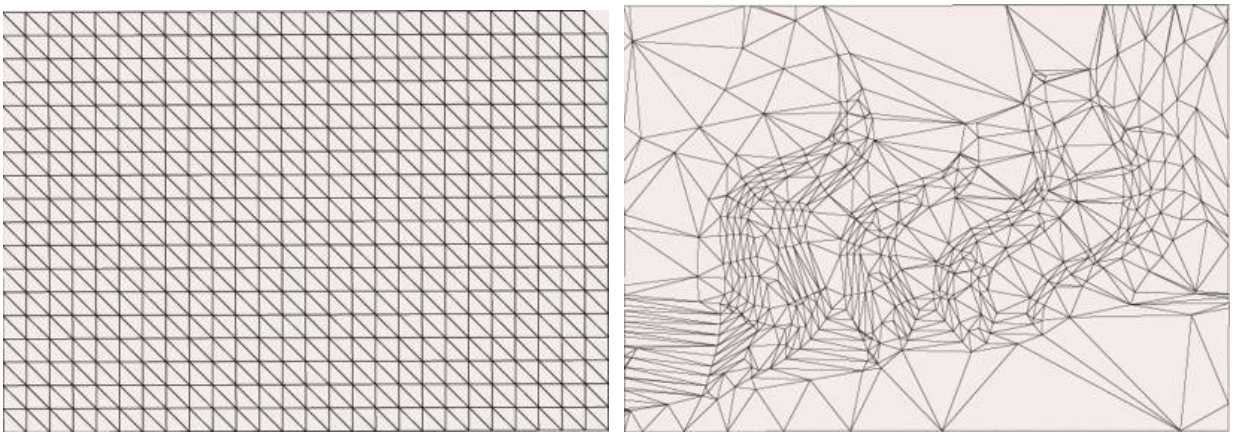
No Capítulo 2 serão apresentados os fundamentos básicos para a adaptação. Tais fundamentos são necessários para a compreensão do capítulo 3, onde serão apresentadas as especificidades de algumas soluções. Serão apresentadas três soluções clássicas e uma solução recente. No capítulo 4 será apresentada a implementação do vizualizador e serão comparados alguns métodos de adaptação.

2 *Fundamentos para adaptação de terrenos*

Para se implementar um algoritmo de nível de detalhe de terrenos, precisamos estudar as principais variáveis necessárias para tal. Os fundamentos e conceitos básicos serão apresentados neste capítulo. As soluções clássicas se baseiam em duas noções básicas: malhas adaptáveis e métricas de erro.

2.1 Malhas adaptáveis de terrenos

Se precisamos criar um modelo virtual de um terreno real, precisamos de uma estrutura que guarde as informações necessárias para a reprodução desse modelo. Dois modelos muito utilizados são as malhas regulares e as TINs (*Triangulated Irregular Networks*, ou Redes Irregulares Triangularizadas). As malhas regulares são vetores com sequências de altitudes, para coordenadas (x,y) igualmente distribuídas sobre a malha. Já as TINs permitem espaçamento variável entre as coordenadas (x,y). A Figura 1 mostra um exemplo de cada modelo.



(a) Malha Regular (b) TIN

Figura 1: *Exemplo de uma Malha Regular e de uma TIN (STEPHEN; HASBROUCK, 2001)*

As TINs geralmente se adaptam a uma superfície com menos polígonos do que uma malha

regular na mesma resolução. Já que os vértices podem ser distribuídos na malha com espaços variáveis, áreas planas podem ser representadas com uma amostragem inferior, enquanto áreas irregulares, como montanhas ou vales, podem utilizar mais polígonos.

As TINs são mais complexas para serem processadas e também possuem gasto de memória maior, pois é necessário armazenar as coordenadas (x, y, z). Já as malhas regulares, além de mais simples, têm custo computacional menor, pois precisam armazenar menos dados. Em algumas abordagens, basta a coordenada z.

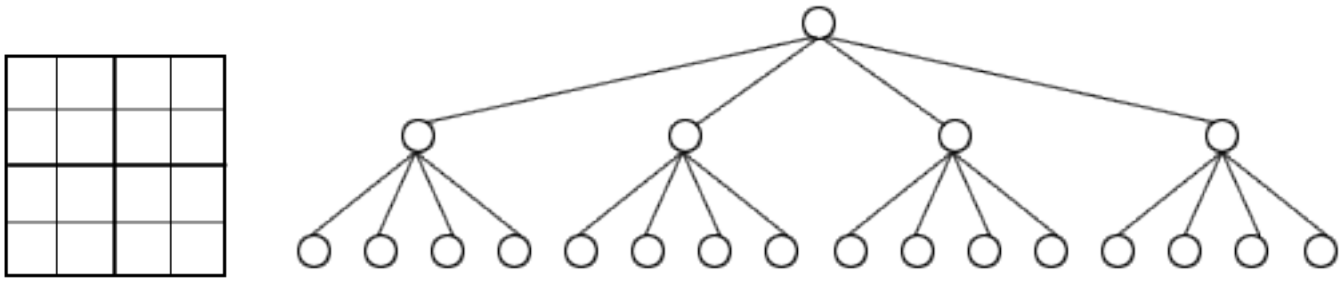
Como as TINs tendem a ser menos eficientes que as malhas regulares, essas últimas são mais utilizadas em adaptação em nível de detalhe de terrenos.

2.1.1 Representação por malhas regulares

As estruturas mais comuns são a ***quadtree*** e a *binary triangle tree*, ou somente ***bintree***. Ambas oferecem uma representação hierárquica que permite a adaptação das diferentes partes da malha em diferentes resoluções.

2.1.1.1 *Quadtree* restrita

A *quadtree* é definida como uma árvore onde cada nó interno possui até quatro filhos. Assim, interpretando geometricamente, a *quadtree* divide uma área retangular em 4 quadrantes, geralmente iguais. Cada quadrante possui um único vértice em seu centro e pode ser dividido recursivamente em outros quatro quadrantes. Na Figura 2 podemos observar alguns níveis da *quadtree*.

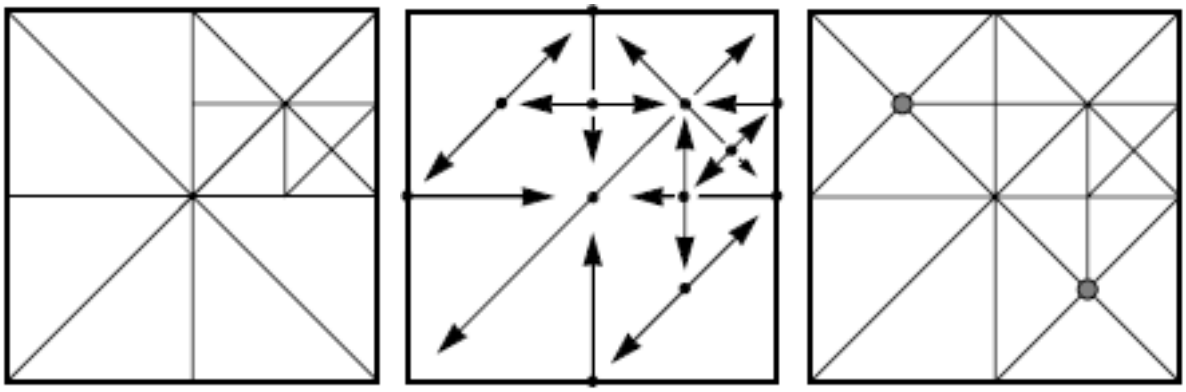


(a) (b)

Figura 2: Quadtree *(a) como malha e (b) sua representação como árvore*

Mesmo trabalhado com áreas retangulares, as *quadtrees* podem perfeitamente representar triângulos. Basta que cada região seja decomposta em dois ou mais triângulos.

Durante a geração da malha, precisamos adicionar algumas restrições que impeçam o aparecimento de algumas descontinuidades (discutidas na Seção 2.3). Uma abordagem para se evitar descontinuidades na *quadtree* é através de **níveis de hierarquia**. Tem-se uma malha contínua adicionando-se a restrição de que a diferença dos níveis (profundidade na árvore) de dois quadrantes vizinhos seja no máximo de um nível e que cada quadrante seja dividido em oito triângulos, a menos que o quadrante seja vizinho a um quadrante maior. Todavia, este método produz triângulos extras e não há algoritmo determinista para converter uma *quadtree* simples em uma restrita. Outra maneira de garantir a continuidade é através de um **grafo de dependência** dos vértices. O grafo de dependência fornece um conjunto de regras que garantem que o conjunto de vértices gerem uma malha contínua. Assim, a partir de uma *quadtree* não-restrita, basta que sejam resolvidas as regras de dependência para que uma *quadtree* contínua seja alcançada (Fig. 3).



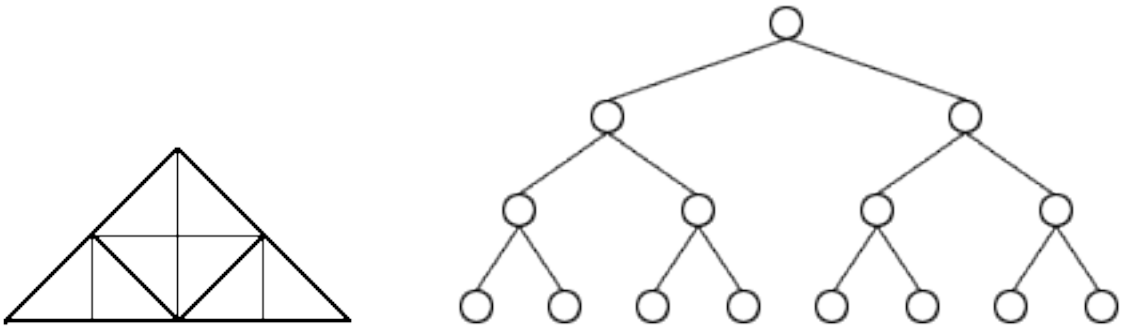
(a) *Quadtree* não-restrita (b) Grafo de dependência (c) *Quadtree* restrita

Figura 3: *Restrição da* quadtree *a partir de grafo de dependência (PAJAROLA, 1998)*

2.1.1.2 *Bintree* triangular

A *bintree* triangular é uma árvore binária, onde, geometricamente, cada nó da árvore representa um triângulo. O triângulo raiz, $T = (v_a, v_0, v_1)$, é definido como um triângulo reto isósceles no nível menos refinado, $k = 0$. No próximo nível, $k = 1$, os filhos da raiz são definidos através da inserção da aresta definida entre o vértice ápice v_a e o ponto médio v_c da aresta base (v_0, v_1). O filho à esquerda é $T_0 = (v_c, v_a, v_0)$ e o filho à direta é $T_1 = (v_c, v_1, v_a)$. Repetindo recursivamente este processo, obtém-se o restante da *bintree*. A Figura 4 ilustra alguns níveis de uma *bintree*.

Também há a necessidade de adicionarmos restrições à *bintree* para evitar descontinuidades. A abordagem mais comum é através da **divisão forçada** dos vértices. Para isso precisamos definir a vizinhança de um triângulo. Define-se como triângulo base T_b, um triângulo que



(a) (b)

Figura 4: Bintree *(a) como malha e (b) sua representação como árvore*

divide a aresta base (v_0, v_1) de T. O vizinho à esquerda T_e é um triângulo que divide a aresta esquerda (v_a, v_0) e o vizinho à direita T_d, o que divide a aresta direita (v_1, v_a).

Quando dois triângulos T e T_b estão no mesmo nível k (ou seja, no mesmo nível de detalhe), o par (T, T_b) é referido como um **diamante**. A operação de refinamento sobre o diamante é realizada dividindo os dois triângulos, gerando assim quatro novos filhos: (T_0, T_1), filhos de T e (T_{b0}, T_{b1}), filhos de T_b. Se o triângulo T_b não existir, somente T é refinado. A operação de simplificação une os filhos $(T_0, T_1, T_{b0}, T_{b1})$, os substituindo na malha pelos seus pais (T, T_b). Nesse caso, $(T_0, T_1, T_{b0}, T_{b1})$ formam um diamante simplificável (Fig. 5).

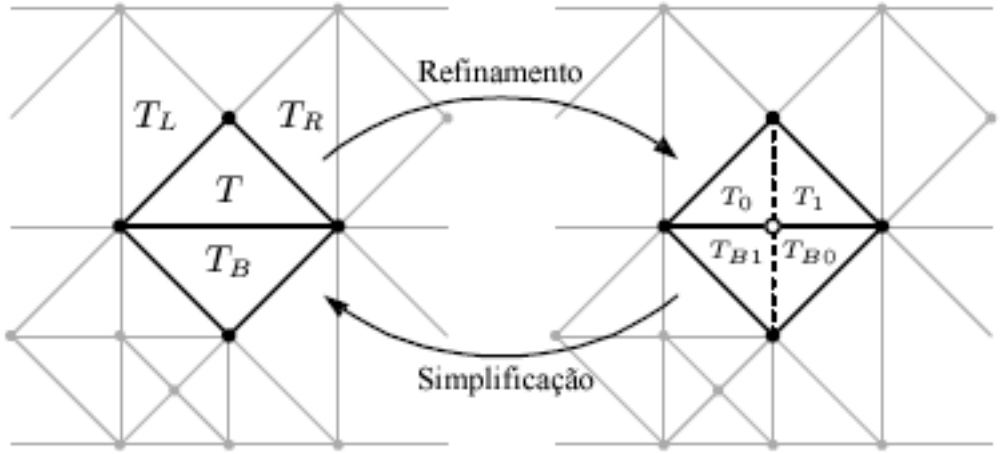


Figura 5: Refinamento e Simplificação do diamante (DUCHAINEAU et al., 1997)

Um triângulo T não pode ser dividido caso seu triângulo base T_b esteja num nível k menor (menos refinado). Para dividirmos T, é preciso antes forçar a divisão de T_b, o que pode exigir outras divisões forçadas. Um exemplo onde a divisão forçada é necessária é apresentada na Figura 6.

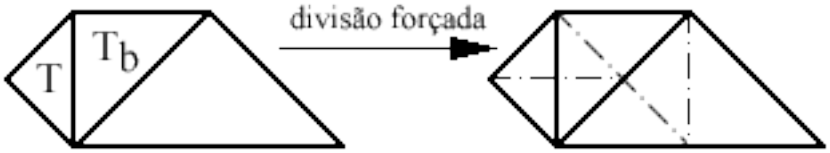


Figura 6: Divisão forçada para divisão do triângulo *T*

Vale citar que estes métodos de divisão são os mais comuns, porém não são os únicos. Por exemplo, cada quadrante da *quadtree* pode ser dividido em 4 quadrantes de áreas diferentes, e cada triângulo da *bintree* pode ser dividido não necessariamente pela divisão da hipotenusa em duas partes iguais. Estes dois exemplos, todavia, geram malhas irregulares.

2.2 Abordagem *top-Down* ou *bottom-Up*

Com uma estrutura já definida, precisamos agora definir como adaptá-la. Há duas abordagem possíveis: ***top-down*** ou ***bottom-up***. A abordagem *top-down*, que também podem ser chamada de método de **refinamento**, se caracteriza por iniciar com uma malha na mínima resolução possível e, progressivamente, subdividi-la adicionando novos triângulos até atingir a definição desejada. Já a abordagem *bottom-up*, também chamada de método de **simplificação**, faz o caminho inverso. A partir da malha com resolução máxima, seus triângulos vão sendo unidos, até que a malha esteja na resolução desejada (Fig. 7).

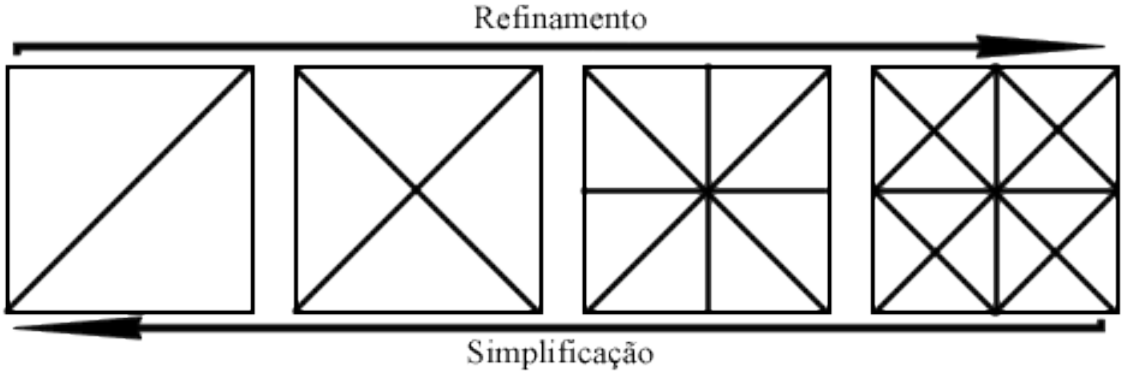


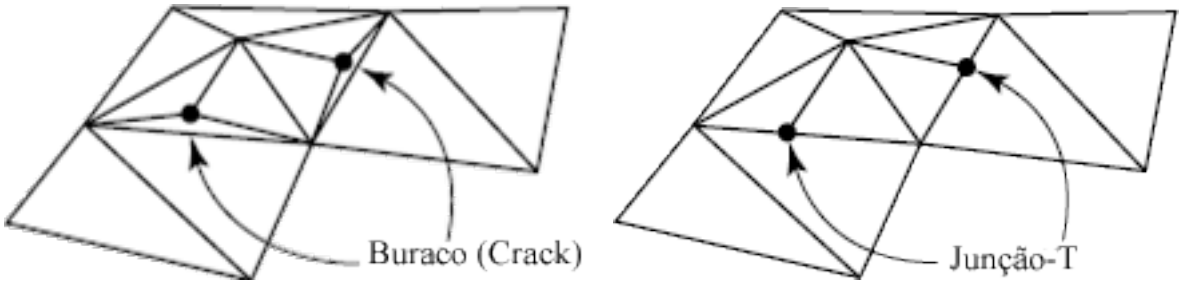
Figura 7: Refinamento e simplificação da malha

Usualmente, os algoritmos *bottom-up* são utilizados para pré-processamento dos dados, antes da execução do programa, pois são mais custosos. Já durante a simulação, os algoritmos *top-down* são preferidos pela maior simplicidade e por exigir bem menos recursos para

a adaptação. Alguns sistemas utilizam-se de uma técnica híbrida, simplificando ou refinando o terreno em cada quadro, para se utilizar da coerência quadro-a-quadro (sec. 2.6.3). Estes sistemas não são classificados nem como *top-down* nem como *bottom-up*.

2.3 Problemas em adaptação de malhas de terrenos

Ao trabalharmos com adaptação de malhas de terrenos, temos um problema quando dois triângulos adjacentes existem em diferentes níveis. Nessa situação, é possível que aconteçam descontinuidades na malha, quando um vértice inserido num nível mais refinado não está contido na malha do nível anterior. Quando renderizado, aparecem buracos. Outro problema é a junção-T, que acontece quando um vértice de um modelo mais refinado não compartilha um vértice do modelo no nível anterior. Isso pode resultar em quebras pelo modelo, por causa de pequenos erros de arredondamento do computador. A Figura 8 ilustra esses casos.



(a) Buraco (*Crack*) (b) Junção-T

Figura 8: *Descontinuidades que podem aparecer na malha (LUEBKE et al., 2003)*

Existem muitas maneiras de se tratar essas descontinuidades. Algumas das mais comuns são:

- Os triângulos ao redor da descontinuidade são recursivamente subdivididos para produzir uma superfície contínua. Há o aparecimento de triângulos adicionais, mas o resultado é suave. Essa abordagem é a utilizada para evitar descontinuidades na *bintree*.
- O vértice extra do modelo mais definido tem sua altura modificada, a fim de que ele fique contido no modelo mais simples. Essa abordagem não altera o número de triângulos, mas adiciona uma junção-T, o que pode ser prejudicial, pois podem ocorrer quebras.
- Um novo vértice é inserido na malha mais simples e sua altura é igualada a altura do vértice da malha mais refinada. O modelo criado é mais fiel do que a solução anterior, mas pelo custo de mais um vértice.

- Um novo triângulo é inserido entre as duas malhas, para cobrir a falha. Apesar do resultado ser uma malha contínua, há o aparecimento de pequenos despenhadeiros. Há também o problema do custo da inserção de novos polígonos.

2.4 Gerenciamento de grandes massas de dados

Um objetivo desejado para algoritmos de nível de detalhe para terrenos é a capacidade de lidar com grandes massas de dados, mesmo que elas excedam o tamanho da memória principal disponível. Com a capacidade de recuperar os dados mesmo fora da memória principal é possível, por exemplo, transmitir os dados pela Internet.

Uma das maneiras propostas mais simples para resolver o problema gerado pela grande massa de dados é reorganizar os dados no disco, fazendo com que o próprio sistema operacional seja o responsável pela paginação dos dados para a memória. Alguns sistemas foram criados usando também um mecanismo explícito de paginação, carregando pequenas áreas da malha, ou ladrilhos, de acordo com o necessário, utilizando um buffer para controle dessa operação. A primeira solução impede que os dados estejam distribuídos. Já a segunda oferece a oportunidade de se obter dados em outras máquinas. Assim, esse mecanismo suporta massas de dados distribuídas via rede.

Métodos que processam dados que são muito grandes para caber na memória principal são chamados *out-of-core*. Quando um método não é *out-of-core*, é necessário que a malha esteja completamente carregada na memória.

2.5 Métricas de erro - dividir ou não dividir?

Ao se trabalhar com terrenos, é desejável que possamos adaptar a sua malha automaticamente e não trabalhar com somente alguns níveis de detalhes pré-modelados. Para tal, além de uma malha adaptável, precisamos de uma medida que permita sabermos quando refinar ou simplificar a malha. Esta é a idéia da métrica de erro: fornecer um mecanismo para que a adaptação da malha seja automática e que áreas mais importantes, como regiões montanhosas e áreas perto do observador, estejam mais refinadas do que aquelas que não exigem tantos detalhes, como áreas no horizonte ou planas. Geralmente dois tipos de métricas de erro são utilizadas: a **métrica de erro geométrico** e a **métrica de erro dependente do observador**.

A maneira mais comum de se calcular o erro geométrico é utilizando o conceito de **espaços aninhados**. Sendo D_i o conjunto dos descendentes de um vértice i, tem-se que $\text{erro}(i) \geq$

erro $(j), \forall j \in D_i$. Assim, garante-se a monotonicidade da métrica de erro, pois o erro dos pais é sempre maior ou igual ao erro de todos os seus descendentes.

O erro dependente do observador comumente é uma medida de distância. Calcula-se a distância de um ponto real e do ponto aproximado pela malha. Tal medida de erro faz com que o erro dos pontos mais pertos do observador e dos pontos que estão em áreas mais detalhadas sejam maiores, fazendo com que essas áreas sejam mais refinadas pelo processo de adaptação.

2.6 Otimizações

Existem algumas otimizações que podem ser aplicadas nos métodos. Tais otimizações diminuem o tempo de processamento de cada quadro, além de diminuir a carga de polígonos passada à GPU.

2.6.1 Descarte de elementos não visíveis

Esta técnica, conhecida na literatura por *view frustum culling*, consiste em, antes de dividir um triângulo, verificar se o mesmo se encontra visível. Assim, pode-se descartar triângulos que, por não serem visíveis, são desnecessários. Basicamente, basta marcá-lo, antes da divisão como DENTRO, FORA ou NO LIMITE e, quando o algoritmo for processá-los, ignorar todos os que estiverem marcados como FORA.

2.6.2 Construção de faixas de triângulos

Se dois triângulos são vizinhos, então eles compartilham uma de suas arestas e, consequentemente, dois de seus vértices. Com isso, os triângulos formam uma faixa. Ao enviarmos os triângulos para a GPU, podemos nos aproveitar desse fato utilizando o que se chama de *stripping*.

Sejam dois triângulos $T_a = (v_a, v_0, v_1)$ e $T_b = (v_b, v_0, v_1)$, que compartilham a aresta (v_0, v_1). A GPU pode formar triângulos com os três últimos vértices passados a ela. O *stripping* funciona da seguinte maneira: ao invés de enviarmos os três vértices de cada triângulo repetidas vezes, podemos enviar somente os vértices v_a, v_0, v_1 e v_b, nessa ordem, e a GPU processa os triângulos normalmente. Assim, pode-se economizar com o envio repetido de vértices para a GPU.

2.6.3 Coerência quadro-a-quadro

A cada instante de tempo, as malhas pouco se modificam. Uma malha A num instante de tempo t é bem próxima da malha no instante $t+1$. Assim, ao invés de recalcular toda a malha a cada quadro, pode-se partir da malha anterior A_t e com uma série de refinamentos e simplificações, chegar na malha A_{t+1}, num tempo menor do que se refinássemos a malha a partir da raiz a cada quadro.

2.6.4 Metamorfose geométrica

A cada inserção de um vértice v, ocorre uma alteração da altura da ordem de $(z_{real} - z_v)$. Assim, quando essa diferença é grande, artefatos temporais tendem a ficar visíveis. Tais artefatos são vistos como aparecimento ou desaparecimento súbito de detalhe.

Uma solução para esse problema é utilizar o que se chama de *geomorphing*. Assume-se a posição do vértice como uma função paramétrica $p(t)$ e quando um novo vértice for inserido, ele é inserido na superfície atual em $(t = 0)$ e é gradativamente trazido até sua posição final, quando $(t = 1)$.

3 *Principais trabalhos*

O objetivo deste capítulo é apresentar algumas soluções clássicas, mostrando qual malha e quais métricas foram utilizadas, ilustrando algumas particularidades de cada solução.

3.1 Método ROAM

Proposto por Duchaineau et al. (DUCHAINEAU et al., 1997), este método foi criado para construir malhas adaptáveis utilizando métricas de erro que dependem do observador, produzindo limites de erro bem definidos. Também utiliza coerência quadro-a-quadro para operar com uma alta taxa de quadros por segundo, mesmo com uma alta carga de triângulos.

3.1.1 Malha utilizada

O ROAM utiliza como estrutura a *bintree* triangular (sec. 2.1.1.2). Este método é um método híbrido, pois utiliza tanto refinamento (*top-down*) quanto simplificação (*bottom-up*) no cálculo da malha, criando assim coerência quadro-a-quadro (sec. 2.6.3).

3.1.2 Métricas de erro

As métricas apresentadas se restringem a triangularizações de mapas de alturas. Formalmente, o mapeamento dos vértices no mundo é dado da seguinte forma:

$$
\begin{aligned}
w(v) &= (v_x, v_y, z(v)) \\
&\text{onde} \\
&\qquad (v_x, v_y) \text{ são as coordenadas dos vértices e} \\
&\qquad z(v) \text{ é a altura real em } v
\end{aligned}
$$

3.1.2.1 Filas de prioridades

O método ROAM utiliza as métricas de erro para calcular prioridades para cada triângulo. Estas prioridades são calculadas para gerenciar as filas utilizadas para guiar o processo de refinamento e simplificação. A idéia dessas filas é simples: manter prioridades para simplificação e para refinamento para cada triângulo da malha, começando a partir da raiz, e repetidamente forçar o refinamento ou a simplificação do triângulo com maior prioridade.

Na fila de refinamento, os triângulos são organizados do maior erro para o menor e na fila de simplificação, do menor erro para o maior. Assim, a cada iteração, força-se o refinamento e a simplificação do primeiro elemento de cada fila. Com isso a prioridade máxima (tipicamente um limite de erro) é minimizada. Para tal, é necessário que métrica seja saturada, isto é, as prioridades dos filhos nunca podem ser maiores que as dos seus pais.

3.1.2.2 Erro geométrico

Como medida de erro geométrico o ROAM utiliza o que se chama de ***wedgie***. Um *wedgie* é definido como o volume do mundo que contém pontos (x,y,z) onde (x,y) pertencem a T e $|z - z_T(x,y)| \leq e_T$, para alguma espessura do wedgie $e_T \geq 0$.

Os erros aninhados dos *wedgies* são calculados de maneira *bottom-up*. Assume-se $e_T = 0$ para todos os triângulos no nível mais refinado possível. A espessura do *wedgie* de um triângulo T é calculada em função as espessuras dos *wedgies* de seus filhos, e_{T_0} e e_{T_1}. Assim, calcula-se e_T da seguinte maneira:

$$e_T = \max(e_{T_0}, e_{T_1}) + |z(v_c) - z_T(v_c)|$$

onde

$$z_T(v_c) = (z(v_0) + z(v_1))/2$$

A Figura 9 mostra um exemplo univariado de *wedgies* aninhados, juntamente com a cadeia de *wedgies* que dependem de um determinado vértice v.

3.1.2.3 Erro dependente do observador

A métrica dependente do observador do ROAM, chamada de **distorção geométrica da tela**, é simplesmente um cálculo da distorção geométrica: a distância entre onde cada ponto da

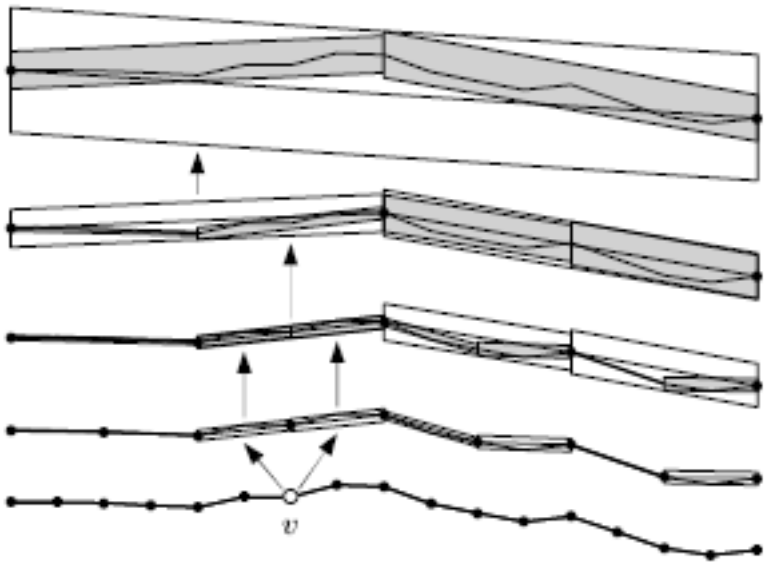


Figura 9: *Wedgies* aninhados em 1D, com os dependentes de v (DUCHAINEAU et al., 1997)

superfície deveria estar no espaço de tela e onde a triangulação toca a tela.

Seja $s(v)$ a posição correta no espaço de tela de um ponto v e $s_T(v)$ sua aproximação pela triangulação T. O erro neste ponto é definido por:

$$dist(v) = \|s(v) - s_T(v)\|_2$$

Na imagem inteira, o erro máximo é dado por $dist_{\max} = \max_{v \in V} dist(v)$, onde V é o conjunto de pontos v cujas posições do mundo $w(v)$ estão dentro do *frustum* de visão (região do espaço que pode aparecer na tela).

Na prática, é calculado um limite superior para a distorção máxima. Para cada triângulo T na triangulação, um limite local é calculado projetando o *wedgie* no espaço de tela. O limite é definido como o maior tamanho das espessuras e_T de todos os pontos $v \in T$ projetadas na tela. Testando este limite calculado contra um valor de corte definido pelo usuário, força-se a manutenção da malha a uma resolução desejada.

3.1.3 Particularidades

Além da métrica de erro, pode-se utilizar como limite de refinamento e simplificação uma quantidade fixa de triângulos. Além disso, o método ROAM permite que o processamento da malha seja parado quando o tempo do quadro acabar. Assim, garante-se a taxa de quadros constante, mesmo que a malha ainda não tenha sido totalmente processada. Além disso, como

utiliza uma métrica de distorção na tela e são feitas poucas alterações na malha entre cada quadro, devido à coerência quadro-a-quadro, o número de artefatos temporais é reduzido.

3.2 Método com *quadtree* restrita

Este método foi proposto por Pajarola em (PAJAROLA, 1998). Tem como objetivo permitir a renderização de massas muito grandes de dados de terrenos, independente do tamanho da memória disponível.

3.2.1 Malha utilizada

A malha utilizada por este método é a *quadtree*. Para garantir a continuidade da malha, é utilizado o grafo de dependências G_{dep} para restringir a malha (sec. 2.1.1.1).

3.2.2 Métricas de erro

Este método propõe uma métrica de erro no espaço do objeto que, operada juntamente com o grafo de dependência, consegue controlar a acurácia da aproximação. O erro $e_T(P)$ de um ponto P e uma triangulação T é a distância euclidiana $d(P,t)$ até o triângulo $t \in T$, com $P_{z=0} \in \Delta(t)$. Ou seja, a projeção de $P = P_{z=0}$ deve estar contido na projeção de $t = \Delta(t)$, ambos projetados no plano xy. Se $P_{z=0} \notin \Delta(t)$ então $d(P,t) = 0$. Consequentemente, o erro da triangulação T é o erro máximo de todos os pontos que não pertencem a T, definidos por $e_T = \max_{\forall P \notin T}(d(P,t))$ e $t \in T$.

Seja L_l o conjunto de todos os pontos do nível l e $L_l^{central}$ o conjunto dos pontos centrais de cada quadrante do nível l. A melhor triangulação não incluindo $L_l^{central}$ é $T_{l-1} = \{G_{dep} \cup L_{l-1}\}$. Ou seja, são todos os pontos do nível L_{l-1}, com as dependências do grafo aplicadas. Já a melhor triangulação não incluindo $L_l \backslash L_{l-1}$ é $T_{l-1} = \{G_{dep} \cup L_{l-1} \cup L_l^{central}\}$. A *cobertura* de um vértice P é o conjunto dos triângulos afetados pela seleção de P:

$$Cov(P \in L_l) = \{t \in T_{l-1} | \Delta(t) \cap P_{z=0} \neq \oslash\}$$

Através da relação de dependência inversa $P \leftarrow Q \Leftrightarrow (Q,P) \in E_{dep}$, definimos o *fecho transitivo* de P como:

$$\zeta(P) = \{Q \in V_{dep} | \exists P_1, \ldots, P_k; P \leftarrow P_1, \ldots, P_k \leftarrow Q\} \cup \{P\}$$

Com isso, definimos o erro de $P \in L_l$ como o erro máximo de todos os pontos em $\zeta(P)$ sobre todos os triângulos em $Cov(P)$

$$e_{t_{l-1}}(P) = \max_{Q \in \zeta(P), t \in Cov(P)} (d(Q,t)).$$

3.2.3 Particularidades

Diferente dos outros métodos, a métrica de erro utilizada não é monótona. Ao se utilizar o fecho transitivo, pode-se visitar não só os nós abaixo na hierarquia da árvore, mas também os anteriores. Dessa maneira, pode-se tratar métricas insaturadas.

3.3 Método SOAR

Este método foi proposto por Lindstrom e Pascucci em (LINDSTROM; PASCUCCI, 2002), estendendo o que foi proposto em (LINDSTROM; PASCUCCI, 2001). Se propõe a ser um ambiente de desenvolvimento (*framework*) para renderização e gerenciamento de massas de dados maiores que a memória principal. Os dois componentes básicos deste método são: refinamento dependente do observador; e um esquema simples de organização dos dados, de forma a otimizar o acesso ao disco.

3.3.1 Malha utilizada

Assim como Duchaineau com o método ROAM, o SOAR também utiliza a *bintree* (sec. 2.1.1.2) neste método. Diferentemente do ROAM, este método é estritamente *top-down*, recalculando a malha a partir da raiz a cada quadro.

3.3.2 Métricas de erro

3.3.2.1 Esferas aninhadas

O critério de refinamento adotado neste método é baseado em **esferas aninhadas**. Um critério semelhante foi proposto em (BLOW, 2000). Cada esfera é centralizada na posição p_i de um vértice i da malha e representa o isocontorno do erro de i projetado no espaço de tela, $\rho = \rho(\varepsilon_i, p_i, e)$, onde ε_i é um erro de i (no espaço do objeto ou do mundo) e e é o ponto de observação. Sendo Ci o conjunto dos descendentes de i, para garantir o aninhamento das esferas e a monotonicidade dos erros, basta que:

$$\rho\left(\varepsilon_i, p_i, e\right) \geq \rho\left(\varepsilon_j, p_j, e\right) \quad \forall j \in C_i$$

Este cálculo é impraticável pois precisa percorrer todos os descendentes de i. Ao invés disso, pode-se utilizar um superconjunto de pontos para projeção definidos por uma bola $B \supseteq P_i$ de raio r, centrada em p_i.

$$B_i = \{x : \|x - p_i\| \leq r_i\}$$

O raio de B é então

$$r_i = \begin{cases} 0, & \text{se } i \text{ é um nó folha} \\ \max_{j \in C_i} \rho\left(\varepsilon_i, x, e\right), & \text{senão} \end{cases}$$

E o erro projetado máximo é o maior erro projetado dos pontos contidos na bola B.

A maneira que ρ e ε são calculados pouco importa nesse critério de refinamento, desde que a monotonicidade seja respeitada.

Para o critério de refinamento proposto, precisamos calcular ρ e ε. Este método propõe duas métricas para o espaço do objeto para o cálculo de ε e uma métrica dependente do observador para o cálculo de ρ.

3.3.2.2 Erro geométrico

Para calcular erros geométrico em mapas de alturas, a medida mais comum é a distância vertical entre o ponto original e o ponto aproximado pela malha. Lindstrom & Pascucci sugerem utilizar o erro incremental ou o erro máximo. O erro incremental oferece uma maneira de se calcular o quanto a malha mudaria removendo um vértice. Já o erro máximo é útil para visualizar o quanto a malha se diferencia da malha mais refinada. Estes erros são definidos para um vértice i em função do conjunto de triângulos T_i que formam diamante com i. Sendo $z_t\left(x_i, y_i\right)$ a altura de um triângulo t no ponto (x_i, y_i). O erro vertical entre i e t é

$$\hat{\delta}_{i,t} = \left|z_i - z_t\left(x_i, y_i\right)\right|$$

O erro incremental então é dado por

$$\hat{\varepsilon}_i^{inc} = \max_{t \in T_i} \left\{ \hat{\delta}_{i,t} \right\} = \left| z_i - \frac{z_0 + z_1}{2} \right|$$

O erro incremental é então o desvio vertical entre i e o meio de sua aresta base (z_0, z_1). Já o erro máximo é a maior distância entre i e todos os descendentes dos triângulos no diamante T_i.

$$\hat{\varepsilon}_i^{\max} = \max \left\{ \hat{\varepsilon}_i^{inc}, \max_{t \in T_i} \max_{j \in D_{i,t}} \left\{ \hat{\delta}_{j,t} \right\} \right\}$$

O erro incremental oferece uma medida de quanto a malha tende se modifica se removermos um vértice. Já o erro máximo diz o quanto a malha desvia da malha com maior nível de definição ao removermos um vértice. É indiferente para o método a escolha do erro incremental ou do erro máximo. Esses erros são calculados e propagados na fase de pré-processamento.

3.3.2.3 Erro dependente do observador

O método SOAR utiliza como métrica de erro dependente do observador a **projeção de erro isotrópica**. Dado um erro ε do objeto, para calcular o erro dependente do observador, basta projetar ε na tela, resultando no erro do espaço de tela $\rho(\varepsilon)$. Para contornar as singularidades da projeção perspectiva, substitui-se a distância sobre a direção de visão pela distância Euclidiana d entre o ponto de visão e e o vértice p. Assim, define-se a métrica como

$$\rho(\varepsilon, p, e) = \lambda \frac{\varepsilon}{\|e - p\|} = \lambda \frac{\varepsilon}{d}$$

Esta é uma medida isotrópica de erro, já que o erro projetado é o mesmo em qualquer direção a uma distância d do vértice. Para a projeção perspectiva, $\lambda = \frac{\omega}{2tan(\varphi/2)}$, onde ω é o número de *pixels* dentro do campo de visão φ. Para simplificação e como a projeção que desejamos é em um plano, uma escolha melhor é assumir $\lambda = \frac{\omega}{\varphi}$. O erro ρ é então comparado com um fator de tolerância τ, definido pelo usuário.

No método, o erro é dado como a maior projeção $\rho(\varepsilon, p, e)$ sobre um conjunto de pontos B (sec. 3.3.2.1).

3.3.3 Particularidades

Este método consegue gerenciar grandes massas de dados através da reorganização dos dados, deixando que o próprio sistema operacional seja responsável pela busca dos dados no disco. Com isso, tem-se maior simplicidade no algoritmo. Mais espeficamente, o arquivo com

as informações do terreno é mapeado em endereços lógicos do sistema e o acesso é feito como se ele estivesse totalmente na memória principal, com o sistema operacional controlando o paginamento dos dados do disco conforme é necessário.

Além disso, ele utiliza a metamorfose geométrica (sec. 2.6.4) como maneira de aumentar o limite de erro sem que artefatos temporais fiquem visíveis.

3.4 Método de mapa de recorte geométrico

Este método foi proposto por Losasso e Hoppe em (LOSASSO; HOPPE, 2004). Com o avanço das GPU's, o número de triângulos processados ultrapassou a casa dos 100 milhões de triângulos por segundo. Este método tem como premissa o fato da adaptação não ser mais essencial, enquanto propõe uma tesselagem mais uniforme do terreno.

3.4.1 Como funciona

Para isso, é utilizado o **mapa de recorte geométrico**, também chamado de *geometry clip-map*. O mapa de recorte geométrico armazena o terreno em um conjunto de malhas regulares centradas no observador. Cada malha representa versões do terreno em resoluções diferentes.

O mapa de recorte geométrico armazena o terreno em uma pirâmide utilizando um conjunto de m níveis, representando extensões do terreno em diferentes resoluções. Cada nível contém um conjunto de $n \times n$ vértices. Cada nível é acessado de maneira toroidal, permitindo atualizações incrementais eficientes. Para cada nível l do mapa de recorte, são definidas algumas regiões retangulares. A **região de recorte** é a extensão do mundo da malha $n \times n$ armazenada naquele nível. A **região ativa** é a área que desejamos renderizar. Durante a movimentação do observador, o mapa de recorte é atualizado movimentando a região de recorte de maneira que ela contenha a região ativa desejada. Também há a **região de renderização**, que é a área entre a região ativa do nível $l+1$ e a região ativa do nível l (Fig. 10).

O refinamento dependente do observador é feito pela seleção das regiões ativas de cada nível do mapa de recorte. De acordo com a profundidade do espaço de tela, é possível encontrar um valor limite para o tamanho dos triângulos. Este valor é calculado com o observador na horizontal. Caso o observador esteja olhando para o terreno de cima para baixo, o tamanho dos triângulos tendem a diminuir e há o aparecimento de irregularidades na visualização. Este problema é contornado não renderizando níveis mais refinados desnecessários.

Ao invés de posicionar o observador no centro da região ativa, pode-se também adaptar o

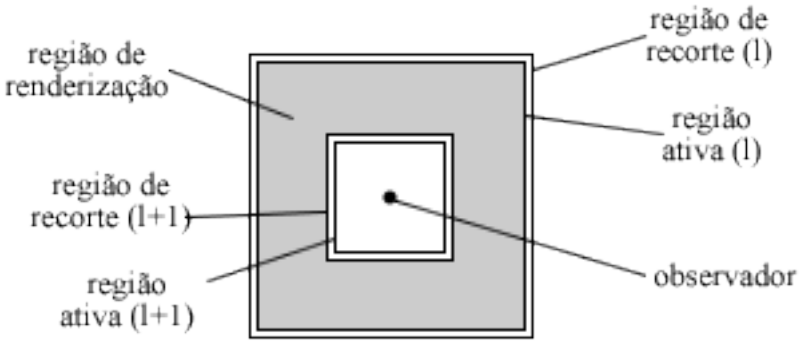


Figura 10: Regiões definidas do mapa de recorte

local e o tamanho de cada região de acordo com o *frustum*. Mas a opção da posição central permite que a rotação seja feita de maneira instantânea, o que pode ser desejável para algumas aplicações.

3.4.2 Particularidades

Ao invés de se apoiar na adaptação da malha, este método cria uma tesselação (cobertura de um plano com objetos planos) mais suave. Além disso, é proposta uma compressão dos dados. Esta compressão consegue diminuir o tamanho dos dados necessários em um fator de 100, o que faz com que mesmo grandes terrenos sejam mapeados completamente na memória. Também há a utilização constante da GPU, já que cada nível da pirâmide é armazenada como um *buffer* de vértices na memória de vídeo.

4 *Desenvolvimento e resultados comparativos*

Este capítulo se propõe a comparar dois dos métodos apresentados. Os métodos escolhidos foram o ROAM e o SOAR. Estes dois métodos foram escolhidos porque utilizam a mesma malha, a *bintree*. Para a comparação foi implementado um visualizador simples, onde podemos analisar o resultado visual da métrica de cada erro.

4.1 Desenvolvimento

Para compararmos as soluções apresentadas, foi desenvolvido um modelo computacional para visualização de terrenos. Primeiramente, uma malha foi implementada para que fosse possível acoplar as métricas das soluções de maneira simples. A classe ABDR foi gerada para atender esse pré-requisito. Assim, somente a métrica pode ser utilizada como fator de comparação entre os métodos.

Na geração da *bintree* foi escolhido a divisão por largura. Assim, todos os triângulos de um nível são processados antes de se processar o nível seguinte (vale citar que nem todos os triângulos processados são refinados - isso ocorre somente se a operação for necessária).

Os parâmetros das funções foram omitidos para facilitar a visualização da estrutura da classe:

```
class ABDR {
  public:
   int num_nodos, ultimo;
 NODO raiz;
 NODO *aloca;
 int  ntriangulos;
```

```
  private:
  void dividir_nodo(...);
   void bissecta_triangulo(...);

  public:
    ABDR(...);
    ~ABDR();
   int tesselar(...);
   void triangulos(...);
};
```

Os métodos principais da ABDR são:

- `dividir_nodo(...)`: Método responsável pelo controle da divisão do triângulo. Este método é responsável pelo controle das divisões forçadas.
- `bissecta_triangulo(...)`: Método que divide efetivamente um triângulo, através da bisseção da maior aresta, atualizando todos os ponteiros necessários.
- `tesselar(...)`: Método que inicia a adaptação, controlando quando um triângulo é refinado ou não.
- `triangulos(...)`: Faz a preparação dos triângulos para saída, após a adaptação.

Além destes métodos, temos ainda o vetor `*aloca` que mantém todos os nós existentes da árvore, de forma que a alocação dos nós seja contígua, o que ajuda na não fragmentação dos dados pela memória, e a variável `ntriangulos` que é responsável pela limitação do número de triângulos por quadro. Estes elementos são públicos para que o acesso a eles seja mais rápido, sem a sobrecarga de chamadas extras a funções.

Duas estruturas auxiliares também são utilizadas. As suas definições são:

```
typedef struct Parametro {
float u, v, w;
} PARAM;

typedef struct Nodo {
PARAM vapex, vesq, vdir;
```

```
int nivel, indice;
struct Nodo *f_esq, *f_dir, *pai;
struct Nodo *t_esq, *t_dir, *t_base;
} NODO;
```

A estrutura Parametro armazena cada parâmetro da ABDR e a estrutura Nodo é responsável por armazenar toda a informação de cada triângulo. A estrutura Nodo é exemplificada a seguir pela Figura 11.

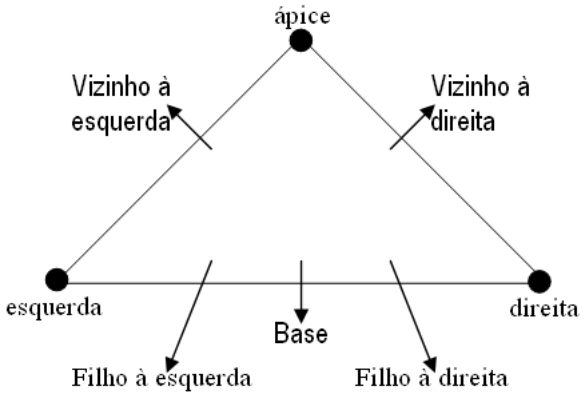


Figura 11: Estrutura do nodo da *bintree*

A adaptação da malha é feita exclusivamente *top-down* e não foram utilizadas as filas de prioridade do método ROAM. Dessa forma, a métrica do método ROAM não é utilizada para cálculo de prioridades (sec. 3.1.2.1) e sim para decisão sobre o refinamento de cada triângulo. Além disso, como o método ROAM exige que todos os dados estejam na memória principal, o método SOAR também não teve implementado a sua função de gerenciar dados fora da memória principal. Para conseguirmos um aumento de desempenho foi implementado a remoção de faces ocultas.

4.2 Resultados comparativos

Todos os testes apresentados foram feitos em um computador Athlon XP 2000+, com 512 MB de RAM, placa de vídeo ATI Radeon® X1600Pro 256MB. Construiu-se um visualizador para todo o planeta sobre uma esfera. Os exemplos a seguir foram gerados a partir da região do Himalaia. Essa região foi escolhida por ser muito montanhosa, facilitando a percepção das diferenças dos métodos avaliados.

A Figura 12 mostra como a malha se adapta conforme nos aproximamos utilizando o método ROAM. A Figura 13 mostra a mesma aproximação, porém com o método SOAR, sem-

pre com o erro dependente do observador. A Figura 14 mostra a diferença entre utilizar o erro geométrico e o erro dependente do observador com o método ROAM. O erro dependente do observador utiliza mais triângulos para as áreas mais próximas do observador, o que garante maior qualidade em áreas onde realmente interessam. A Figura 15 mostra o mesmo caso, dessa vez utilizando o método SOAR. Nas Figuras 12 e 13 pode-se observar o incremento no número de triângulos conforme nos aproximamos do Himalaia. Os parâmetros utilizados como limites para cada método foram:

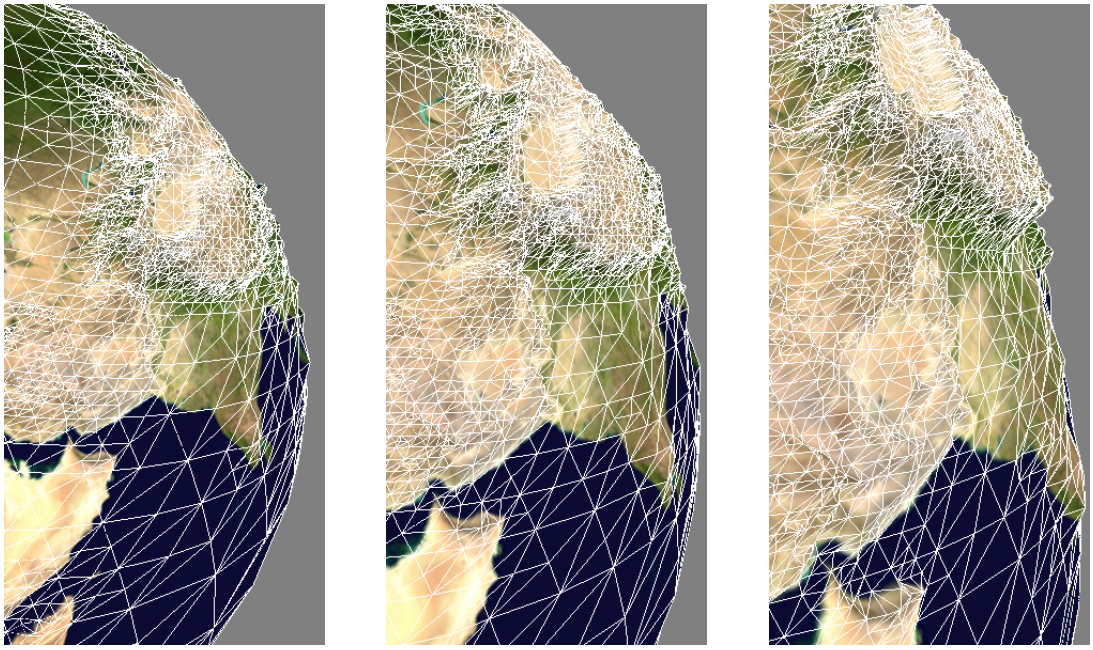
- SOAR: Erro dependente do observador
  $\tau = 0.2$,
  campo de visão $= 45$graus e
  largura da tela $= 800$*pixels*.
  Erro geométrico
  limite $= 1$

- ROAM: Erro dependente do observador
  $\text{dist}_{\text{max}} = 1.0$.
  Erro geométrico
  limite $= 120$

Comparando as duas estruturas em função desses parâmetros, o método SOAR apresenta uma métrica capaz de adaptar a malha mais eficientemente do que a do ROAM, gerando menos triângulos, porém mantendo bom aspecto visual.

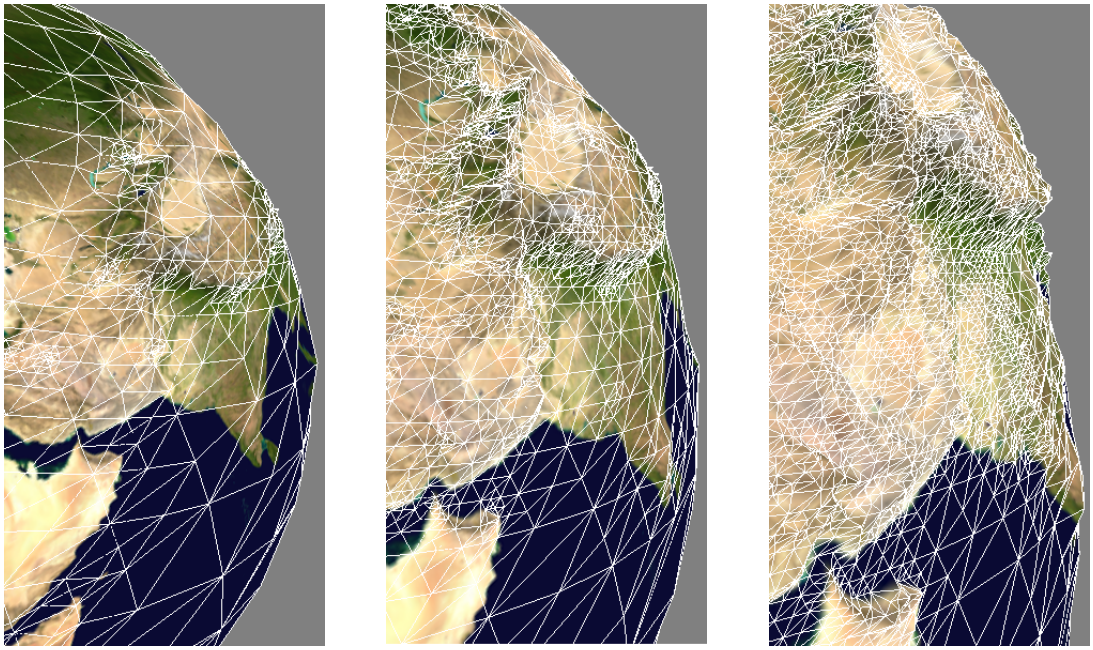
As figuras 14 e 15 apresentam as diferenças entre o refinamento feito com a métrica dependente do observador e com somente o erro geométrico. No método ROAM, o erro geométrico utilizou menos triângulos, mas a simplificação não é dependente do observador, enquanto o SOAR teve um aumento no número de triângulos(sec. 3.3.2.2).

Um último teste foi feito alterando-se o limite contra o qual cada método é testado. Em ambos os métodos o resultado foi o esperado. Ao diminuirmos esse limite máximo de erro, houve um aumento significativo no número de triângulos e o terreno ficou bem mais refinado. Já quando o limite foi aumentado, houve queda no número de triângulos e a qualidade visual do terreno diminuiu. As figuras 16 e 17 mostram os resultados do teste.



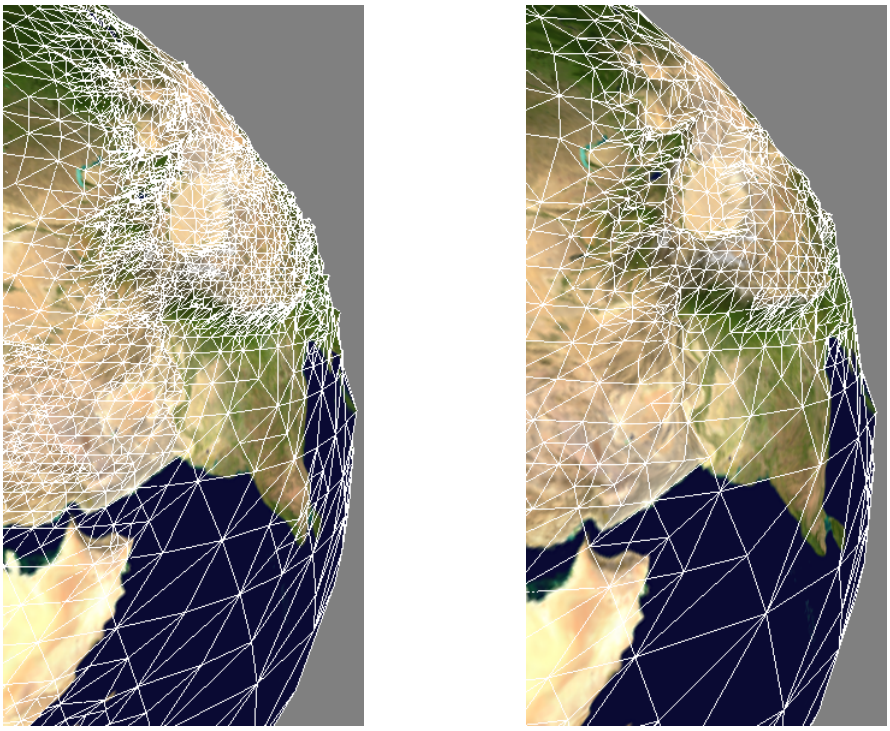
(a) 17259 triângulos (b) 37530 triângulos (c) 106480 triângulos

Figura 12: *Diferentes níveis de detalhe com o método ROAM*



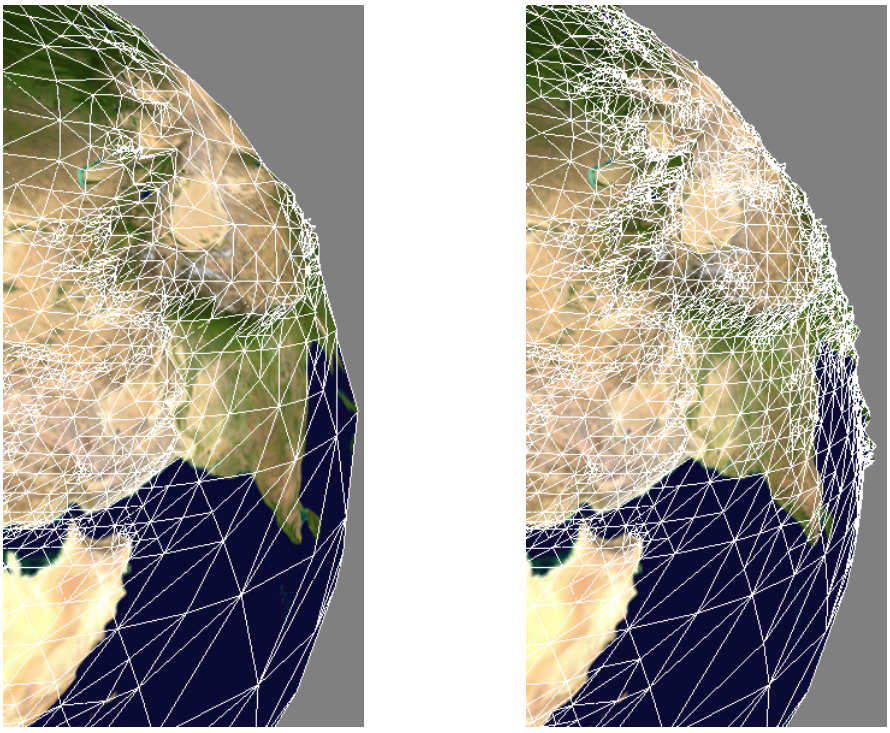
(a) 2806 triângulos (b) 10445 triângulos (c) 43845 triângulos

Figura 13: *Diferentes níveis de detalhe com o método SOAR*



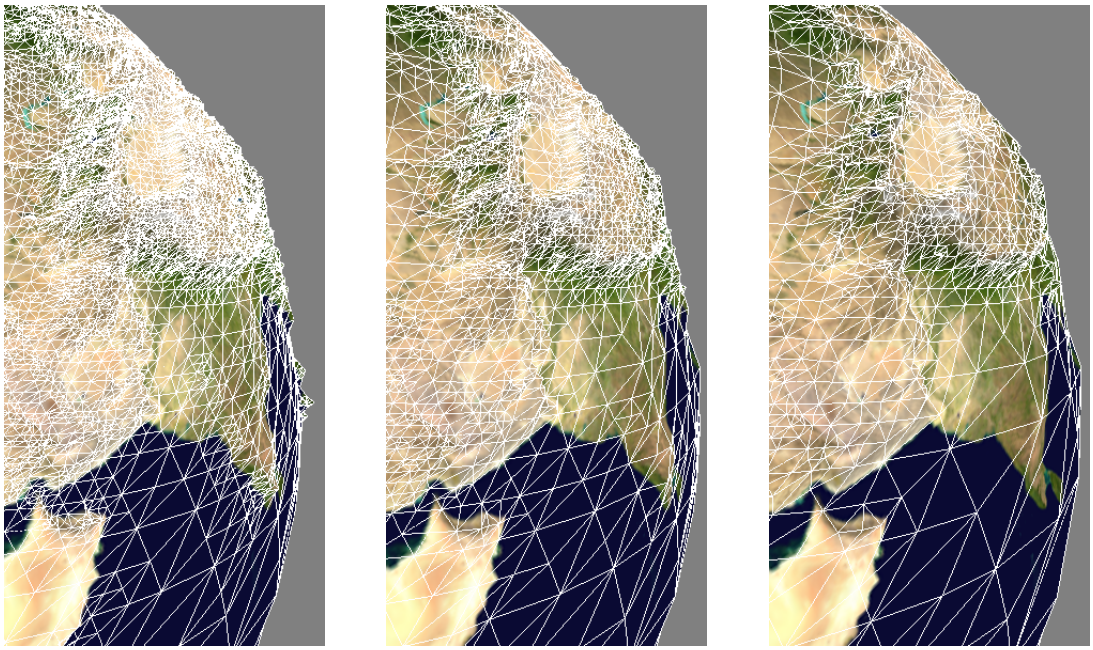
(a) 23100 triângulos (b) 8132 triângulos

Figura 14: *ROAM com (a) métrica dependente do observador e (b) com erro geométrico*



(a) 6431 triângulos (b) 39157 triângulos

Figura 15: *SOAR com (a) métrica dependente do observador e (b) com erro geométrico*

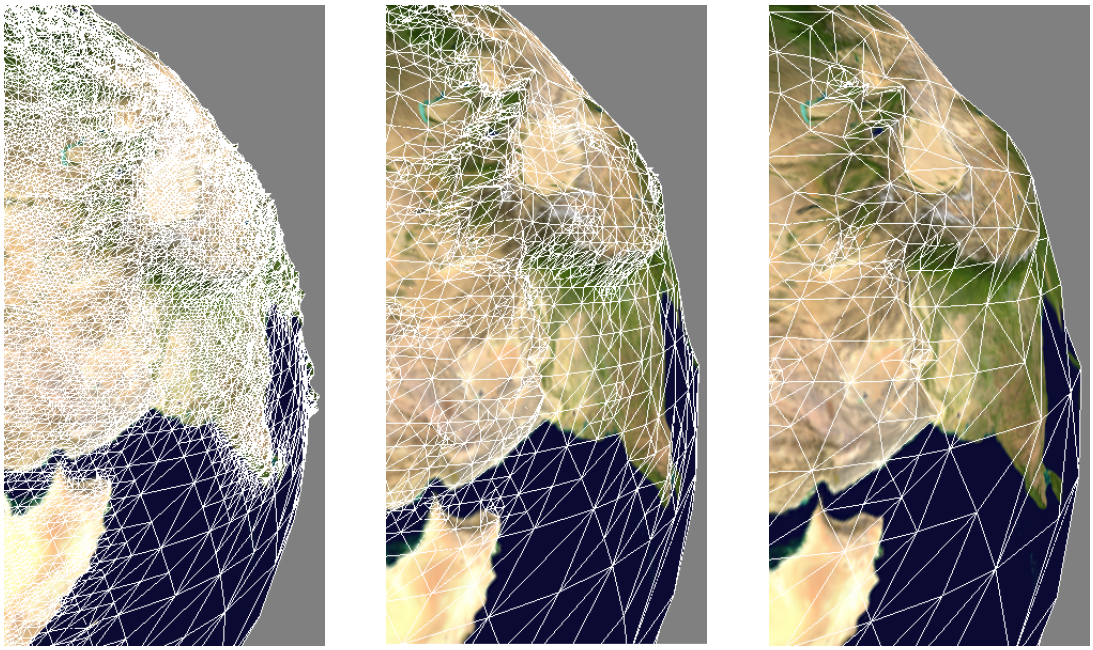


(a) 90357 triângulos. Limite = 0.5

(b) 37530 triângulos. Limite = 1

(c) 15382 triângulos. Limite = 2

Figura 16: *Diferentes limites máximos de erro no método ROAM*



(a) 92488 triângulos. Limite = 0.1

(b) 10445 triângulos. Limite = 0.2

(c) 2720 triângulos. Limite = 0.3

Figura 17: *Diferentes limites máximos de erro no método SOAR*

5 *Conclusão*

Neste trabalho, foram apresentados fundamentos básicos para construção de um esquema de adaptação de terrenos e as principais soluções clássicas foram estudadas. O objetivo de desenvolver um visualizador de terrenos que permita a comparação dos métodos ROAM e SOAR foi alcançado com alguma dificuldade. Ainda assim, os resultados foram satisfatórios, já que o visualizador se mostrou funcional e permitiu uma boa análise comparativa das soluções utilizadas. A pesquisa de adaptação por nível de detalhe se desenvolveu bastante nos últimos anos. O avanço do *hardware* gráfico oferece soluções que cada vez mais reproduzem com maior fidelidade um terreno real. Ao contrário do que se imagina, processadores gráficos programáveis e eficientes abrem ainda mais o campo de pesquisa sobre o tema deste trabalho.

Algumas tarefas não são simples de serem feitas, como gerenciar dados fora da memória de vídeo e da memória principal, por exemplo. Além disso, é necessário uma boa implementação para que se consiga taxas interativas de quadros.

Como trabalhamos com um espaço discreto, é importante a atenção com o erro calculado. Somente em casos especiais podemos reconstruir fielmente o terreno real. Conseqüentemente a utilização das métricas merece atenção. O cálculo do erro deve levar em consideração relações geométricas da cena para que o refinamento ou a simplificação sejam feitas de maneira coerente e que a aproximação seja o mais fiel possível ao original. Ou seja, que o resultado tenda a minimizar globalmente os erros da adaptação.

A pesquisa de adaptação por nível de detalhe para terrenos está em constante evolução e apresenta muitas maneiras de se resolver esse problema. Uma proposta de trabalho futuro é otimizar a ABDR, para que possamos trabalhar com gerenciamento de dados, para utilizarmos com massas de dados muito extensas ou de forma distribuída. Uma outra proposta interessante é trabalhar com processamento na GPU, já que as GPU's atuais fornecem bom desempenho.

Referências

BLOW, J. Terrain rendering at high levels of detail. In: *Proceedings of the 2000 Game Developers Conference*. [S.l.: s.n.], 2000.

DUCHAINEAU, M. et al. Roaming terrain: Real-time optimally adapting meshes. In: *Visualization '97., Proceedings*. [S.l.: s.n.], 1997. p. 81–88.

LINDSTROM, P.; PASCUCCI, V. Visualization of large terrains made easy. In: *Visualization, 2001. VIS '01. Proceedings*. [S.l.: s.n.], 2001. p. 363–574.

LINDSTROM, P.; PASCUCCI, V. Terrain simplification simplified: a general framework for view-dependent out-of-core visualization. *Visualization and Computer Graphics, IEEE Transactions on*, v. 8, n. 3, p. 239–254, July-Sept. 2002.

LOSASSO, F.; HOPPE, H. Geometry clipmaps: terrain rendering using nested regular grids. In: *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*. New York, NY, USA: ACM Press, 2004. p. 769–776.

LUEBKE, D. et al. *Level of Detail for 3D Graphics*. 1. ed. [S.l.]: Morgan Kaufmann, 2003.

PAJAROLA, R. Large scale terrain visualization using the restricted quadtree triangulation. In: *Visualization '98. Proceedings*. [S.l.: s.n.], 1998. p. 19–26,515.

STEPHEN, E.; HASBROUCK, H. *Landscape Modeling: Digital Techniques for Landscape Visualization*. 1. ed. [S.l.]: McGraw-Hill, 2001.