5
Renderização de terreno usando processo local paralelo em GPU

5.1
Introdução

Renderização de terrenos com taxas interativas é essencial para aplicações GIS, simuladores de vôo, simuladores de veículos terrestres e jogos. Contudo, apesar dos avanços na última década, alcançar o equilíbrio entre alta qualidade de imagem e bom tempo de processamento é ainda um desafio. Um estudo recente [Asirvatham and Hoppe, 2005] atesta uma taxa de 90 fps em um vôo interativo sobre um mapa de altura com mais de 20 bilhões de amostras (216.000 x 93.600), porém, severas restrições são aplicadas. Extrema fidelidade visual e LOD automático são alguns dos recursos disponibilizados pelo tessellator para a nova geração de placas gráficas, porém, o uso do tessellator não é trivial e técnicas novas de LOD são necessárias.

Terrenos são tipos de modelos representados por mapas de altura - um grupo de amostragens de altura sobre um domínio plano. Uma forma simples de renderização de terreno é criar uma malha de polígonos regulares e mover seus vértices de acordo com um mapa de altura, o que não é eficiente nem escalável.

Algoritmos para uma renderização eficiente de terrenos já foram propostos por mais de uma década, a maioria envolvendo estruturas hierárquicas associadas com LOD e técnicas de descarte de geometria. Bons estudos sobre o assunto podem ser encontrados em [Floriani et al., 1996] e [Pajarola and Gobbetti, 2007].

A avaliação é necessária localmente, pois uma parte do terreno pode ter uma altura constante e não é preciso subdividi-la. Contudo, os vértices devem ser reduzidos de forma adequada, pela possibilidade de porções de terreno apresentarem variação de altura em alta frequência - o que exige uma malha refinada para uma representação confiável. A malha do terreno deve ser adaptada automaticamente para ser refinada ou não, de acordo com o ponto de vista da câmera. Isso exige um controle de LOD de acordo com o ponto de

vista do observador e que mantenha a conectividade consistente (ou seja, sem quebras na malha) e animação suave (sem artefatos salientes ou saltitantes). Porém, a maioria das técnicas de divisão de vértices e colapso de arestas (predominante em técnicas de LOD) exigem algoritmos de CPU sequenciais para atualização de suas estruturas de dados e, como consequência, são difícies de serem implementadas eficientemente, apesar de algumas exceções, como em [Losasso, 2004]. Há também a possibilidade de utilizar o Geometry Shader, que tem acesso aos vértices vizinhos das primitivas e pode atualizar estruturas de dados que estão na memória de vídeo. Todavia, essas abordagens são muito complexas, pois a continuidade da malha precisa ser preservada.

Controle de LOD paralelo de acordo com o observador baseado no uso de Geometry Shaders foi proposto recentemente para malhas arbitrárias [Hu et al., 2010b]. Esse controle pode ser adaptado para terrenos, mas a complexidade da solução não é reduzida significativamente.

Escalabilidade é outro ponto crítico quando se trata de grandes mapas de altura. Nesse caso, durante uma navegação aproximada a malha deve ter o máximo de refinamento possível (um quadrado para cada amostra), causando um overhead crítico. Algoritmos de renderização de terrenos devem apresentar alta escalabilidade. É indispensável a utilização de alguma estrutura hierárquica para navegar em um terreno com escalas altamente variáveis. Entretanto, processos de construção e acesso a essas estruturas de dados significam menor eficiência computacional e, muitas vezes, um obstáculo para a comunicação entre CPU e GPU. Outra forma de escalabilidade se refere à capacidade de algoritmos paralelos usarem o máximo do número de unidades de processamento que se tornaram disponíveis em consecutivas gerações de GPU.

Neste capítulo apresentamos uma nova técnica para uma eficiente renderização de terrenos usando uma técnica de LOD contínua e dependente do observador com base no uso do tessellator. Nossa técnica é baseada em um processamento local paralelo, ou seja, os resultados de um patch do terreno não dependem dos resultados obtidos em outros patches. A técnica proposta não usa estrutura hierárquica para fácil implementação em GPU.

5.2 Trabalhos Relacionados

Técnicas de multirresolução para renderização de terrenos podem ser classificadas através de três classes básicas:

1. (i) Modelos de Multirresolução através de malhas irregulares [Floriani et al., 1996], e

2. (ii) Modelos de multirresolução que exploram certa semi-regularidade de dados [Pajarola and Gobbetti, 2007].

3. (iii) Modelos de multirresolução híbrida com estrutura regular para malhas irregulares [Toledo et al., 2001].

Seguindo a classificação proposta por [Pajarola and Gobbetti, 2007], a segunda classe de modelos pode ser agrupada em três linhas:

1. Triangulações diretas [Falby et al., 1993] e malhas regulares sobrepostas [Losasso, 2004] e [Asirvatham and Hoppe, 2005],

2. Triangulações baseadas em árvores [Lindstrom et al., 1996] e [Duchaineau et al., 1997].

3. Triangulações agrupadas em cluster [Schneider and Westermann, 2006] e [Levenberg, 2002].

A primeira linha utiliza grids regulares, o que a torna escalável, simples de implementar e feita sob medida para hardwares gráficos. A segunda linha utiliza malhas com conectividade semi-regular, que se baseia em estruturas de dados mais poderosas. A terceira trabalha com porções contínuas da malha, regulares ou semi-regulares, formando patches quadrados ou triangulares. Essas partes são teceladas dentro da GPU com uma comunicação mínima com a CPU.

O modelo proposto neste artigo utiliza um grid 2D regular de patches quadrados (chamados tiles) que são tecelados pela GPU de forma independente. Esta abordagem classifica o modelo proposto como uma triangulação em cluster, porém com características de simplicidade encontradas em triangulações diretas. A principal diferença entre nosso trabalho e todas as outras técnicas é que nosso sistema é o primeiro a fazer renderização de terrenos com LOD em tempo real sem utilizar nenhum tipo de estrutura hierárquica de dados.

Estratégias de LOD não específicas para terrenos também são aplicadas para casos em terrenos. Muitas dessas estratégias contam com o Geometry Shader, e podem ser muito eficientes. Há métodos baseados em GPU [Hu et al., 2010b],[Decoro and Tatarchuk, 2007] e [Ji et al., 2006]. Apesar dessas estratégias baseadas em GPUs serem muito eficientes, elas contam com estruturas hierárquicas complexas e condições para atualizar a malha e evitar fraturas nela.

A maioria das novas propostas para modelos baseados em multirresolução na GPU evitam a fase de pré-processamento, mas ainda exigem o percorrimento de toda a estrutura a cada frame. Uma notável exceção é o trabalho

realizado por [Hu et al., 2010b], que, de qualquer forma, usa uma estrutura de dados hierárquica - o que faz dele mais complexo do que nosso sistema. Percorrer toda a estrutura de dados a cada frame não é apropriado quando o terreno muda dinamicamente - um problema crítico em jogos 3D. Em nosso modelo, o terreno pode ser deformado dinamicamente por manipulação direta do mapa de altura sem nenhuma sobrecarga extra para o sistema.

Apesar de todos os avanços prévios de tecelagem em hardware, pesquisadores de técnicas de LOD para terrenos não parecem estar cientes dos problemas enfrentados pelos desenvolvedores de jogos para utilizar isso funcionalmente. Acreditamos que a razão para esta falta de popularidade são os problemas popping de artefatos que são causados devido ao uso trivial e direto do hardware de tecelagem. Nossa técnica propõe um modelo para superar estas dificuldades através de controles simples e critérios de erro. Podemos encontrar alguma similaridade de nosso critério com técnicas propostas por [Tatarchuk et al., 2010]. No entanto, nossa abordagem é mais ampla do que a encontrada nesta referência, pois calculamos uma subdivisão mínima e máxima para as arestas e também para o centro de cada patch.

Nosso modelo estabelece um framework claro para o uso de renderização de terrenos com LOD por processo local paralelo.

No melhor de nosso conhecimento, nenhum outro trabalho atual na literatura propõe um meio de implementar renderização de terreno robusta e eficiente com LOD e sem uso de estruturas de dados hierárquicas. Além disso, nenhum outro trabalho estabelece uma estrutura clara para processamento local paralelo na renderização de terrenos.

5.3 Visão Geral

Processar tiles de forma contínua e em uma abordagem view-dependent é um problema desafiador, especialmente se estas malhas não puderem ser refinadas do mesmo modo. Devemos garantir a continuidade da malha. Arestas de tiles adjacentes devem ter o mesmo número de subdivisões, sem levar em consideração o número de sub-quads que tem cada tile adjacente.

Consequentemente, a idéia é tecelar o tile independentemente dos resultados dos outros tiles, estabelecendo um verdadeiro processamento local paralelo.

Podemos alcançar essa independência definindo uma malha regular na qual uma aresta compartilhada entre dois tiles tem o mesmo valor do fator de tecelagem (chamado TessFactor).

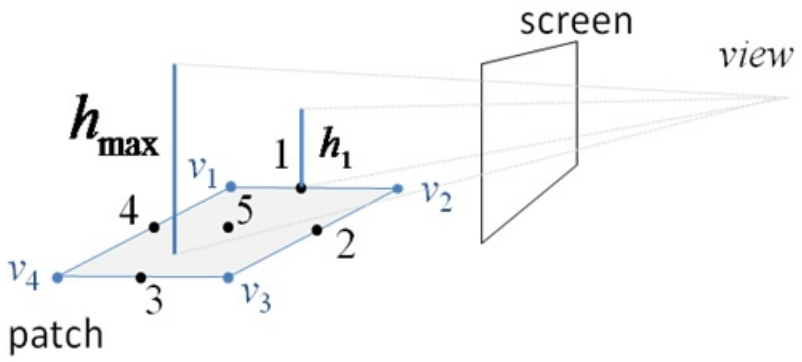

Figura 5.1: Pontos de referência de mosaico (quatro pontos no meio dos contornos e um no centro do pedaço). V_i é um vértice do patch.

Definimos pontos de referência de tecelagem Figura 5.1 e o fator de tecelagem para estes pontos, dado por:

$$T_i = f(view, \nabla^2 h) \quad i = 1, .., 4 \tag{5-1}$$

onde h_i é o valor de altura correspondendo ao ponto $i, view$ é a posição da câmera e $\nabla^2 h$ é a derivada de segunda ordem do mapa de altura $h(u, v)$ correspondendo ao tile. A posição de um ponto de referência na aresta corresponde ao centro da aresta. O quinto ponto está no centro do tile e sua tecelagem é dada por: $T_5 = f(h_{max}, view, \nabla^2 h)$

onde h_{max} é o valor máximo de altura para os modelos dentro do tile.

Um dos princípios por trás da equação 5-1 é que a relação entre h_i e view deve estabelecer um valor mínimo de TessFactor (T_{min}) que garante uma tecelagem precisa do ponto de vista da câmera. Por exemplo, se a câmera está muito longe do patch e h_i está baixo, nenhuma subdivisão será necessária em torno do ponto i. Outro princípio que ronda a equação 5-1 é que $\nabla^2 h$ pode ser utilizado para estabelecer um limite superior para o TessFactor (T_{max}) e garantir que os patches não sejam divididos em excesso. Por exemplo, nenhuma subdivisão é necessária para um terreno plano ($\nabla^2 h = 0$). Em outro aspecto, penhascos íngremes exigem altos fatores de tecelagem. De acordo com esses princípios, a equação 5-1 deve calcular um valor de TessFactor no intervalo $[T_{min}, T_{max}]$.

O cálculo do TessFactor de uma aresta depende da função $\nabla^2 h$ do tile adjacente. Contudo, esta restrição não destrói a natureza do processo local no método proposto. Além disso, consideramos algumas simplificações na equação 5-1 que transformam o cálculo de influência de $\nabla^2 h$ em um simples

procedimento.

5.4 Análise do mapa de altura

5.4.1 Idetinficação de fraturas

Para ser representado, um terreno plano necessitaria apenas de uma única primitiva. No entanto, terrenos reais têm irregularidades (como em penhascos íngremes) que necessitam de mais geometria para serem renderizados corretamente. Neste documento, chamamos estas irregularidades de "fratura".

O processo para encontrar as fraturas consiste em fazer a média das derivadas do mapa de altura nas direções horizontais e verticais Figura (5.2). O cálculo Laplaciano $\nabla^2 h$ não é aceitável porque a Laplaciana é extremamente sensível a ruídos. Além disso, desejamos um modo mais simplificado para levar em conta as fraturas do terreno. Propomos uma combinação de dois operadores Sobel. A magnitude do vetor gradiente calculado pelo operador Sobel sobre o mapa de altura representa a variação de altura em um pixel dado. Contudo, o gradiente não é suficiente para determinar uma fratura.

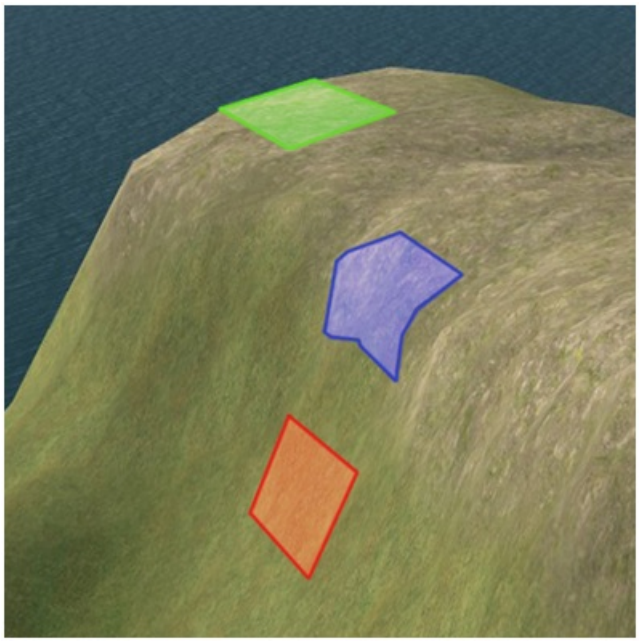

Figura 5.2: A área no topo possui primeira e segunda derivada pequenas. A parte inferior tem maior valor de primeira derivada e menor de segunda derivada. Já a área do meio apresenta maior segunda derivada. A segunda derivada denota locais que exigem refinamento.

Locais que devem ser interpretados como fraturas apresentam mudança na variação de altura. No algoritmo proposto, estimamos este valor aplicando

um operador Sobel sobre o campo de gradiente obtido através da aplicação do primeiro operador Sobel. Os novos valores de inclinação podem ser visualizados como um mapa de aceleração da altura (height acceleration map) ou HAM, que pode ser utilizado para identificar fraturas. Na fase de renderização, nosso algoritmo precisa consultar acelerações de altura com o objetivo de determinar o refinamento máximo que será detalhado mais adiante.

5.4.2 Tamanho do tile

A dimensão do tile é determinada pelo usuário, que decide quantas amostras do mapa de altura um sub-tile deve cobrir. O caso mais refinado seria quando tivermos um quad para cada amostra. Em casos reais, um mapa de altura nem sempre exige que toda amostra tenha um quad para representá-la. Porém, se a densidade de divisões é maior que a densidade de vértices da área tecelada, podem ocorrer algumas impressões indesejadas e artefatos. Estes efeitos indesejados sempre ocorrem quando a frequência de amostragem (vértices do quad) é muito diferente da frequência de sinal (densidade de texels).

Em nosso modelo, eliminamos esses problemas considerando que um quad sempre cobre um texel.

5.5 View-dependent LOD

5.5.1 Refinamento máximo: T_{max}

Nossa primeira preocupação é não tecelar demais partes que não necessitam de refinamento. Intuitivamente, um terreno plano não precisa de subdivisão, logo, todo tile que contêm uma parte quase plana do terreno não deve ser subdividido quando a câmera se aproxima. Partes planas podem ser identificadas pelo mapa de aceleração de altura (HAM). Além disso, precisamos identificar fraturas associada aos tiles também consultando o HAM. Nesses dois casos, nós não precisamos de uma expressão exata para a equação 5-1, mas somente uma proproção razoável entre a aceleração e o TessFactor. Propomos a equação 5-2.

$$T_{max} = g(HAM_{max}) \tag{5-2}$$

Onde HAM_{max} é o valor máximo para aceleração de altura associada ao tile e g pode ser uma função logaritma ou a função de raiz quadrada. A Figura 5.3 ilustra a equação 5-2. Na prática, HAM_{max} está dentro do intervalo que não contém os extremos do domínio

A equação 5-2 não oferece nenhum obstáculo à complexidade do algoritmo proposto. Quando o ponto de referência de tecelagem está em uma aresta, o algoritmo seleciona o valor de T_{max} entre dois tiles adjacentes.

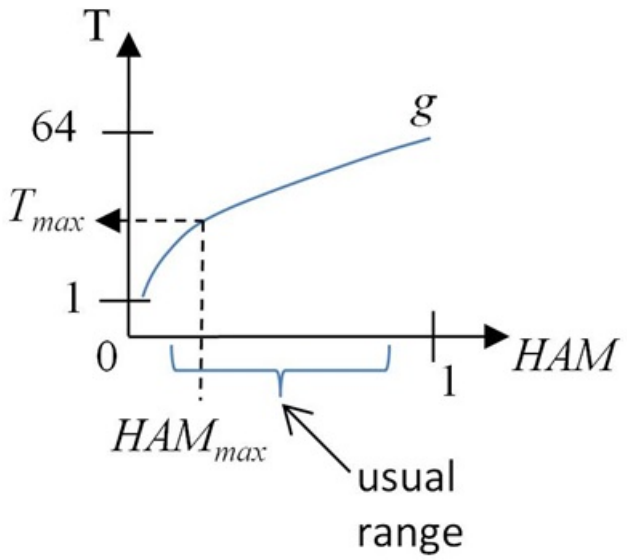

Figura 5.3: TessFactor como uma função de HAM. g não é definido por $T < 1$, porque o intervalo do TessFactor é $[1, 64]$.

5.5.2
Refinamento mínimo: T_{min}

Outro problema a ser resolvido é o refinamento minimo requerido quando se está a uma longa distância. Um exemplo real é uma parte do terreno que contém um pico. Um pico não pode desaparecer no horizonte, mesmo que a câmera esteja muito longe dele. Portanto, precisamos estabelecer um refinamento mínimo por tile (T_{min}) baseado no ângulo entre a câmera e a normal do tile.

T_{min} é obtido a partir do controle do erro de aproximação. É comum na literatura avaliar o erro de aproximação de uma malha tesselada através de seu desvio no vértice. [Duchaineau et al., 1997] e [Lindstrom and Pascucci, 2002] mediram este desvio no espaço projetado. Nós também utilizamos o espaço projetado, mas de um modo diferente. A Figura **??** ilustra o processo proposto, onde sucessivamente interpolamos o ponto que deve corresponder à maior altura h abrigada pelo tile. Devemos ressaltar que este valor máximo

(h) não necessariamente ocorre em um dos pontos de referência de tecelagem Figura 5.4. Em cada fator de tecelagem, o ponto move um pequeno passo em direção ao valor correto. O desvio é monitorado no espaço projetado e o processo acaba quando o desvio está menor do que o erro permitido ϵ . Se o processo é recursivo, a profundidade da recursão é o valor de T_{min}.

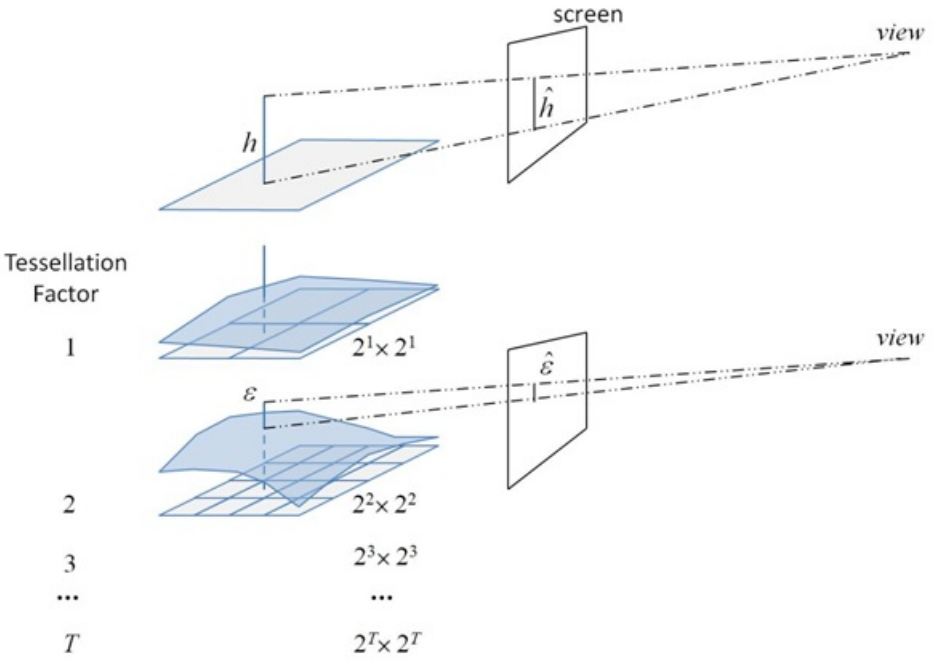

Figura 5.4: Processo de cálculo do T_{min} que para quando o erro projetado E é igual ao erro permitido ϵ. h é o valor máximo de altura.

De qualquer forma, o processo recursivo não é necessário, pois uma boa estimativa de T_{min} pode ser obtida observando que $\epsilon/\hat{h}$ é proporcional a 2^T . Sem perda de generalidade podemos assumir que

$$\epsilon/\hat{h} = 2^T \tag{5-3}$$

E substituir a equação 5-1 pela seguinte equação:

$$T_{min} = \log_2 \epsilon/\hat{h} \tag{5-4}$$

5.5.3 Distância da câmera

Agora, o valor de TessFactor para um ponto de referência de tecelagem i é obtido a partir do cálculo baseado na distância da câmera d, mantendo-

os dentro do limite entre valores máximos e mínimos. Aqui assumidos uma proporcionalidade inversa entre TessFactors e distâncias da câmera. Além disso, supomos que há um valor de distância de câmera (d_{max}) que se $d < d_{max}$, então T deve ser igual a T_{max}. Considerando essas suposições, propomos a seguinte equação para substituir a equação 5-1.

$$T = \frac{T_{max} {d_{max}}^2}{d^2} \qquad se \;\; d \in [d_{max}, d_{min}] \tag{5-5}$$

$$T = T_{max} \qquad se \;\; d < d_{max} \tag{5-6}$$

$$d_{min} = \frac{\sqrt{T_{max}}}{\sqrt{T_{min}}} d_{max} \tag{5-7}$$

A figura 5.5 ilustra as equações 5-5, 5-6, 5-7.

5.6 Implementação

Para ser tecelada posteriormente, uma malha não refinada precisa ser transferida para a GPU. O tamanho do grid é calculado com base no tamanho do tile como relatado anteriormente.

Além da posição, precisamos também transferir as coordenadas de textura do height map correspondendo a cada vértice do quad.

5.6.1 Mapa de aceleração de altura

Dado um terreno representado por um mapa de altura H, nosso primeiro objetivo é encontrar a taxa de mudança de cada texel em H. Para obter este valor, utilizamos uma abordagem baseada no operador [Sobel and Feldman, 1968].

Dois Kernels 3x3 são convoluídos com H para calcular aproximações de derivativas - um para mudanças horizontais e outro para verticais. Assumindo que e são as aproximações das derivadas horizontais e verticais correspondentes, o cálculo é como segue:

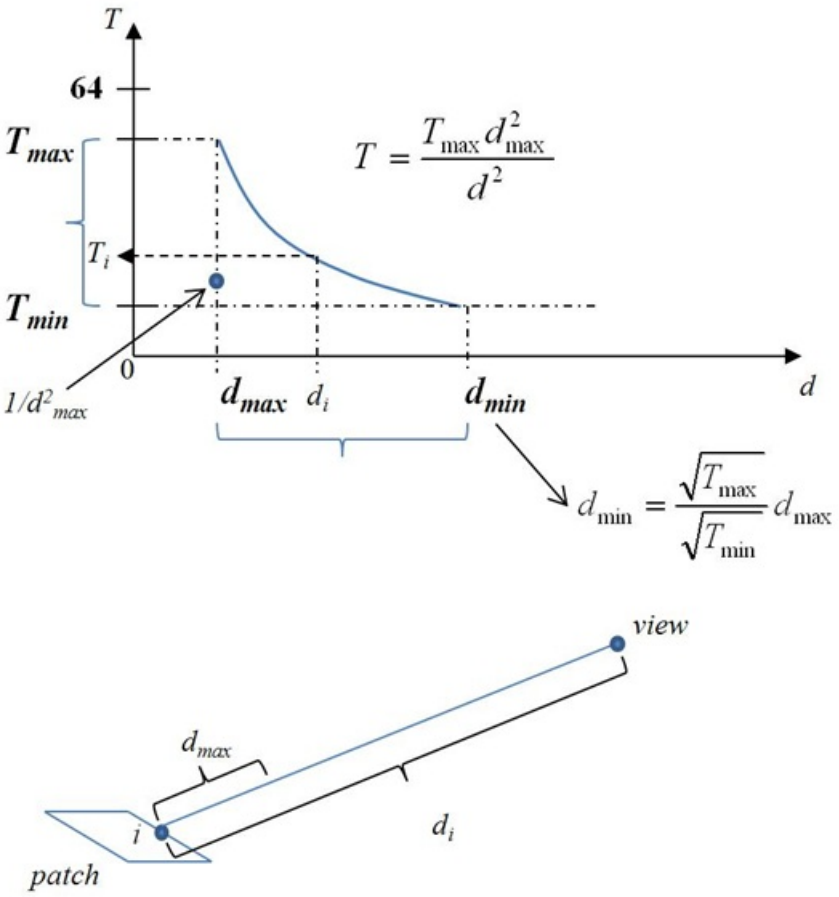

Figura 5.5: TessFactor como uma função de distância da câmera d (equação 5-1)

$$D_x = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} * H$$

$$D_y = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} * H$$

onde * representa a operação de convolução em duas dimensões. Em cada ponto na imagem, a magnitude do gradiente é obtida através da combinação das duas aproximações:

$$D = \sqrt{{D_x}^2 + {D_y}^2} \tag{5-8}$$

Um novo mapa M representando a magnitude do gradiente é criado. Cada Texel de M possui o valor de D correspondente ao respectivo ponto em H. O mesmo processo é repetido com M para calcular aproximações da segunda

derivada que chamaremos de height acceleration map (HAM). O processo de geração do HAM é similar a outros algoritmos de processamento de imagem e pode ser facilmente paralelizado na GPU usando pixel shaders [Mitchell, 2002].

5.6.2 Transferindo o HAM para o GPU

A nova pipeline baseada no Shader Model 5.0, trabalha com o conceito de patches, que em nosso caso é um tile do terreno. Os fatores de tecelagem devem ser determinados por patch no estágio Hull Shader.

Cada patch exige quatro valores informativos (cada um em um canal de textura): valor de HAM_{max}, máxima altura h, a posição do texel x de h e a posição do texel y de h. Em seguida, uma textura H com quatro canais de dados é criada. O primeiro canal recebe o HAM_{max}. O segundo canal recebe o máximo valor de altura h. Já o terceiro e quarto canais recebem a posição (x,y) do texel correspondente.

5.6.3 Deslocamento de terreno

Após a malha ser tesselada, o próximo estágio de processamento é o Domain Shader, que é executado para cada novo vértice gerado pelo Tessellator. O Domain Shader recebe os quatro vértices do patch inicial v1, v2, v3, v4 e duas coordenadas normalizadas ([0..1]), u e v, que representam os vértices gerados para este patch. Para encontrar a posição universal do novo vértice, nós linearmente interpolamos os vértices do patch, utilizando u e v como fatores de interpolação. O mesmo processo é feito com as coordenadas de textura. Após isso, o movimento vertical do vértice é obtido a partir do mapa de altura utilizando as coordenadas de textura do vértice.

5.7 Resultados

Nossos testes foram executados em um Intel Core i7 920 com uma Nvidia GTX480 e 6GB de RAM. Iniciamos com um mapa de altura de 2048x2048, que é equivalente ao mapa do Rio de Janeiro ($4,4\text{x}10^{10}m^2$) com um detalhe máximo de $100m^2$. Também foi feito um frustum culling em GPU para descartar os patches que não estão no frustum da câmera (basta setar o TessFactor para 0).

As figuras 5.6 e 5.8 mostram os resultados para o caso de 2048x2048 (Figuras 5.7c e 5.7d). Os resultados mostraram que apenas por ser capaz de explorar o Tessellator sem aplicar nenhum algoritmo LOD, em termos de fps

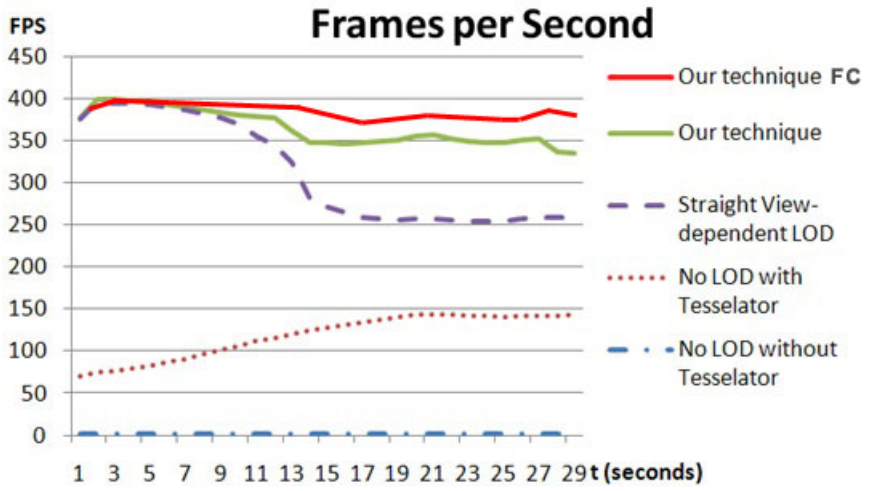

Figura 5.6: Frames por segundo (fps) para o caso das Figuras 5.7c e 5.7d. "Straight View-dependent LOD" não usa T_{max}, "No Lod with Tessellator" utiliza o TessFactor 64 para toda a malha, "No LOD, without Tessellator" é apenas uma referência (o caso onde toda a malha é transferida do CPU para o GPU). "Our technique FC" mostra a performance com o frustum culling ligado.

nossa renderização já é muito superior a uma abordagem simples de CPU para terrenos, como pode ser visto nas curvas mais baixas da Figura 5.6. Nesse caso, ambas as figuras apresentaram o mesmo número de triângulos. Comparando um tradicional algoritmo de LOD view-dependent com nossa técnica, somos capazes de manter valores consistentes de frame independente da posição do visualizador. Além disso, o número de triângulos foi reduzido significantemente com nossa técnica Figura 5.8.

A Figura 5.9 apresenta valores de fps para o caso de um height map 65536x65536, que é equivalente ao mapa do Brasil ($8x10^{12}m^2$ com um detalhe máximo de $40m^2$). O resultado mostra que nosso sistema é capaz de promover um rendering consistente para um caso de terreno massivo. Como os resultados indicam, nossa técnica é aplicável para a maioria dos mapas de altura disponíveis atualmente.

5.8 Discussão

Apresentamos uma nova técnica para renderização de terrenos utilizando tecelagem em hardware, sem estruturas hierárquicas. Nossa técnica é capaz de renderizar grandes mapas de alturas com frame rates interativos, além de ser muito simples de implementar. Apresentamos também um verdadeiro processo local paralelo, no sentido de que os resultados de cada patch de terreno não dependem dos resultados já obtidos em patches adjacentes ou outros patches.

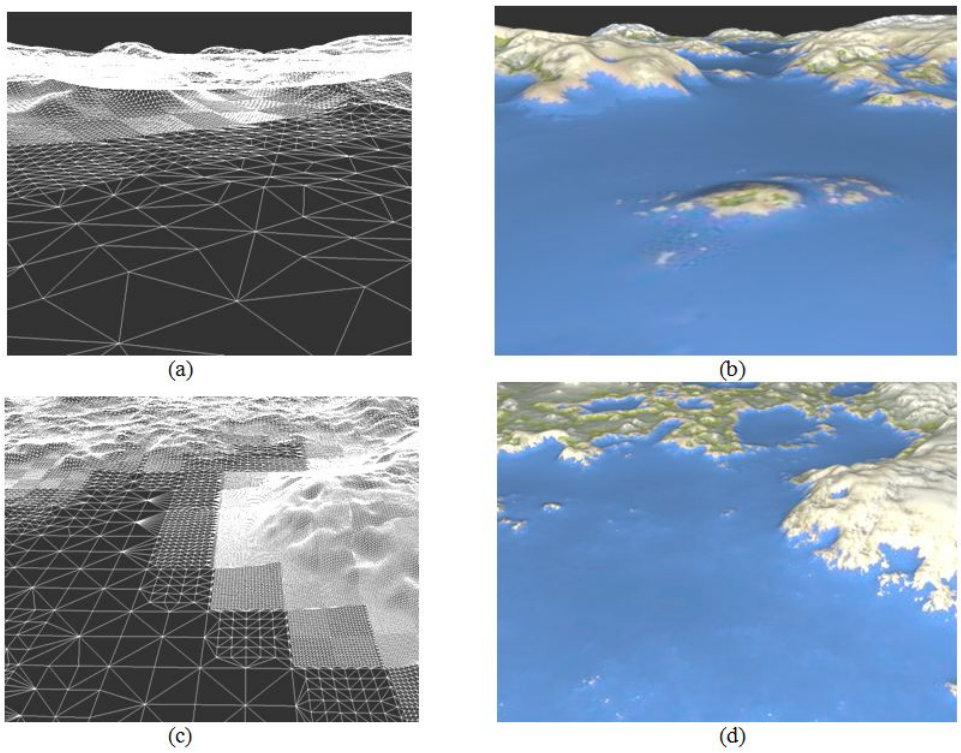

Figura 5.7: Modelos de terreno em wireframe e visualização final correspondente gerados pela técnica proposta rodando em uma Nvidia GTX480 e um Intel core i7. (a) e (b): mapa de altura de 65536x65536, área de $8\text{x}10^{12}m^2$ com precisão de $40m\text{x}40m$, à 52-109 fps. (c) e (d): mapa de altura de 2048x2048, area de $4.4\text{x}10^{10}m^2$ com precisão de $100m\text{x}100m$, à 347-399 fps.

Com esta abordagem somos capazes de determinar, quando necessário, o mínino de tecelagem exigido para um mínimo de erro em espaço projetado. Mantendo o menor refinamento possível, podemos reduzir overheads de subdivisão. Consequentemente, em áreas onde um refinamento profundo é exigido, podemos usar o tessellator em todo seu potencial. Fazendo o melhor uso deste hardware, nós drasticamente minimizamos o overhead do barramento CPU-GPU com apenas um pequeno grupo de primitivas sendo transferido. Nosso algoritmo é linearmente escalável, enquanto alguns outros modelos apresentam algoritmos hierárquicos logarítmicos. Contudo, nossa constante é muito pequena devido ao uso extensivo do tessellator. Em nosso testes, usando o tessellator podemos renderizar modelos em torno de 10^8 triângulos com frame rates interativos e erros menores que 3 pixels. Em qualquer outra abordagem de CPU com LOD este caso corresponderia a um exorbitante número de 5.0 GB de dados sendo transferidos em cada frame entre CPU e GPU. O contínuo fotorrealismo do terreno sem erros é garantido pelo estável geomorphing e edge-splitting através do estágio fixo do tessellator na GPU.

Para futuros trabalhos, planejamos criar uma hierarquia de tiles em terreno que seja capaz de manter a continuidade das divisões. Com tal

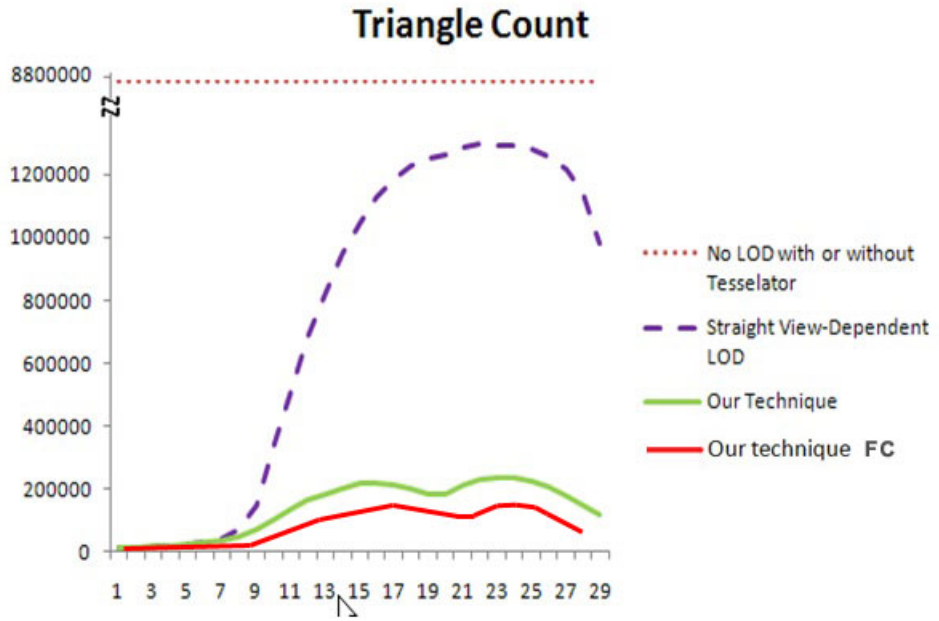

Figura 5.8: Contagem de triângulos para o mesmo caso da Figura 5.6.

estrutura, o sistema estaria apto a selecionar por toda parte do terreno qual nível de hierarquia utilizar. Portanto, o número de primitivas poderia ser reduzido drasticamente para terrenos massivos.

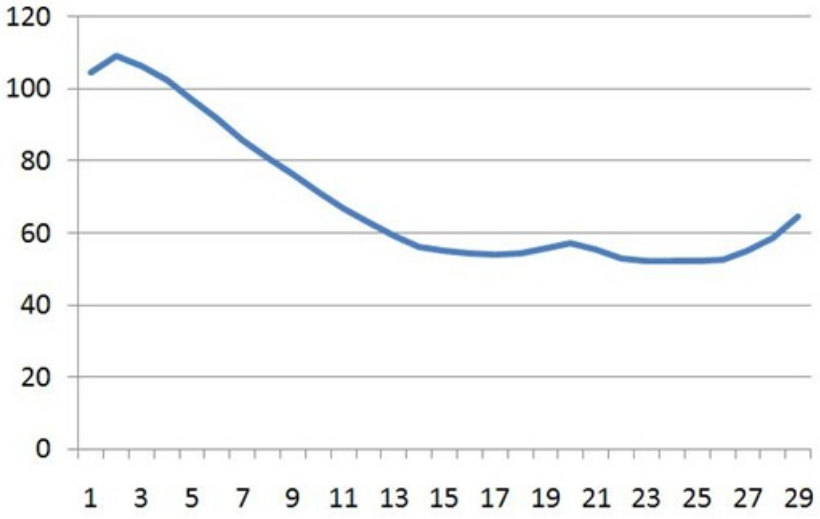

Figura 5.9: Frames por segundo (fps) para o caso 65536x65536 nas Figuras 5.7a e 5.7b.