UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Alex Thomas Almeida Frasson

RENDERIZAÇÃO DE TERRENOS EM ESCALA PLANETÁRIA

Santa Maria, RS
2018

PPGCC/UFSM, RS
Frasson, Alex
Mestre
2018

Alex Thomas Almeida Frasson

RENDERIZAÇÃO DE TERRENOS EM ESCALA PLANETÁRIA

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-graduação em Ciência da Computação, Área de Concentração em Computação Aplicada, da Universidade Federal de Santa Maria(UFSM, RS), como requisito parcial para a obtenção do grau de **Mestre em Ciência da Computação**.

Orientador: Dr. Cesar Tadeu Pozzer

Santa Maria, RS
2018

Frasson, Alex Thomas Almeida
Renderização de terrenos em escala planetária / Alex Thomas Almeida Frasson.- 2018.
52 f.; 30 cm

Orientador: Cesar Tadeu Pozzer
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação, RS, 2018

1. Renderização em tempo real 2. Terrenos esféricos 3. LOD 4. Geração procedural I. Pozzer, Cesar Tadeu II. Título.

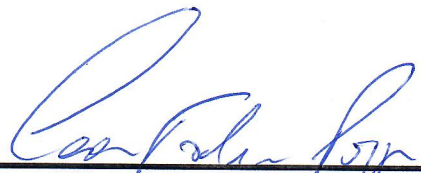
Sistema de geração automática de ficha catalográfica da UFSM. Dados fornecidos pelo autor(a). Sob supervisão da Direção da Divisão de Processos Técnicos da Biblioteca Central. Bibliotecária responsável Paula Schoenfeldt Patta CRB 10/1728.

Alex Thomas Almeida Frasson

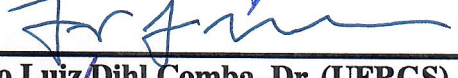
RENDERIZAÇÃO DE TERRENOS EM ESCALA PLANETÁRIA

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-graduação em Ciência da Computação, Área de Concentração em Computação Aplicada, da Universidade Federal de Santa Maria(UFSM, RS), como requisito parcial para a obtenção do grau de **Mestre em Ciência da Computação**.

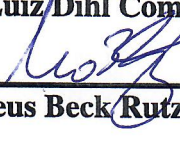
Aprovado em 26 de Fevereiro de 2018:



Cesar Tadeu Pozzer, Dr. (UFSM)
(Presidente/Orientador)



João Luiz Dihl Comba, Dr. (UFRGS)



Mateus Beck Rutzig, Dr. (UFSM)

Santa Maria, RS
2018

AGRADECIMENTOS

Agradeço ao projeto SIS-ASTROS, dentro do qual este trabalho foi desenvolvido, pelo suporte e auxilio financeiro.

"The cord that tethers ability to success is both loose and elastic. It is easy to see fine qualities in successful books or to see unpublished manuscripts, inexpensive vodkas, or people struggling in any field as somehow lacking. It is easy to believe that ideas that worked were good ideas, that ideas and plans that did not were ill conceived. And it is easy to make heroes out of the most successful and to glance with disdain at the least. But ability does not guarantee achievement, nor is achievement proportional to ability. And so it is important to always keep in mind the other term in the equation – the role of chance."

Leonard Mlodinow

RESUMO

RENDERIZAÇÃO DE TERRENOS EM ESCALA PLANETÁRIA

AUTOR: Alex Thomas Almeida Frasson
ORIENTADOR: Dr. Cesar Tadeu Pozzer

Embora um tópico muito pesquisado, a renderização de terrenos ainda apresenta desafios. Por exemplo, muitas aplicações dependem de uma visualização precisa de um terreno altamente detalhado e extenso, exigindo o uso eficiente da GPU e CPU para fornecer renderização em tempo real. Além disso, o aumento da disponibilidade de dados de elevação de cobertura global de alta resolução exige sistemas de renderização capazes de considerar a forma do planeta. Mesmo que as técnicas anteriores sejam capazes de renderizar tais conjuntos de dados, a maior parte visa apenas a visualização e não considera interações entre o terreno e objetos. Portanto, neste trabalho, apresentamos uma técnica de renderização para terrenos esféricos que é capaz de visualizar terrenos extremamente detalhados ao mesmo tempo em que suporta interações com objetos simulados. Adaptamos uma técnica de nível de detalhamento contínua ao domínio esférico e empregamos técnicas de geração procedural para produzir dados de elevação de altíssima resolução em tempo de execução.

Keywords: renderização, terrenos esféricos, LOD, geração procedural.

ABSTRACT

PLANETARY-SCALE TERRAIN RENDERING

AUTHOR: Alex Thomas Almeida Frasson
ADVISOR: Dr. Cesar Tadeu Pozzer

Although a thoroughly researched topic, terrain rendering still presents challenges. For instance, many applications rely on accurate visualization of highly detailed and vast terrain, requiring the efficient use of both GPU and CPU in order to provide real-time rendering. Additionally, the increased availability of high resolution global coverage elevation datasets demands rendering systems capable of accounting for the shape of the planet. Even though previous techniques are capable of rendering such datasets, most target visualization only and do not account for interactions between the terrain and simulated objects. Therefore, in this work we present a rendering technique for spherical terrains that is capable of visualizing extremely detailed terrains while also supporting interactions with simulated objects. We adapt a continuous level-of-detail technique to the spherical domain and employ procedural generation techniques to produce sub-meter elevation data at run-time.

Keywords: rendering, spherical-terrain, LOD, procedural generation.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A qualidade visual e a fidelidade esperadas por aplicações voltadas a visualização de dados aumentaram com os avanços nos recursos e no desempenho do hardware. O mesmo se aplica ao nicho específico de renderização de terrenos, que tem sido de grande interesse de pesquisa nas últimas décadas. O aumento da disponibilidade e resolução da aquisição de dados de elevação reforçou a importância desta área de pesquisa e motivou o desenvolvimento de novas técnicas de renderização ao longo dos anos. Tais técnicas foram empregadas em muitas aplicações. Por exemplo, Sistemas de Informações Geográficas (GIS) podem ser usados por cientistas para visualizar conjuntos de dados de elevação e realizar análises topográficas; softwares de simulação, como aplicações para treinamento militar e de voo exigem uma renderização ampla e detalhada dos dados reais do terreno; e a indústria do entretenimento aproveita a renderização do terreno para fornecer muitas horas de exploração em grandes paisagens em jogos de mundo aberto.

Embora a maioria das aplicações dependa de uma visualização de terreno de alta qualidade, algumas exigem maiores níveis de precisão na visualização. Tais aplicações incluem GIS e softwares de simulação e têm sido o principal incentivo de pesquisas que focam na visualização de alta precisão de conjuntos de dados de elevação. Em contraste com a maioria dos jogos em que a renderização do terreno é principalmente planar, esses aplicativos visam conjuntos de dados grandes o suficiente para serem afetados pela curvatura do planeta, destacando a necessidade de técnicas de renderização não planares que visualizem corretamente tais conjuntos de dados. Estas técnicas incluem a projecção de dados de elevação na superfície aproximada de um planeta, como por exemplo, um elipsóide ou uma esfera.

Enquanto as aplicações GIS, em geral, têm uma forte demanda por manipulação eficiente de dados, jogos e softwares de simulação também exigem interação entre objetos simulados e a geometria subjacente do terreno. Embora a manipulação e a renderização de alta fidelidade de grandes terrenos tenham sido abordadas com sucesso em trabalhos anteriores, a integração entre bibliotecas físicas e técnicas de renderização de terrenos em escalas planetárias não está devidamente documentada. Portanto, um trabalho abrangente abrangendo a renderização em tempo real de alta qualidade e o suporte para simulação de física em escalas planetárias seria de grande interesse para a comunidade de pesquisadores.

Por exemplo, o treinamento militar de um sistema de lançamento de mísseis guiados inclui a cuidadosa manobra e posicionamento de veículos militares antes do lançamento, o que requer uma representação altamente detalhada do terreno. Após o lançamento, a trajetória do míssil deve ser simulada até atingir o alvo final, o qual pode estar a uma distância de milhares de quilômetros da posição de lançamento. Tal simulação requer a renderização de grandes extensões de terreno que são afetadas pela curvatura da Terra, tornando o uso de terrenos planares inadequados para este propósito. Portanto, este sistema requer o manuseio de dados de elevação de alta resolução com grandes extensões, exigindo um sistema eficiente de nível de de-

talhamento para fornecer renderização em tempo real e de alta qualidade. Por fim, esse sistema poderia ser integrado ao projeto SIS-ASTROS (Sistema de Simulação Integrada para a Bateria ASTROS) – no qual esta pesquisa foi desenvolvida – um sistema de simulação do Exército Brasileiro destinado a treinar militares em doutrinas táticas sobre o uso de Bateria ASTROS (Artillery Saturation Rocket System).

Motivado pelos requisitos de tais aplicações, neste trabalho apresentamos um sistema de renderização em tempo real com nível de detalhamento contínuo para terrenos em escala planetária que podem renderizar dados de elevação de resolução submétrica. Aplicamos um esquema de tesselação que fornece uma distribuição altamente uniforme de amostras. Além disso, também abordamos a integração com uma biblioteca de física para fornecer interação com objetos simulados que podem estar espalhados pelo planeta. Nosso sistema fornece precisão suficiente para softwares de simulação e a maioria dos aplicativos de visualização de dados e mantendo o desempenho em tempo real. No entanto, não abordamos o carregamento de dados de elevação sob demanda, portanto, técnicas de geração procedural são empregadas para validar o sistema de renderização proposto. Finalmente, fornecemos a descrição de um processo para prover renderização de grandes conjuntos de dados que pode ser integrado com o nosso sistema.

Alcançamos taxas de quadros em tempo real aplicando uma técnica contínua de renderização de terreno em nível de detalhe contínuo às faces rômbicas de um poliedro de 30 lados, o qual fornece uma distribuição uniforme de amostras. Este poliedro é então projetado em uma esfera e cada vértice é deslocado de acordo com os dados de elevação. Aproveitamos os compute shaders para gerar progressivamente dados de elevação procedurais a uma resolução abaixo de um metro que pode ser usada para gerar um planeta inteiro ou simplesmente para aumentar o nível de detalhamento dos dados reais de elevação, se disponibilizados através de carregamento sob demanda. A mesma estrutura de dados também fornece uma malha de colisão que é gerada dinamicamente com base no posicionamento de objetos, possibilitando a simulação de objetos espalhados pelo planeta.

1.1 Contribuições

Este trabalho apresenta as seguintes contribuições:

- A adaptação de uma técnica contínua de nível de detalhamento ao domínio esférico;
- Uso de tesselação de esferas altamente uniforme para fornecer renderização de baixa distorção de dados de elevação;
- Um processo para integrar uma biblioteca de física com a técnica de renderização proposta;
- Detalhes de implementação destacando as ressalvas encontradas ao longo do desenvolvimento;

- Discussão aprofundada de problemas de desempenho e possíveis direções para melhorias.

1.2 Esboço da dissertação

O Capítulo 2 apresenta uma revisão das técnicas de renderização de terrenos planos e esféricos e outros tópicos relevantes, como a tesselação de esferas e os elipsóides de referência. No Capítulo 3, fornecemos uma descrição detalhada do nosso sistema. O Capítulo 4 fornece detalhes da implementação, juntamente com uma discussão sobre os resultados qualitativos e de desempenho. Finalmente, no Capítulo 5, discutimos a solução proposta e fornecemos descrições para futuras direções de pesquisa e melhorias de desempenho.

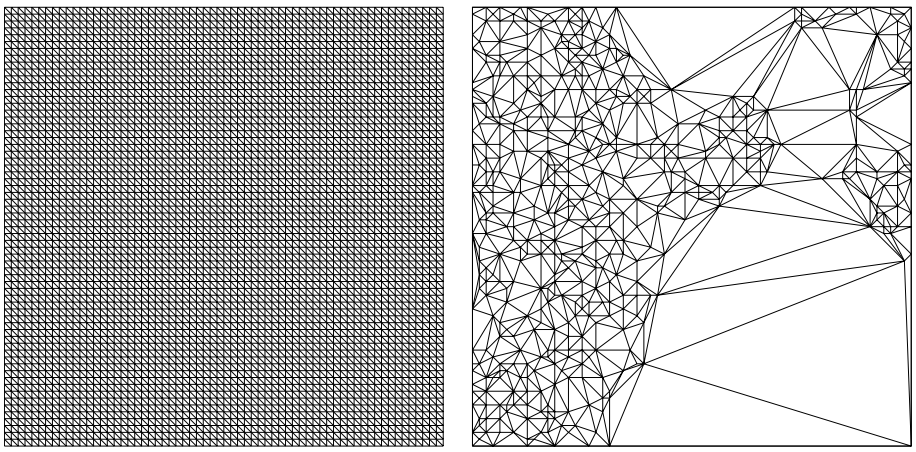
2 FUNDAÇÕES

Neste capítulo apresentamos uma breve história e uma revisão aprofundada de técnicas notáveis de renderização de terrenos e conceitos essenciais sobre a renderização de terrenos esféricos.

2.1 RENDERIZAÇÃO DE TERRENOS PLANARES

Os dados de elevação são geralmente armazenados como uma imagem em que cada pixel representa uma altura sobre um plano base e pode ser referida como modelo de elevação digital (DEM), campo de altura, heightmap, etc. A técnica mais simples para visualizar esta representação é o **Grid Regular**, onde uma malha é gerada para que cada vértice seja mapeado para um pixel, ou amostra, em um mapa de altura. Em contraste com a sua simplicidade, é também a abordagem menos eficiente para a visualização do terrenos. Por exemplo, um grid regular de 1000x1000 exigiria o processamento de 2M triângulos, um número que claramente não se encaixaria no orçamento apertado de 10.000 triângulos do hardware disponível em meados dos anos 90. Por outro lado, uma representação em **redes triangulares irregulares (TIN)** de um mapa de altura pode ser usada para se encaixar nesse orçamento de renderização. O TIN é uma representação vetorial de dados de elevação e sua principal vantagem sobre um grid regular, como mostrado na Figura 1, é a amostragem irregular de dados que preserva vértices em áreas de detalhes de alta frequência. Em seu trabalho, Garland and Heckbert (1995) investigam o uso de vários algoritmos para gerar TINs de mapas de altura. Embora o TIN minimize efetivamente o custo de geometria da renderização, ele requer longos períodos de pré-processamento, tornando-o inadequado para terrenos dinâmicos por si só.

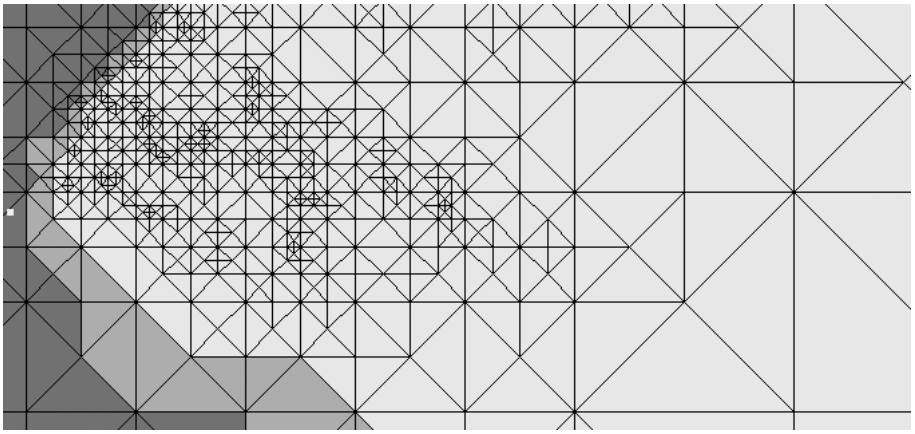
Figura 1 – Representações em grid regular e TIN de um heightmap. (GARLAND AND HECKBERT, 1995)



Ambos grids regulares e a TINs por conta própria não são adequadas para renderizar terrenos grandes. Para fornecer visualização em tempo real de grandes conjuntos de dados, téc-

nicas de nível de detalhamento que dependem do observador devem ser usadas. Modelos multirresolução baseados em triangulações irregulares que foram capazes de refinar e simplificar dinamicamente uma malha com base na posição do observador foram propostos. No entanto, as malhas irregulares são complexas de manipular, uma vez que as operações de simplificação devem levar em conta a vizinhança aleatórias das amostras de elevação. Impondo restrições nas operações de conectividade e atualização de malha, abordagens de malha semi-regulares como **Real-time Optimally Adapting Mesh (ROAM)** foram capazes de reduzir o custo de operações de armazenamento, renderização e atualização. Pajarola and Gobbetti (2007) fornecem uma extensa pesquisa sobre este tópico. ROAM é uma abordagem de malha semi-regular e é um algoritmo contínuo de nível de detalhamento que usa uma bintree triangular pré-processada para simplificar e refinar uma malha. Esse processo progressivo aproveita a coerência entre quadros e usa duas filas de prioridade: uma que armazena nós candidatos para uma operação de mesclagem e outra que armazena nós candidatos para uma operação de divisão.

Figura 2 – Malha semi-regular produzida pelo ROAM. O obervador está a esquerda olhando para a direita. (DUCHAINEAU, WOLINSKY, ET AL., 1997)



Embora as técnicas baseadas em malhas semi-regulares minimizem o número de triângulos a serem rasterizados, no final dos anos 90, o processo de geração de uma malha ótima tornou-se o gargalo com o deslocamento do processamento de vértices da CPU para a GPU. Abordagens baseadas em lote foram desenvolvidas para aproveitar os novos recursos de hardware. Essas abordagens concentraram-se em reduzir o tempo gasto na simplificação da malha e nas transferências entre a CPU e a GPU. Isto é obtido operando em conjuntos de triângulos em vez de primitivas individuais, o que reduz a qualidade da malha final (aumento no número de triângulos), mas também reduz o tempo gasto na CPU com geração de malha.

Geomipmap (DEBOER, 2000) aplica a idéia de envio de primitivas em lote através de uma estrutura quadtree onde cada nó armazena um grid regular de vértices com uma resolução fixa. Para gerar essa hierarquia, os dados de elevação são subamostrados de forma semelhante a técnica de mipmap de texturas, em que cada nível sucessivo tem metade da resolução do anterior. Durante o tempo de execução, os nós de são selecionados de acordo com sua distância para

o visualizador e removidos com base no frustum de visualização. **Chunked LOD** (ULRICH, 2002) melhora a abordagem de De Boer, otimizando a geometria de cada nó da quadtree com o uso de uma bintree. A abordagem de Ulrich também adiciona suporte para compactação e resolve o problema de rachaduras aplicando saias nas bordas dos nós. Rachaduras e junções em T são portanto escondidas e não eliminadas. Além disso, sua abordagem requer um pré-processamento extensivo e mais espaço de armazenamento, visto que todos os componentes (x, y, z) de cada amostra devem ser armazenados. Da mesma forma, em **Batched Dynamic Adaptive Meshes (BDAM)** (CIGNONI, GANOVELLI, ET AL., 2003A) uma malha irregular é gerada para cada patch usando um algoritmo de inserção de Delaunay guloso no estágio de pré-processamento.

Figura 3 – Solução do geomipmap para junções em T e rachaduras. (DEBOER, 2000)

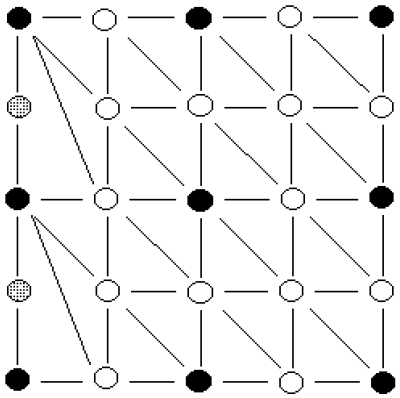
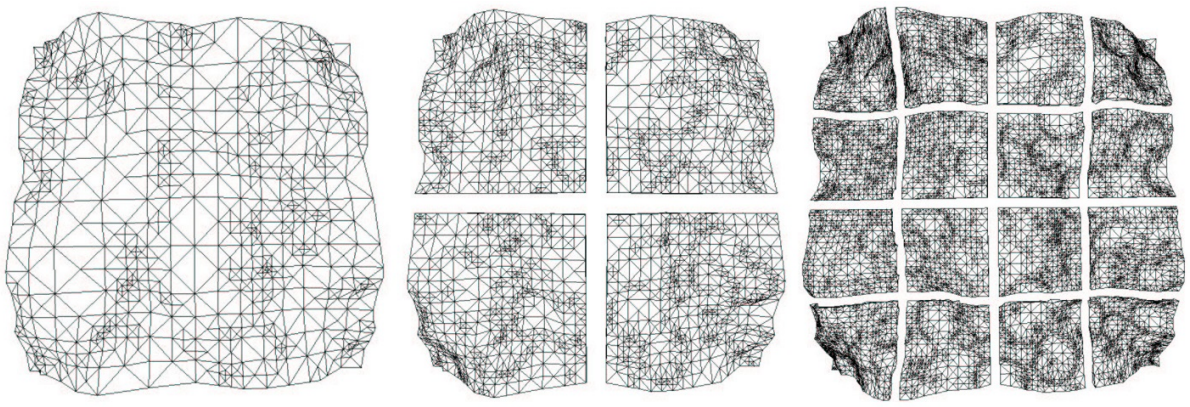


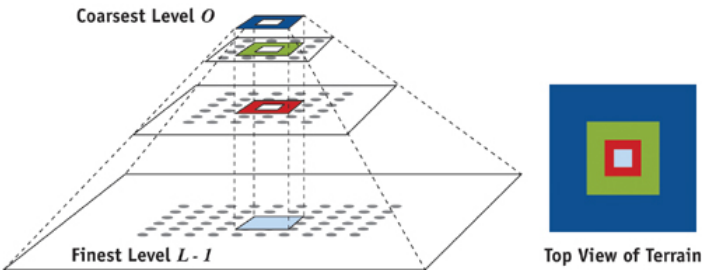
Figura 4 – Malha triangular hierárquica do Chunked LOD. (ULRICH, 2002)



No início dos anos 2000, GPUs com estágios programáveis começaram a surgir. Os estágios do pipeline programáveis permitiram que o programador escrevesse código personalizado para os novos estágios de shader fragmento e vértice. Essa flexibilidade gerou novas abordagens de renderização de terreno, como **Geometry Clipmap** (LOSASSO AND HOPPE,

2004). Geometry clipmap tira proveito do pipeline programável transformando vértices no shader de vértices onde a elevação é amostrada de uma textura. A geometria é composta por uma série de grids regulares aninhados armazenados em na memória da GPU. Cada um dos grids é centralizado em torno do observador e é incrementalmente atualizado com novos dados à medida que o observador se move. Diferentemente do Chunked LOD e BDAM, essa técnica requer apenas a geração do mipmap de um heightmap em seu estágio de pré-processamento. A contagem de triângulos é maior, uma vez que a malha é composta de grids regulares e não se adapta as variações do terreno. Isso adiciona mais vértices do que o necessário a regiões homogêneas e dificulta o controle de erro em espaço de tela. A contagem de triângulo é constante, no entanto. Ele requer menos espaço de armazenamento, uma vez que apenas as alturas devem ser armazenadas. Além disso, compactação pode ser aplicada aos dados de elevação, reduzindo muito os requisitos de armazenamento.

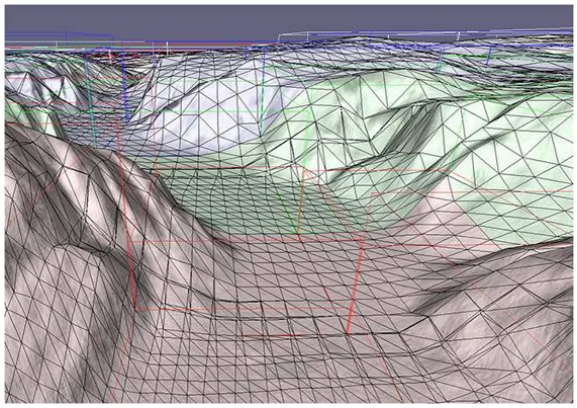
Figura 5 – Grids aninhados do Geometry clipmap. (LOSASSO AND HOPPE, 2004)



Continuous Distance-dependent LOD (CDLOD) (STRUGAR, 2009) segue a mesma idéia de amostrar um mapa de altura e transformar vértices de um grid regular no shader de vértices. A malha é gerada pela seleção de nós de uma quadtree com base na distância do observador. A transição entre os níveis de LOD também é realizada no shader de vértices pela transformação dos vértices adjacentes de cada nó em uma malha de resolução mais baixa. A transição é baseada na distância de vértices individuais para o observador, resultando em transições suaves entre os níveis de LOD. Além disso, as transições ainda podem ser visíveis em terrenos extremamente acidentados. O pré-processamento é reduzido ao mipmap de um heightmap para fornecer renderização sob demanda de grandes conjuntos de dados.

Com a introdução da **tesselação em hardware** no final de 2009, foram propostas abordagens para tirar proveito desta nova capacidade do hardware. Cantlay (2011) apresenta uma técnica LOD de terreno que se baseia em shaders de tesselação para refinar uma malha grosseira composta de patches quadrados. O nível de tesselação é determinado pelo tamanho das arestas dos patches em espaço de tela, o qual pode ser controlado com um único parâmetro. Isso garante uma distribuição quase uniforme de triângulos em espaço de tela. Ele também propõe uma solução para a geração de uma malha livre de rachaduras composta de tamanhos de patches não uniformes. No entanto, essa abordagem pode sofrer de aliasing causado pelo movimento contínuo de vértices dentro de cada patch. Esse problema pode ser minimizado

Figura 6 – Realce da seleção de nós da quadtree do CDLOD. A malha mostra a transição entre diferentes níveis. (STRUGAR, 2009)



com o aumento no nível de tesselação, diminuindo o tamanho dos triângulos em espaço de tela. Fernandes and Bruno (2012) estendem essa técnica adicionando um parâmetro de rugosidade pré-calculado para cada patch para evitar a super-tesselação de áreas homogêneas. Devido ao requisito de pré-computação, a adição do parâmetro de rugosidade torna esta técnica inadequada para terrenos dinâmicos.

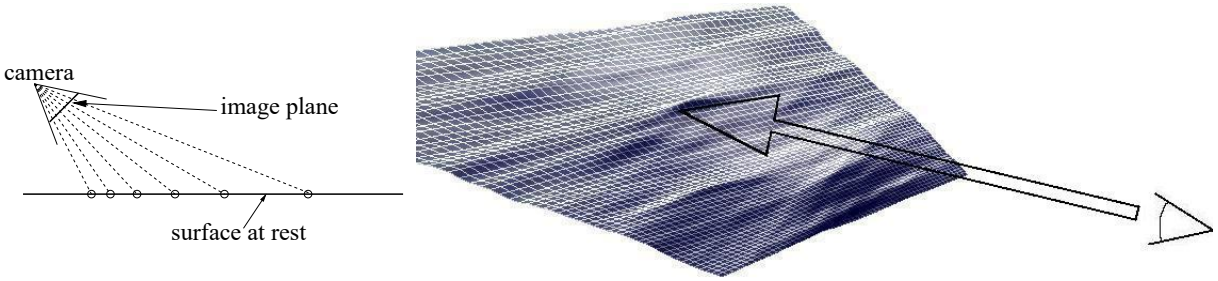
A jornada para extrair todo o poder de processamento disponível da GPU continuou com Mistal (2013) e Dupuy, Iehl, et al. (2014). Ambas as abordagens de renderização de terreno propostas são executadas quase exclusivamente na GPU, liberando a CPU para outras tarefas cruciais. Dupuy, Iehl, et al. (2014) utilizam um nível de detalhamento baseado em uma quadtree similar a Strugar (2009). Eles armazenam a árvore linearmente na memória da GPU com um único inteiro por nó. Um shader de geometria é usado para atualizar a árvore dividindo ou mesclando nós e gravando os resultados em outro buffer em uma maneira pingue-pongue. A renderização é obtida instanciando um grid regular para cada nó e recuperando a escala e a posição do nó a partir de sua representação inteira. Rachaduras são removidas com tesselação da malha com um shader de tesselação que usa a distância para o espectador para avaliar uma função de LOD e determinar a quantidade de tesselação necessária. Em contraste com o CDLOD, sua abordagem não requer uma quadtree restrita para remover completamente as rachaduras. Os resultados mostram que os tempos de CPU são mantidos no mínimo. No entanto, eles não fornecem uma solução carregamento sob demanda de dados de elevação. A sincronização CPU-GPU pode ser necessária para suportar tal funcionalidade bem como a detecção de colisão, o que pode afetar o desempenho.

Até agora, revisamos técnicas de renderização de terreno baseadas em rasterização. No entanto, abordagens usando **ray-casting** para gerar uma imagem dependente do observador também foram propostas ao longo dos anos. A renderização de terrenos baseada em ray-casting geralmente lança um raio a partir de um shader de fragmento e executa testes de interseção com os dados de elevação para encontrar o ponto de intersecção mais próximo. Tevs, Ihrke,

et al. (2008) propuseram uma estrutura hierárquica chamada **maximum mipmaps**, que acelera o ray-casting, permitindo que o raio salte grandes partes de um heightmap. A estrutura pode ser eficientemente computada na GPU. Dick, Krüger, et al. (2009) mostram que o ray-casting é uma alternativa viável à rasterização ao renderizar conjuntos de dados extremamente grandes. O trabalho deles aplica maximum mipmaps para patches do terreno que são inicialmente selecionadas de acordo com o frustum do observador. Dick, Krüger, et al. (2010) melhoram a renderização do terreno propondo o uso de um renderizador híbrido que mistura rasterização e ray-casting. Ele se adapta à GPU em uso analisando o número de pixels e triângulos que seriam necessários para renderizar um bloco e escolher a técnica de renderização apropriada.

Projective Grid Mapping (PGM) é outro mix de ray-casting e rasterização e foi introduzido pela primeira vez em animação de ondas (HINSINGER, NEYRET, ET AL., 2002). Em PGM, uma malha de grid regular em espaço de tela é gerada por meio de ray-casting. A malha é gerada pelo lançamento de um raio composto pela posição do observador e a posição do vértice em espaço do mundo em direção a um plano base. O ponto de intersecção é usado para determinar o valor da altura, deslocando ao longo da normal do plano e depois definindo esta posição como a posição final do vértice. A figura 7 mostra como essa abordagem gera triângulos com uma área em espaço de tela quase uniforme. Taxas de quadros quase constantes são esperadas desde que o tamanho do grid seja constante. Além disso, o tamanho do grid pode ser ajustado para corresponder aos recursos de hardware disponíveis. Se o tamanho do grid projetado corresponder ao tamanho da viewport, os triângulos terão aproximadamente o tamanho de um pixel. Deve-se tomar cuidado, pois os picos de elevação podem cruzar o campo de visão de baixo, exigindo a amostragem de tais áreas. Além disso, baixos ângulos horizontais produzirão amostragem esparsa, resultando na perda de detalhes. O PGM foi adaptado para a renderização do terreno. Livny, Sokolovsky, et al. (2008) combinam PGM e **maximum mipmap** para aliviar os efeitos da undersampling. Löffler, Schumann, et al. (2009) adaptam o PGM para amostrar uniformemente o plano base do terreno, a fim de mitigar também os efeitos da falta de amostragem.

Figura 7 – Projective Grid Mapping. (HINSINGER, NEYRET, ET AL., 2002)



Revisamos até aqui as técnicas de renderização de terreno mais notáveis e vimos que a evolução do hardware inegavelmente levou à busca de novas abordagens. Como esperado, todas

as técnicas apresentadas pressupõem que os dados de elevação são distribuídos em um plano. No entanto, conforme destacado na seção 2.2.1, muitas dessas abordagens foram adaptadas ao domínio esférico.

2.2 Terrenos Esféricos

A maioria dos planetas não é perfeitamente esférico e pode apresentar formas que nem sequer se parecem com uma esfera. No entanto, a visualização precisa de planetas não é trivial e, portanto, aproximações são geralmente aplicadas na maioria das aplicações. A Terra, por exemplo, pode ser aproximada por uma esfera ou mais precisamente por um elipsóide. O modelo elipsóide de referência mais utilizado para o planeta Terra é o elipsóide WGS84. Ele define a Terra como um esferoide oblato com semi-eixo maior de 6.378.137 m e um eixo semi-menor de $6.356.752,3142$ m. Algumas das pesquisas apresentadas na seção 2.2.1 fornecem suporte para elipsóides. Conforme detalhado no capítulo 3, descrevemos um sistema para renderização de planetas esféricos e fornecemos uma discussão adicional sobre esferóides no Capítulo 5.

Ao renderizar com ray-casting, uma definição analítica da esfera pode ser usada. No entanto, a rasterização requer a definição de triângulos para aproximar a superfície da esfera. Isto é conseguido com a tesselação de esferas. Existem muitas maneiras alternativas de se tesselar a esfera. A maneira mais simples é usar o grid geográfico para gerar os vértices da esfera. Esta abordagem, no entanto, sofre com a sobre-amostragem nos pólos, como mostrado na Figura 8. Uma abordagem mais interessante é projetar um poliedro convexo na superfície da esfera. Isso pode ser facilmente alcançado multiplicando-se os vértices normalizados pelo raio da esfera. A qualidade final da tesselação é dependente do poliedro base. Por exemplo, como pode ser visto na Figura 9, o uso de um cubo produz distorções em triângulos próximos aos cantos. Em seu trabalho, Bernardin, Cowgill, et al. (2011) tesselam a esfera usando um triacontahedron rômbico de 30 lados (Figura 12), conseguindo distribuição de triângulos quase uniforme em toda a superfície da esfera.

Figura 8 – Sobre-amostragem gerada pelo grid geográfico nos polos da esfera. (Cozzi and Ring, 2011)

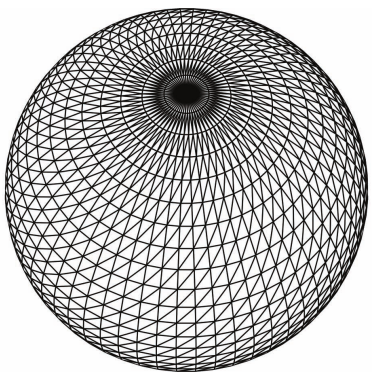
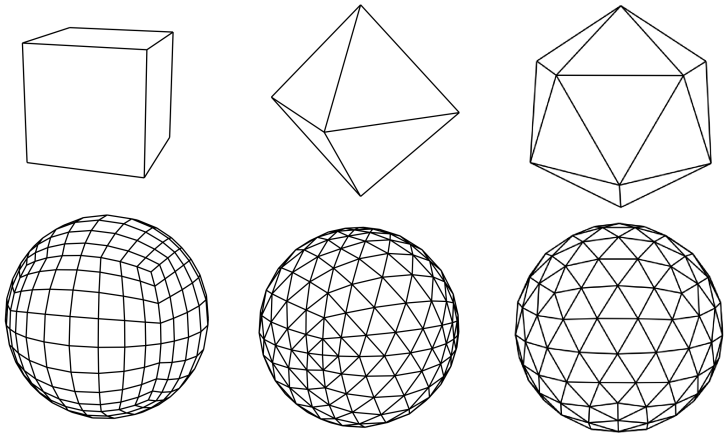


Figura 9 – Tesselação de esferas com diferentes poliedros base. Da esquerda para direita: cubo, octaedro e icosaedro (20 faces triangulares). (KOOIMA, 2008)

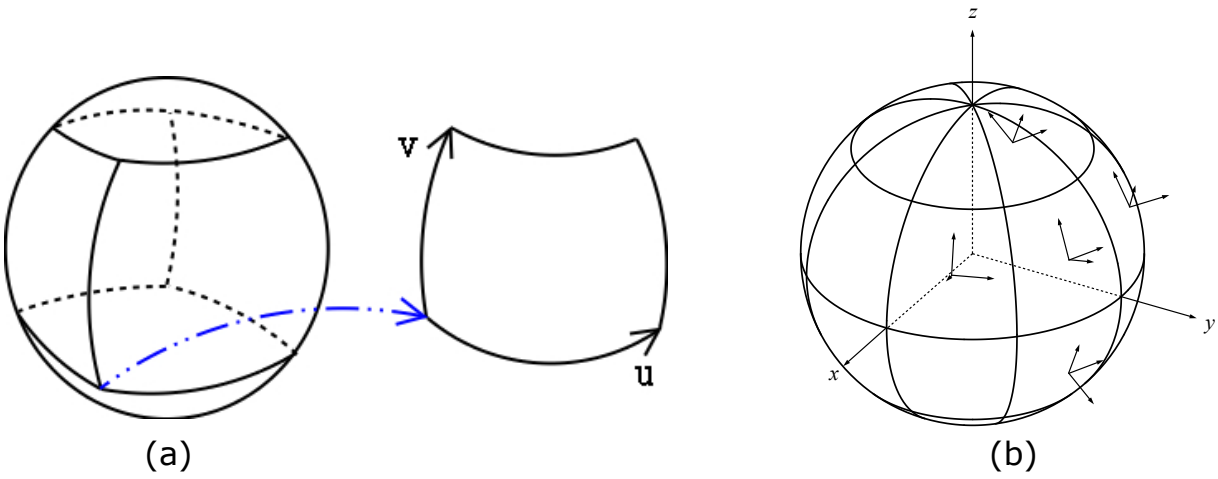


2.2.1 Renderização de Terrenos Esféricos

Hitchner (1992) foi o primeiro trabalho publicado sobre renderização de terrenos planetários onde ele subdividiu os dados de elevação com uma quadtree e usou a estrutura de dados para executar o culling baseado no frustum e selecionar nós com base na distância para o observador. No entanto, não há menção sobre como as junções em T são evitadas ou elas são. Lindstrom, Koller, Ribarsky, Hodges, Bosh, et al. (1997) apresentam um sistema (VGIS) que aplica a técnica de renderização de terreno proposta por Lindstrom, Koller, Ribarsky, Hodges, Faust, et al. (1996) para visualizar conjuntos de dados de elevação global. Reddy, Leclerc, et al. (1999) descrevem outro sistema que visa visualizar o planeta Terra. No entanto, eles não fornecem detalhes sobre seu sistema LOD. Hill (2002) tentou estender o ROAM para manipular planetas esféricos, mas ele abandonou essa abordagem e empregou uma técnica baseada em patches nas faces de um cubo. Cignoni, Ganovelli, et al. (2003b) estenderam o algoritmo BDAM para suportar planetas esféricos aplicando uma hierarquia BDAM para cada uma das seis faces de um cubo.

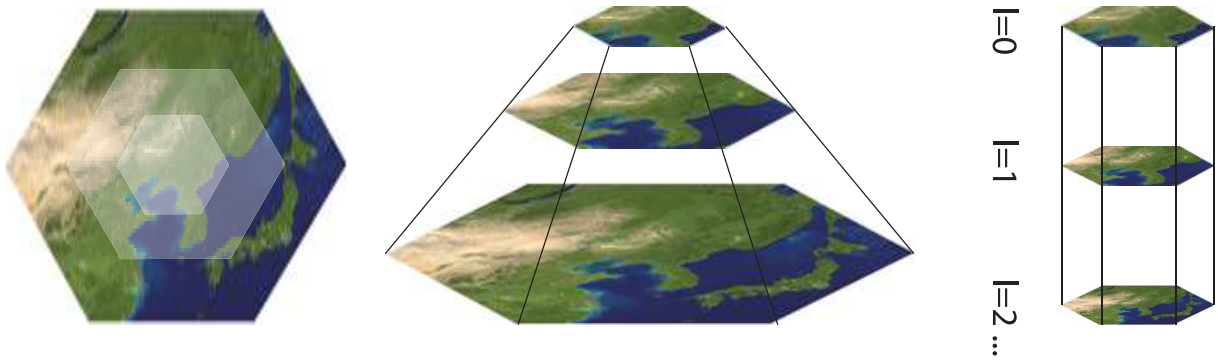
Geometry clipmap foi estendido por Clasen and Hege (2006), Bhattacharjee (2010) e Dimitrijević and Rančić (2015). Em Clipmap Esférico (CLASEN AND HEGE, 2006), anéis aninhados centralizados ao redor do observador são usados no lugar dos grids. Esta abordagem sofre com aliasing, pois os vértices não estão necessariamente alinhados com o heightmap. Bhattacharjee (2010) apresenta um geometry clipmap hexagonal que aproxima a esfera com um octaedro. Esta técnica, em contraste com o Clipmap Esférico, usa uma série de anéis hexagonais aninhados como visto na Figura 11. Dimitrijević and Rančić (2015) por outro lado, adaptam o geometry clipmap para renderizar planetas elipsoidais onde o elipsóide é particionado em três regiões para evitar singularidades e distorções geradas pelo uso de uma única projeção

Figura 10 – Patches PBDAM (a) e sistema de coordenadas locais do VGIS (b). (LINDSTROM, KOLLER, RIBARSKY, HODGES, BOSH, ET AL., 1997) e (CIGNONI, GANOVELLI, ET AL., 2003B)



de mapa. Kooima (2008) apresenta uma técnica para renderizar terrenos esféricos e compor (inserir) conjuntos de dados de diferentes resoluções. A técnica é orientada para a GPU e usa uma subdivisão icosaédrica da esfera para minimizar a distorção da malha. A geração de malha tem dois estágios: no primeiro, uma malha grosseira é gerada na CPU usando uma técnica similar a ROAM; no último, a malha é refinada na GPU, onde as texturas de ponto flutuante atuam como dados vetoriais. Mahsman (2010) aplica o PGM à renderização de terreno esférico e também propõe uma solução para a composição de dados. Essa abordagem, no entanto, sofre de artefatos devido à subamostragem.

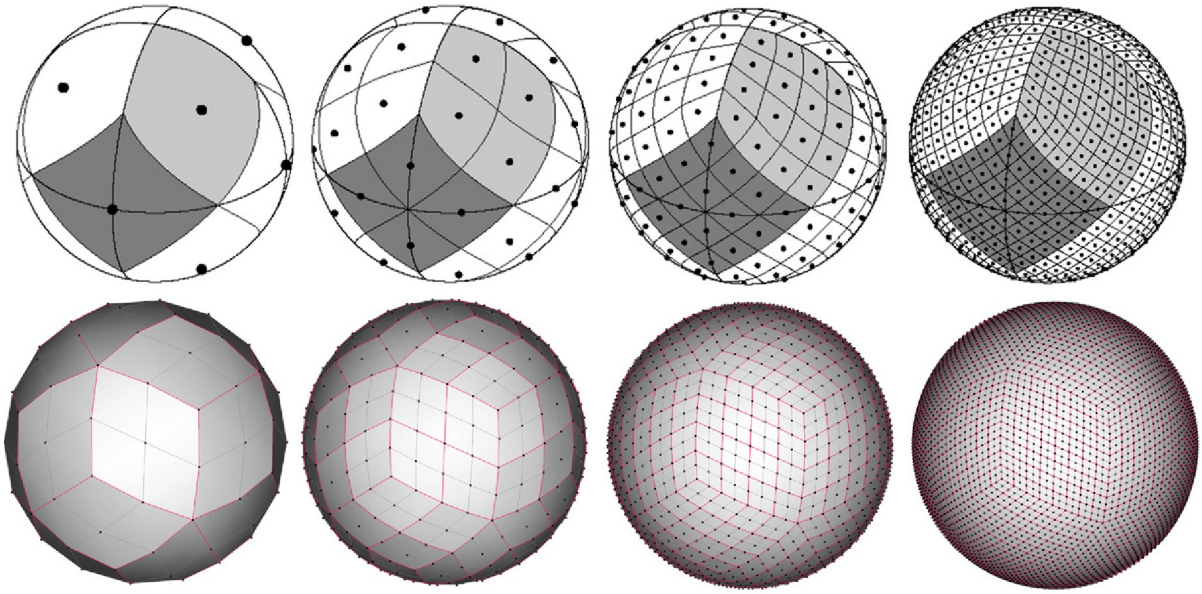
Figura 11 – Adaptação do geometry clipmap usando anéis hexagonais. (BHATTACHARJEE, 2010)



Crusta (BERNARDIN, COWGILL, ET AL., 2011) é um sistema de globo virtual desenvolvido para a visualização de conjuntos de dados submétricos e destinado à investigação geológica virtual. Sua principal contribuição é a tesselação do planeta com uma subdivisão hierárquica de um triacontedro rômbico (Figura 12) que fornece um grande grau de uniformidade

de malha. Da mesma forma, o trabalho de Westerteiger, Gerndt, et al. (2012) tessela o planeta usando os 12 patches quadrilaterais curvilíneos da estrutura HEALPix (Figura 12). Ambas as técnicas suportam composição de dados.

Figura 12 – Tesselação do Crusta (abaixo) e HEALPix (acima). (WESTERTEIGER, GERNDT, ET AL., 2012) e (BERNARDIN, COWGILL, ET AL., 2011)



3 TERRENO EM ESCALA PLANETÁRIA

3.1 VISÃO GERAL

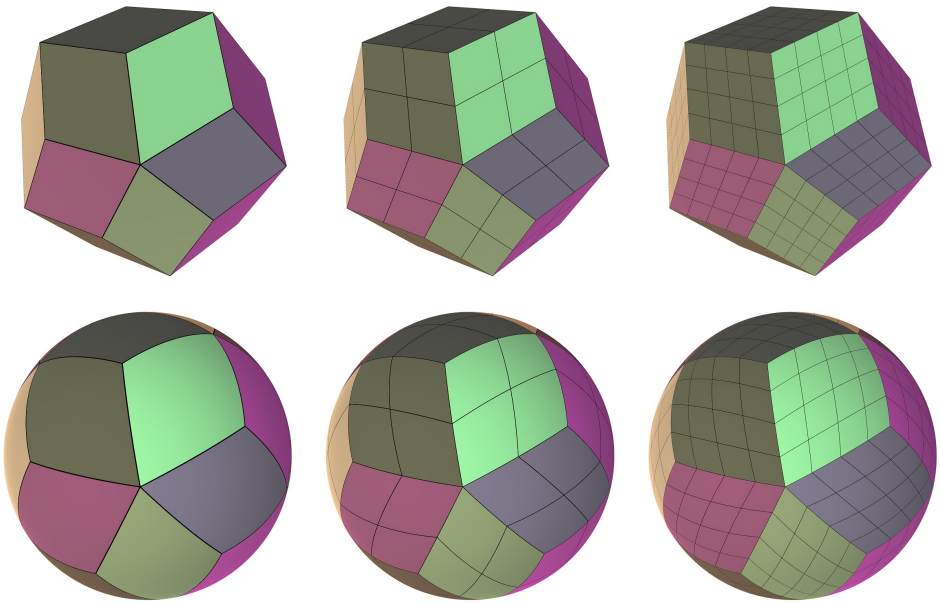
A motivação deste trabalho é descrever a implementação de um motor de renderização de terrenos em escala planetária que aborda tanto a visualização quanto a integração com uma biblioteca de física. Como a maioria das técnicas de renderização de terreno, nós empregamos estruturas quadtree que possuem várias finalidades. A primeira é fornecer gerenciamento de dados onde a hierarquia é usada para rastrear os dados que devem ser descarregados e carregados na memória (Seção 3.5). Ela também tem o objetivo de gerar uma malha dependente da posição do observador. Os nós são selecionados com base na distância até o visualizador, gerando uma malha que reduz continuamente sua resolução (Seção 3.3). Nesse processo, a hierarquia também é útil para executar testes de culling baseados no frustum do observador e no horizonte do planeta (Seção 3.2.2). Com uma travessia de cima para baixo e poucos testes de interseção, podemos remover grandes regiões, economizando eficientemente os tempos de CPU e GPU. A mesma estrutura também é usada no gerenciamento de malhas de colisão, o que é discutido mais adiante na Seção 3.4.

Da mesma forma que Crusta (BERNARDIN, COWGILL, ET AL., 2011), usamos um triacontaedro rômbico de 30 lados (Figura 13) como o poliedro base para tesselar a superfície do planeta. Esta abordagem reduz as distorções na malha quando comparado a uma tesselação baseada em um cubo e gera uma malha altamente uniforme. Essa abordagem também evita as singularidades nos polos gerados pela tesselação do grid geográfico. Uma revisão mais detalhada sobre a tesselação de esferas pode ser encontrada no capítulo 2. Nós representamos cada face rômbica base com uma quadtree, resultando em um total de 30 estruturas de árvores. Quadtrees são percorridas no início de cada quadro para selecionar nós para renderizar a malha final do terreno. Culling de frustum e de horizonte eliminam a necessidade de atravessar todas as 30 estruturas quando o observador está ao nível do solo. Embora uma representação precisa da superfície da Terra, como o elipsóide de referência WGS84, seja ideal (Seção 2.2), neste trabalho usamos a esfera como uma aproximação. Essa decisão simplifica muito nossa definição de limites de nós e testes de interseção. Para evitar rachaduras e junções em T, empregamos a mesma técnica de transição dependente da posição do observador encontrada em Strugar (2009). Nas seções a seguir, fornecemos uma descrição detalhada da nossa técnica.

3.2 REPRESENTAÇÃO DA QUADTREE

O nó raiz de cada quadtree é inicializado com os quatro vértices não projetados da face rômbica que ele representa. Esses vértices base são armazenados para cada nó para derivar nós filhos quando necessário. Quando um nó deve ser subdividido, o ponto médio de cada aresta

Figura 13 – Base rhombic triacontahedron and its projection into the sphere. Each rhombic face is represented by a quadtree.



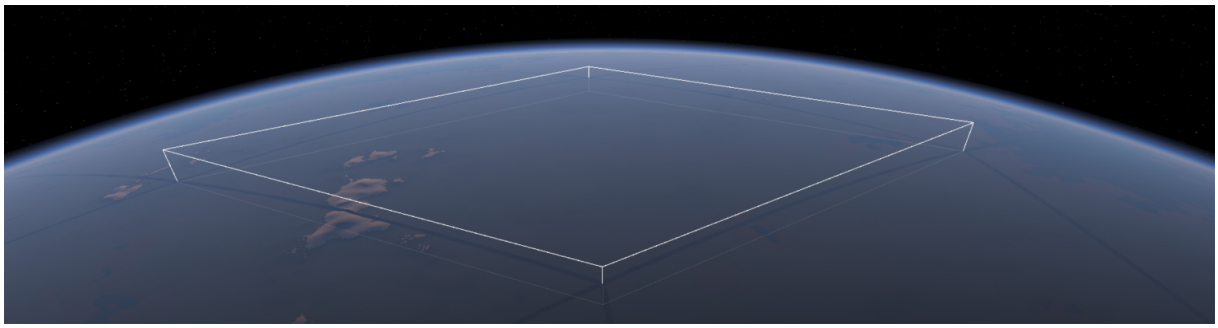
e a posição central do patch são calculados e, juntamente com os vértices de base, são usados para inicializar os quatro nós filhos. Como nossa definição de patch não é paramétrica como em HEALPix (WESTERTEIGER, GERNDT, ET AL., 2012), os vértices base de um nó devem estar disponíveis na GPU sempre que a posição global dos vértices deva ser computada. Além disso, armazenamos para cada nó um mapa de normal e a malha final em espaço global. Portanto, não adotamos a textura tradicional do heightmap. O raciocínio por trás dessa decisão é detalhado na Seção 3.3. Além disso, dois raios que representam as transformações mínimas e máximas de um nó também são armazenados. Esses raios são então usados para calcular os limites que serão usados para seleção e culling de nós.

3.2.1 Limites dos Nós

Como os patches base do triacontaedro rômbico, como o nome sugere, são losangos não quadrados, simplesmente representar os limites do nó com uma caixa delimitadora alinhada aos eixos não é preciso. Além disso, as caixas delimitadoras alinhadas aos eixos apresentam uma precisão baixa quando representam trechos de terreno esféricos que são distribuídos com diferentes orientações. Portanto, definimos os limites dos nós como sendo compostos por seis planos, a fim de fornecer uma representação precisa do nó e garantir que os nós filhos estejam totalmente contidos dentro dos limites de seus pais. Kooima (2008) aplica uma idéia semelhante representando trechos com uma concha de superfície triangular que é composta de três planos e dois raios. Embora pudéssemos seguir o mesmo princípio e representar nossos patches

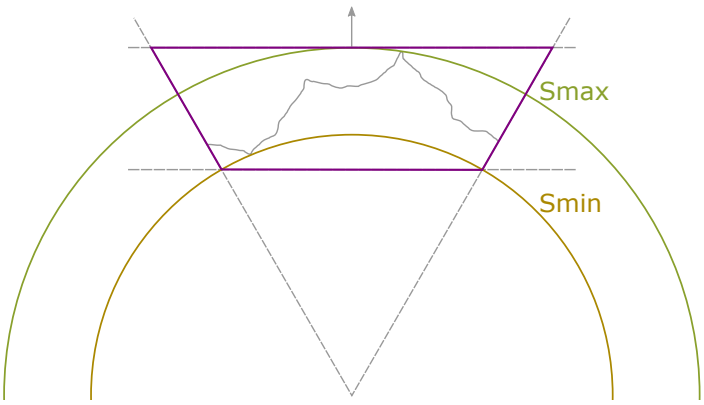
rômbicos com quatro planos e dois raios, optamos por seis planos, devido a testes de interseção simplificados. A Figura 14 mostra as arestas geradas pelas interseções dos seis planos delimitadores.

Figura 14 – Realce dos limites de um nó.



Como mostrado na Figura 15, os quatro planos laterais são definidos pelo centro da esfera e pelas arestas formadas pelos vértices da base do nó. Os planos superior e inferior são calculados usando as esferas com os raios máximo (S_{max}) e mínimo (S_{min}), respectivamente. O plano superior é de fato um plano tangente à esfera S_{max} e, portanto, pode ser definido pelo vetor normal no centro do nó e a posição central projetada em S_{max}. Além disso, os planos superior e inferior são paralelos e, portanto, o plano inferior também pode ser definido pelo mesmo vetor normal central. No entanto, um dos vértices na diagonal mais longa deve ser projetado em S_{min} e a posição resultante deve ser usada no lugar.

Figura 15 – Os limites de um nó são definidos pela interseção de seis planos.



Os planos são inicialmente calculados com S_{max} e S_{min} definidos para os seus valores padrão, gerando um limite que se estende de alturas mínimas e máximas predefinidas. Isso garante que o nó seja selecionado para refinamento, mesmo que suas alturas S_{max} e S_{min} não sejam conhecidas. Os planos delimitadores são recalculados assim que a malha em espaço global do nó estiver pronta e as alturas máxima e mínima estiverem disponíveis. Finalmente,

os testes de interseção e as operações de culling podem ser eficientemente realizadas quando os vértices e a esfera delimitadora do volume delimitador estão disponíveis e, portanto, também são computados e armazenados para uso posterior.

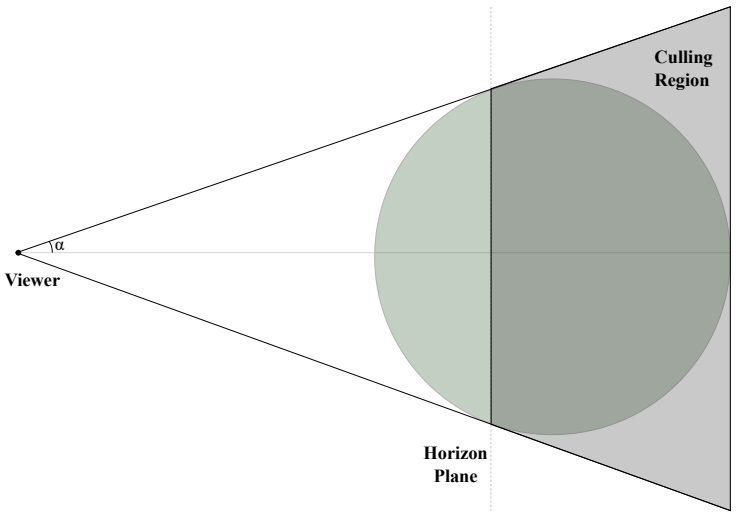
3.2.2 Testes de Interseção e Culling

Para realizar a seleção de nós com base na distância precisa para o observador, devemos ser capazes de intersectar um nó com uma esfera. Isto é alcançado aplicando o Separating Axis Theorem (SAT) (GOTTSCHALK, 1996). Embora o SAT possa ser usado para detectar colisões entre duas formas convexas, este proceso é caro. Uma otimização possível e um tanto eficiente é verificar se quaisquer vértices do limite estão contidos pela esfera e somente executar o teste SAT se todos os vértices falharem nessa verificação antecipada.

Percorrer a árvore para encontrar um nó folha que contenha um ponto pode ser útil em algumas situações. Abordamos esse requisito armazenando todos os planos delimitadores com a mesma orientação, ou seja, apontando para o exterior do volume delimitador. Dessa forma, descobrir se um nó contém um ponto pode ser reduzido de forma eficiente para verificar se o ponto está no mesmo lado de todos os planos, o que pode ser realizado calculando o produto escalar entre o plano e o ponto de consulta.

O culling baseado no frustum do observador é realizado por testes conservadores de interseção, isto é, alguns nós oclusos ainda podem ser selecionados para renderização. Nós efetivamente conseguimos isso verificando se todos os oito vértices de um limite estão do lado de fora de um único plano do frustum. Outra otimização empregada é verificar se o limite está totalmente contido no frustum e repassar este parâmetro para que os testes de culling de frustum possam ser ignorados para os nós filhos.

Figura 16 – Culling baseado no horizonte do planeta.



Também empregamos uma técnica chamada culling de horizonte para reduzir ainda mais o número de nós a serem renderizados, o que é ilustrado na Figura 16. O culling de horizonte tem dois passos. Primeiro, dada a posição da câmera, e o raio e posição do planeta, calculamos o cone do horizonte, que é representado por um ângulo de abertura α e um plano de horizonte. Esta informação é então usada para determinar se um nó está por trás do cone e pode, assim, ser descartado. Realizamos esse teste de seleção verificando se todos os vértices do volume delimitador estão atrás do plano do horizonte e dentro do cone infinito com o ângulo de abertura α.

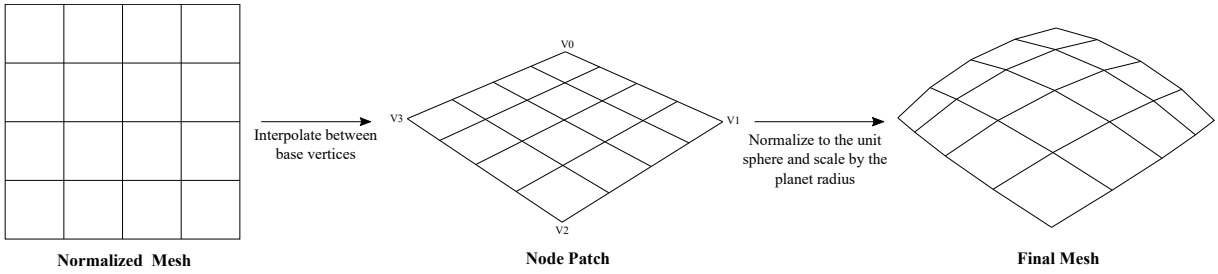
3.3 RENDERIZAÇÃO

O processo de renderização começa selecionando nós com base nos testes de visibilidade, isto é, no frustum e no horizonte, e na sua distância até o observador. O procedimento de seleção é aplicado a todas as quadtrees e produz um conjunto de nós selecionados que estão prontos para renderização, ou seja, o nó foi refinado corretamente e seu mapa de normal e a malha em espaço global estão disponíveis. A seleção baseada na distância resulta numa quadtree restrita, isto é, a diferença de nível entre dois nós vizinhos é sempre 1 ou 0, o que simplifica grandemente as transições de LOD. Esse conjunto de nós selecionados é então iterado e cada nó é desenhado instanciando um grid regular de resolução fixa. Ao desenhar um nó, informações sobre seu nível de LOD, por exemplo, o intervalo de visibilidade e dados do nó pai, por exemplo, mapa de normal e malha em espaço global, são usados para realizar transições de LOD suaves com base na distância para o observador.

Para renderizar um nó, devemos calcular a posição em espaço global de cada vértice do grid regular instanciado seguindo o processo mostrado na Figura 17. Como afirmado anteriormente, cada nó armazena os vértices do patch que ele representa. O grid regular é gerado para que todos os vértices sejam normalizados e, assim, podemos mapear o grid para o patch do nó simplesmente aplicando interpolação bilinear com os quatro vértices da base. A partir dessa posição, que fica na superfície do triacontedro rômbico, podemos calcular o vetor normal da superfície. Esse vetor normal deve ser calculado com precisão dupla, pois é usado no cálculo final da posição global e, dadas as dimensões dos planetas, cálculos de precisão simples se traduzirão em dezenas de metros de erro na superfície do planeta. Finalmente, calculamos a posição final do mundo simplesmente multiplicando o vetor normal da superfície do vértice pela soma do raio do planeta e o deslocamento da altura.

Seguindo a abordagem tradicional de usar mapas de altura para armazenar os dados de elevação, esse processo teria que ser executado a cada quadro, uma vez para cada vértice e duas vezes para vértices que estão em uma região de transição (Seção 3.3.2). Como usamos aritmética em precisão dupla e GPUs de uso comum possuem um desempenho de precisão dupla excepcionalmente menor, decidimos armazenar a posição final em espaço global em vez de um

Figura 17 – Processo para calcular a malha em espaço global de um nó. Um grid regular normalizado é interpolado em relação aos vértices base do nó e, em seguida, projetado na superfície do planeta.



único valor de altura. Embora essa decisão tenha aumentado os requisitos de armazenamento, o desempenho foi bastante aprimorado ao simplesmente amostrar a posição final do mundo em vez de executar cálculos em cada quadro. Além disso, como descrito na seção 3.4, ao armazenar uma malha em espaço global, a geração da malha de colisão para um nó é reduzida para a leitura dos dados da memória da GPU. Finalmente, armazenamos cada malha em relação à posição central inicial do nó. Isso nos permite usar precisão única para a malha, diminuindo o espaço ocupado em memória.

3.3.1 Seleção de Nós

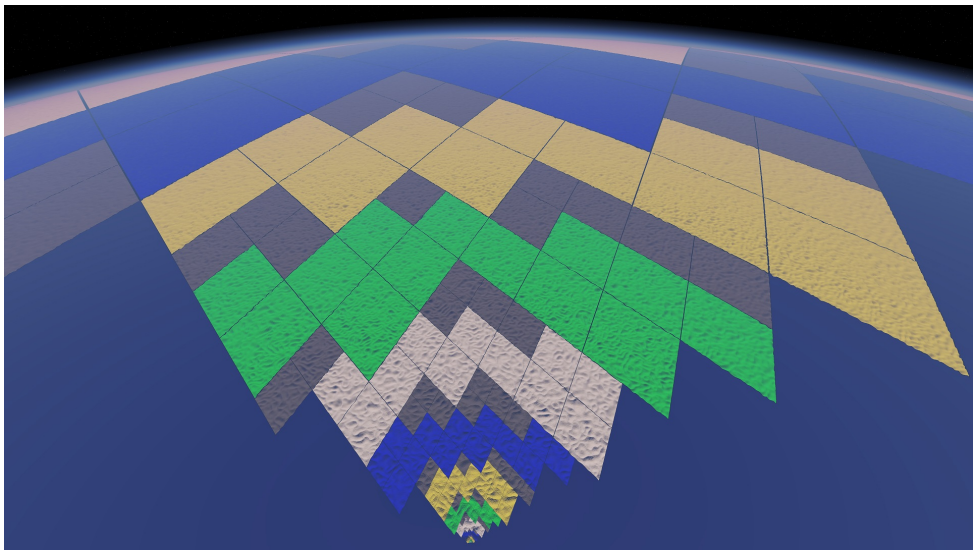
Nossa seleção de nós é baseada principalmente em Strugar (2009). Cada nível da quadtree representa um nível de detalhamento e possui um intervalo de visualização composto de uma distância de visualização inicial e a distância inicial do nível de detalhamento subsequente. O primeiro e o último níveis são casos especiais em que a distância inicial começa em zero e a distância final termina em infinito. O procedimento de seleção (Listagem 3.1) percorre a árvore em profundidade através de uma função recursiva. Um nó é selecionado para renderização quando passa com êxito nos testes de visibilidade e está dentro do seu intervalo de visualização de nível de detalhe. Os nós que não passam nos testes de visibilidade são descartados e seus filhos podem ser ignorados com segurança. Caso contrário, se um nó não estiver dentro de seu intervalo de visualização, o método continuará sua execução verificando recursivamente os nós filhos. No entanto, quando um nó filho é completamente contido no intervalo de visão de seu pai, ele será selecionado como um nó parcial, ou seja, ele será renderizado na metade de sua resolução para corresponder à resolução de nível de detalhe dos pais. A seleção de nós pode ser ainda mais otimizada, ignorando-se os testes de visibilidade para nós filhos de pais que estão completamente contidos dentro do frustum de visualização e fora do cone do horizonte. A figura 18 mostra um conjunto de nós selecionados para observador no nível do solo.

Listing 3.1: Selection procedure pseudocode.

```
void Select(QuadTreeNode node, bool parentFullyInsideFrustum, bool
    parentFullyOutsideHorizonCulling) {
    // Frustum culling
    bool allInsideFustum = parentFullyInsideFrustum;
    if (frustumCulling && !parentFullyInsideFrustum && !IntersectsWithFrustum(node, out
        allInsideFustum))
        return;

    // Horizon culling
    bool fullyOutsideHorizonCulling = parentFullyOutsideHorizonCulling;
    if (horizonCulling && !parentFullyOutsideHorizonCulling && IsCulledByHorizonCone(node,
        out fullyOutsideHorizonCulling))
        return;

    float startingViewDist = viewRanges[node.depth];

    // Only subdivide if this is not a leaf and it is out of its range
    if (!node.IsLeaf() && IntersectsWithSphere(node, cameraPosition, startingViewDist)) {
        if (!node.HasBeenRefined()) {
            node.Refine();
            AddToSelection(node);
        }
        else if (!node.AreChildrenReadyForRendering())
            AddToSelection(node);
        else {
            for (int i = 0; i < 4; i++) {
                if (IntersectsWithSphere(node.children[i], cameraPosition, startingViewDist))
                    Select(node.Children[i], allInsideFustum, fullyOutsideHorizonCulling);
                else
                    AddToSelectionAsPartial(node.children[i]);
            }
        }
    }
    else
        AddToSelection(node);
}
```

Para gerar uma malha livre de rachaduras, devemos garantir que os intervalos de visualização de níveis de detalhamento sejam grandes o suficiente para caber pelo menos um nó completo. Isto é conseguido atualizando constantemente os intervalos de visão de acordo com o tamanho máximo da diagonal dos níveis de LOD, de modo que as alturas sejam levadas em consideração. Quando um nó é gerado e seus limites foram computados, calculamos a diagonal máxima de seus limites e atualizamos cada intervalo de visão de nível de detalhe subsequente. Portanto, os intervalos de visualização são atualizados à medida que o espectador se movimenta pelo planeta. Isto é particularmente necessário para garantir que a seleção seja restrita e renderizar corretamente as regiões onde os nós apresentam uma diferença grande entre suas dimensões horizontais e verticais, como por exemplo regiões montanhosas.

Figura 18 – Seleção da quadtree excluindo os nós fora do frustum e de cone do horizonte. O visualizador está na parte de baixo olhando para o horizonte ao topo. Cores diferentes representam diferentes níveis de LOD. Os nós cinzas são parcialmente selecionados e, portanto, são renderizados com metade da resolução da malha.



3.3.2 Transições

Transições entre diferentes níveis de detalhe também são tratadas de acordo com a abordagem de Strugar (2009). Os nós de maior nível de detalhe são gradualmente transformados na resolução mais baixa do nível subsequente em relação à posição do observador. Os vértices são processados individualmente e um valor paramétrico k é computado com base no intervalo de visualização do nó e na distância do vértice ao observador. A Figura 19 mostra uma visualização de k, onde metade do intervalo de visualização de cada nível de LOD é transformado na resolução de seus pais. Como mostrado na Figura 20, o valor k é então usado para gradualmente transformar os vértices com índices ímpares em seus vértices vizinhos, gerando triângulos degenerados quando $k = 1$. Esse processo efetivamente remove as rachaduras na malha final, mas ainda não fornece transições suaves por conta própria. Portanto, os dados do nó e de seu pai, ou seja, a malha em espaço global e o mapa de normal, devem ser gradualmente interpolados em relação ao mesmo parâmetro k. Para conseguir isso, os dados de ambos os nós devem estar disponíveis na GPU, bem como o quadrante ocupado pelo nó dentro de seu pai.

A abordagem descrita requer que os dados sejam amostrados com interpolação bilinear, de modo que os vértices nas regiões de transição se transformem continuamente entre diferentes amostras. Isso requer a malha em espaço global e os dados do mapa de normal para incluir amostras extras em suas bordas. As amostras extras se sobreporão aos nós vizinhos, garantindo que as amostras nas bordas dos nós vizinhos com o mesmo nível de detalhe combinem perfeitamente entre si ao usar a interpolação bilinear. As transições entre duas quadtrees

Figura 19 – Seleção de nós para um observador no nível do solo posicionado no centro da imagem (esquerda) e a respectiva visualização do parâmetro k (direita) onde preto representa $k = 0$ e branco $k = 1$. Em regiões brancas, a resolução da malha corresponde à resolução do próximo nível de LOD.

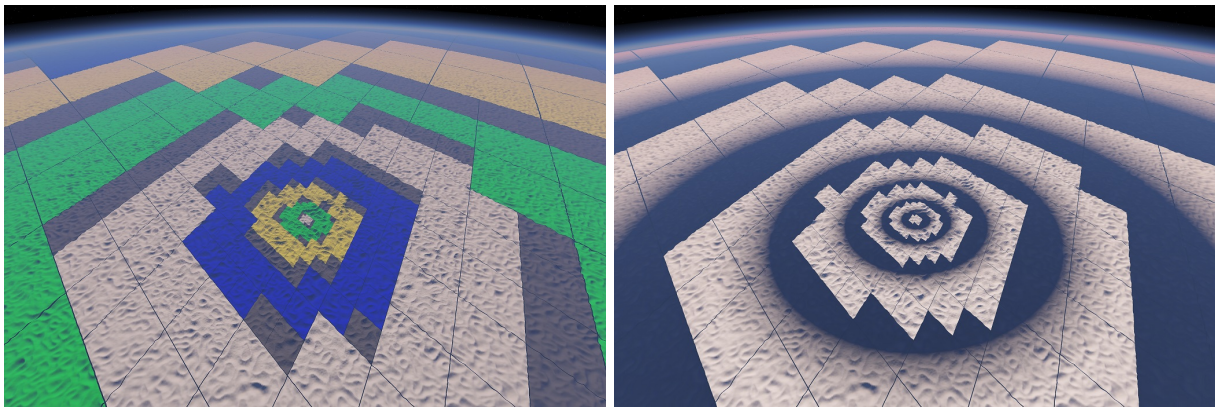
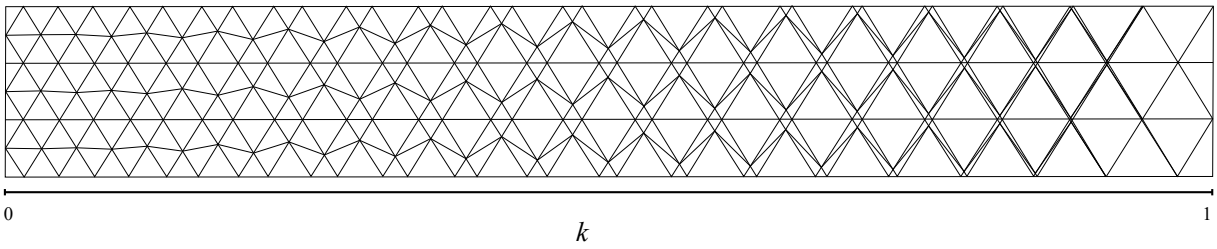


Figura 20 – Transição entre dois níveis de LOD. k cresce da esquerda para direita.



também são resolvidas por essa abordagem, desde que as amostras de extras sobreponham os dados das quadtrees vizinhas.

3.4 Malha de Colisão

A maioria das bibliotecas de física inclui um elemento de forma focada em terrenos, que fornece simulação física otimizada para mapas de altura. No entanto, essa forma de colisão especializada geralmente considera apenas mapas de altura dispostos em um plano e, portanto, não podem ser facilmente adaptados a terrenos esféricos. A solução natural para isso é a forma de malha triangular. Note, no entanto, que a maioria das bibliotecas fornece suporte triangular convexo e côncavo e que as malhas de colisão de terreno devem ser côncavas. Além disso, outro problema encontrado com o fornecimento de simulação física em escalas planetárias é a instabilidade numérica. A maioria das bibliotecas físicas é desenvolvida com o desenvolvimento de jogos em mente e, portanto, é implementada naturalmente no formato de ponto flutuante de precisão simples por motivos de desempenho. Além disso, há uma baixa demanda por simulações

de física de alta precisão, pois a maioria dos jogos está confinada a pequenos cenários que permanecem dentro da precisão submilimétrica do formato de ponto flutuante de precisão simples. Ainda assim, algumas bibliotecas de física, como a Bullet (COUMANS, 2018), suportam simulação de precisão dupla.

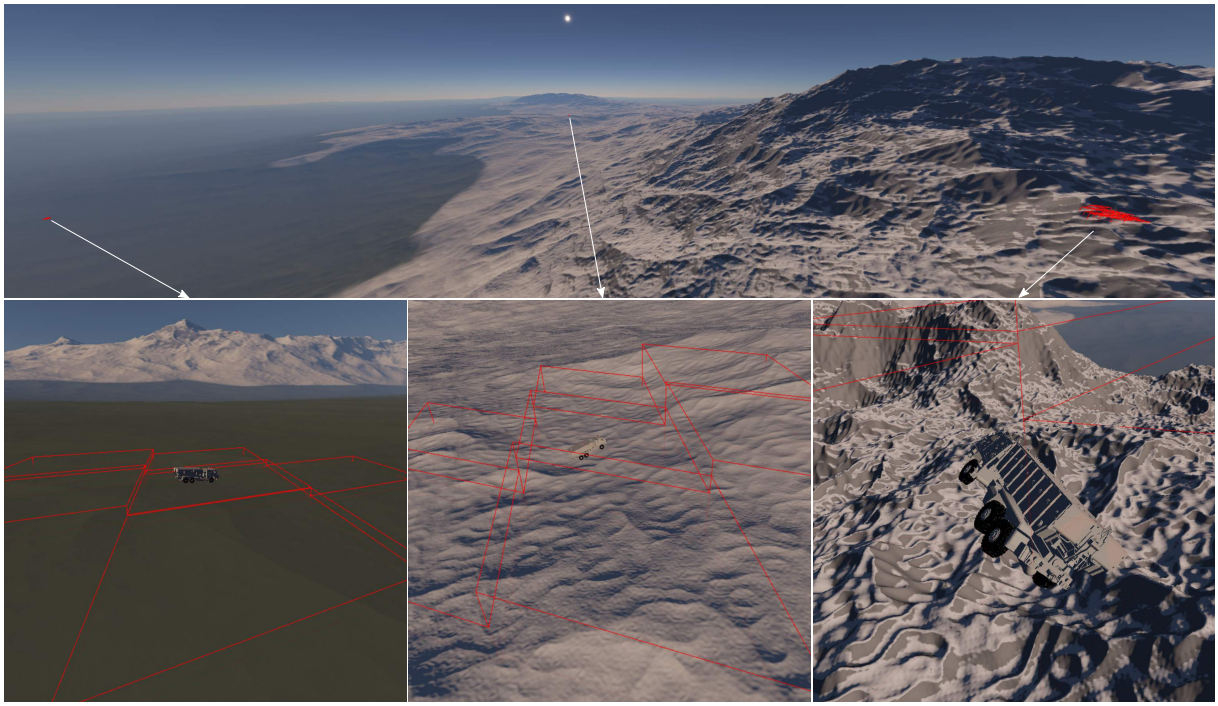
Neste trabalho, compilamos a biblioteca de física Bullet (COUMANS, 2018) com precisão dupla e aplicamos sua forma de colisão de malha triangular côncava para fornecer colisões entre objetos e o terreno. A compilação de precisão dupla, embora adicione custos de desempenho, elimina a necessidade de implementar técnicas para contornar problemas de precisão com números de ponto flutuante de precisão simples. Portanto, a integração com nosso sistema de terrenos torna-se direta, pois a malha em espaço global de um determinado nó já está disponível na memória da GPU. Nossa solução final, portanto, requer apenas a leitura dos dados da malha, compensando-a pela posição central inicial do nó e criando uma forma de colisão de malha triangular com orientação de identidade e posicionada na origem – os dados já estão no espaço global e não exigem outras transformações.

Assim como na renderização, não é possível manter grandes porçoes do terreno na memória para detecção de colisão e, por questões de desempenho, devemos carregar e descarregar dinamicamente dados do terreno e malhas de colisão. Assim, selecionamos nós com base na posição de objetos que exigem interações com o terreno. A figura 21 mostra a seleção de nós resultante para três objetos espalhados pelo terreno. Esse processo pode ser executado de forma eficiente percorrendo recursivamente e em profundidade as estruturas quadtree e processando apenas nós que estão dentro de um raio R de qualquer objeto relevante. Os nós em um nível de detalhe configurável L são então selecionados para geração de malha de colisão.

Ao ajustar o nível de seleção L, podemos controlar a fidelidade da malha de colisão e, assim, reduzir significativamente o custo de desempenho da seleção de nós, geração de malha de colisão, leitura de dados e simulação física em geral. Este controle é necessário para suportar uma visualização altamente detalhada dos dados em altíssima resolução, juntamente com uma simulação física em níveis de detalhes mais grosseiros, pois na maioria das situações, detalhes muito pequenos podem ser ignorados com segurança por terem pouca contribuição na detecção de colisão. O efeito colateral claro de ter diferentes resoluções para a colisão e a malha renderizada real, como mostrado na Figura 22, é a sobreposição visual entre o objeto e o terreno. No entanto, esse efeito é sutil e pode ser ignorado com segurança para a maioria dos casos de uso.

O processo de seleção de nós para geração de malha de colisão é executado em uma thread secundária para reduzir o impacto no desempenho causado por percorrer as estruturas quadtree com uma alta contagem de objetos. A figura 23 ilustra o ponto de sincronização entre a thread principal e a de seleção física. O uso de uma thread secundária requer que seções de código críticas que modificam dados da árvore e dos nós, como subdivisão de nós, sejam encapsuladas por um bloqueio de exclusão mútua. Além disso, a thread secundária percorre a árvore e apenas subdivide os nós conforme necessário – a subdivisão cria um novo nó calculando seus vértices base e limites iniciais. Os dados de elevação, ou seja, o buffer de malha em espaço

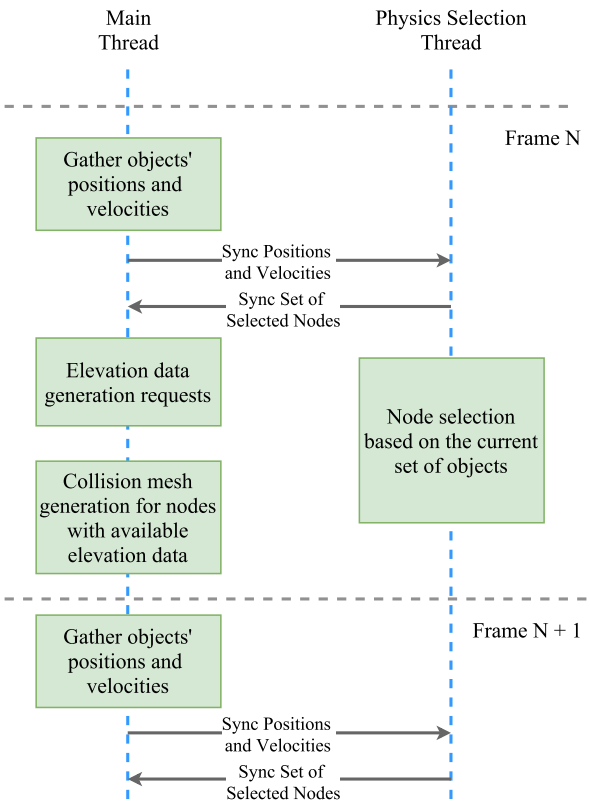
Figura 21 – Realce de nós selecionados para geração da malha de colisão para três objetos dispersos a aproximadamente 33 km de distância.



global e a malha de colisão ainda são gerados para os nós selecionados na thread principal.

Figura 22 – Sobreposição entre um objeto e a malha de terreno renderizada gerada por valores baixos do nível de seleção *L*.

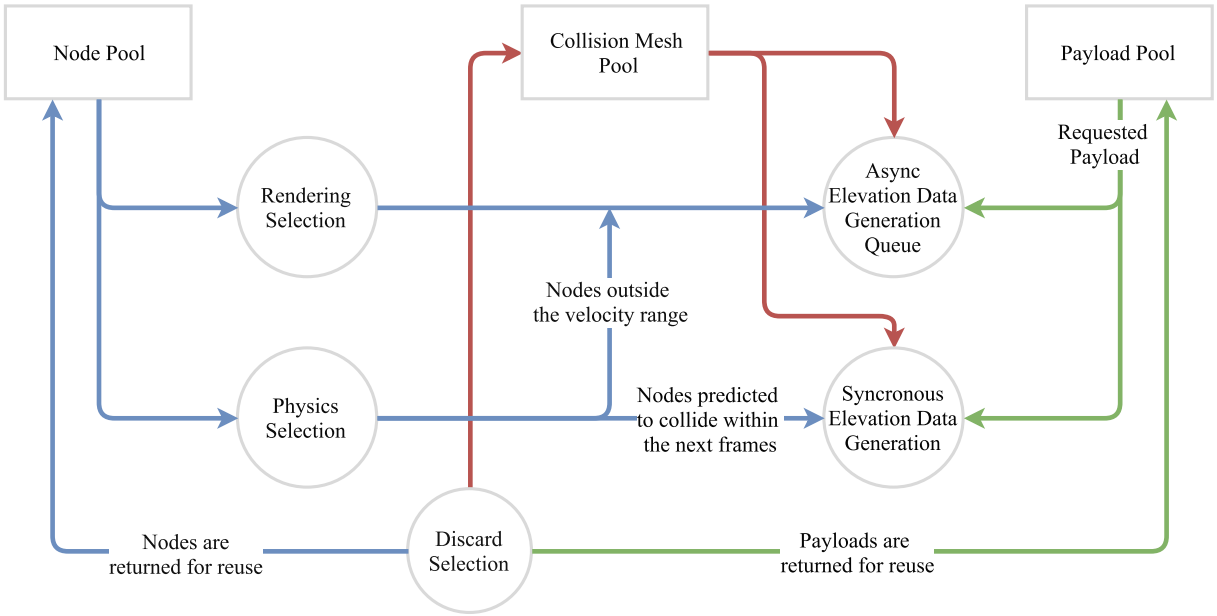
Figura 23 – Pontos de sincronização entre a thread principal e de seleção física.



3.5 Gerenciamento de Dados

O gerenciamento eficiente de dados é fundamental para aplicações que visam visualizar dados em escalas planetárias. Da mesma forma que a maioria dos algoritmos de renderização de terreno, empregamos técnicas de gerenciamento de dados para manter o uso de memória em níveis aceitáveis. Isso é essencial, pois o requisito de memória para armazenar dados de terreno na GPU é alto. Além disso, o fato de que objetos que exigem colisão possam estar dispersos por todo o planeta reforça a necessidade de um gerenciamento eficiente dos dados do terreno e das malhas de colisão. A figura 24 fornece uma visão geral do fluxo de dados dentro do nosso sistema, que é detalhado nas seções seguintes.

Figura 24 – Diagrama ilustrando o fluxo de dados. Quando necessário, as seleções de renderização e física solicitam nós da pool de nós. Esses nós recém-adicionados são então enviados para o processo de geração de dados de elevação, onde a carga útil de cada nó também é solicitada. Finalmente, a seleção periódica de descarte retorna a carga útil e a estrutura do nó a pool de objetos.



Os nós são gerados assim que são requisitados para renderização ou geração de malha de colisão. A geração pode ser dividida em dois estágios: subdivisão do nó e geração de carga útil, em que a carga útil se refere à malha em espaço global e ao mapa de normal. Um nó pode ser subdividido calculando os vértices base e o volume delimitador inicial de todos os seus filhos. A carga útil pode ser calculada como um todo, por exemplo, quando um nó é selecionado para renderizar tanto o mapa de normal quanto a malha em espaço global devem ser gerados, ou seletivamente, por exemplo, nós selecionados para colisão requerem apenas a geração da malha

em espaço global e assim a geração do mapa de normal pode ser ignorado. Portanto, a malha em espaço global, os dados de mapa de normal e o volume delimitador correto permanecem desconhecidos para os nós cuja carga útil não foi solicitada. Isso é útil ao selecionar nós para geração de malha de colisão, pois somente os nós selecionados devem ter sua carga gerada. Além disso, para regiões onde as estruturas são apenas percorridas para fins de colisão, ou seja, o observador não está próximo, os nós acima do nível de seleção física L provavelmente não terão sua carga útil na memória, reduzindo efetivamente o uso da memória.

A geração de carga útil pode ser síncrona, isto é, a aplicação irá parar até que os dados estejam prontos, ou assíncrona, isto é, o encadeamento principal continua sua execução e os dados estarão provavelmente prontos dentro de um a três quadros. Os nós selecionados para renderização são sempre gerados de forma assíncrona para fornecer taxas de quadros interativos estáveis, pois os dados são necessários no lado da CPU. Além disso, o pai de um nó é exibido em seu lugar enquanto o processo de geração está sendo executado para evitar o surgimento de buracos no terreno. No entanto, as rachaduras ainda estarão visíveis enquanto os dados não estiverem prontos, pois a seleção do nó pai de um nó interrompe a seleção da quadtree restrita. Dependendo do tempo necessário para gerar a carga útil do nó, pops de nível de detalhe também podem ocorrer. Além disso, a geração síncrona e assíncrona pode ser usada quando nós são selecionados para geração de malha de colisão. Dado o tempo do último quadro e a posição e velocidade de um objeto, podemos determinar se o objeto colidirá com um nó no próximo quadro. Portanto, os nós que são previstos para colidirem no próximo quadro são gerados de forma síncrona, enquanto outros nós podem ser gerados de forma assíncrona. Essa suposição não é válida para objetos cuja magnitude de velocidade ou direção muda abruptamente. Para garantir a colisão com esses objetos, a geração síncrona deve sempre ser utilizada.

3.5.1 Geração Assíncrona de Carga Útil

O processo de geração da carga útil de um nó começa armazenando o nó em uma fila de pedidos. Os nós são processados despachando compute shaders que gerarão a malha em espaço global e o mapa de normal. A fila é processada dentro de um orçamento por quadro, de modo a evitar os travamentos causados pelo envio de um grande número de compute shaders no mesmo quadro. Também controlamos o número de processos simultâneos de geração de nós para evitar sobrecarregar a GPU, o que resulta em taxas de quadros constantes mesmo para voos de alta velocidade no terreno. Como mencionado anteriormente, essa abordagem tem o efeito colateral de pops de nível de detalhe perceptíveis, especialmente em hardware de baixo custo, que podem ser reduzidos aumentando o orçamento de geração de nós. Os pops podem ser ainda mais reduzidos diminuindo a resolução do nó ou evitados completamente removendo a geração assíncrona, no entanto, isso resulta em efeitos indesejados para aplicativos de visualização em tempo real, como travamentos e interatividade reduzida.

3.5.2 Pool de objetos

Mantemos uma taxa quase constante de geração de nós e solicitações de descarte de nó, descartando nós com base na distância do observador. Aproveitamos esse aspecto mantendo uma pool de nós e de carga útil que reduz efetivamente a quantidade de alocação de memória por quadro. A carga útil do nó pode ser reutilizada, impactando positivamente no desempenho. O tamanho da pool pode ser ajustada para refletir a quantidade disponível de RAM e VRAM. Além disso, uma pool de malhas de colisão também é empregado para reduzir ainda mais as operações de alocação de memória.

Os nós são descartados conforme necessário para manter o consumo de memória em níveis aceitáveis. Experimentamos uma heurística baseada na distância e no tempo para determinar se um nó deveria ser descartado. Embora heurísticas mais elaboradas possam ser empregadas, descobrimos que ambas as abordagens são válidas e fornecem resultados satisfatórios. Primeiro, tentamos descartar um nó com base na última vez em que ele estava ativo, ou seja, selecionado para renderização ou geração de malha de colisão. Isso permitia controlar por quanto tempo um nó ficaria residente na memória principal. No entanto, os nós não selecionados pelo frustum também são descartados e, assim, o simples ato de girar a direção da visão descartaria nós que estão próximos do observador e que têm alta probabilidade de serem selecionados em quadros futuros. Outra questão com essa heurística simples é que a taxa de nós gerados se torna maior do que a taxa de nós descartados para voos de alta velocidade próximos ao nível do terreno. Em seguida, aplicamos uma abordagem baseada na distância para que os nós permanecessem residentes mesmo quando fora do frustum. Isso é conseguido descartando os nós em relação ao seu nível de detalhe e distância ao visualizador e aos objetos que exigem colisões.

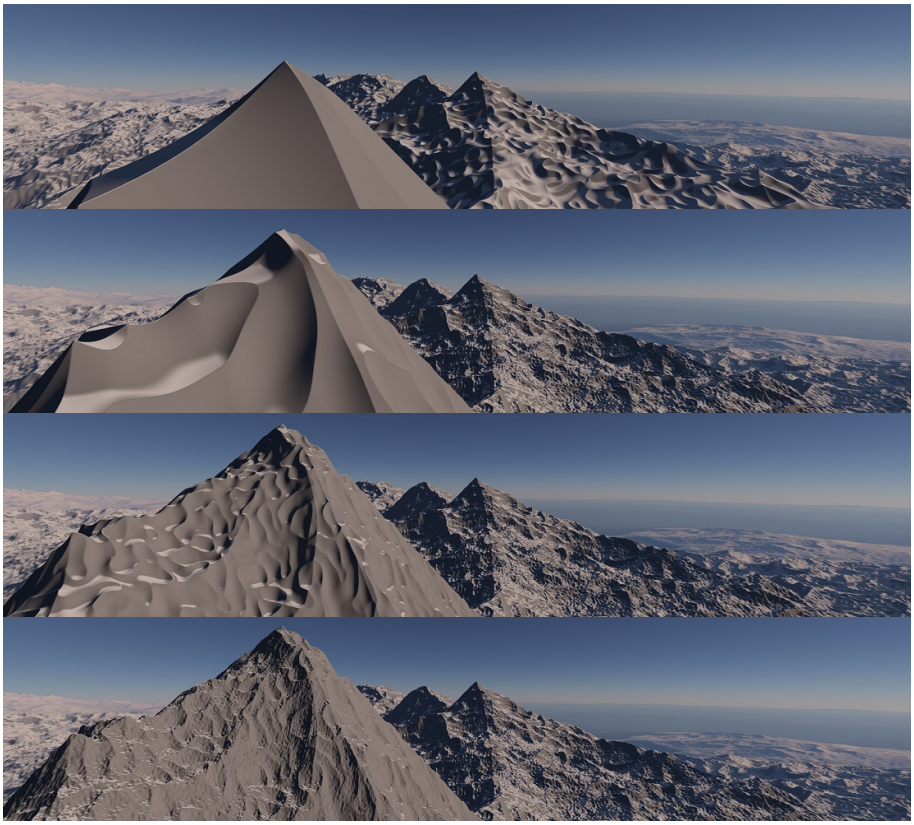
4 RESULTADOS E IMPLEMENTAÇÃO

Como a renderização sob demanda de grandes conjuntos de dados não é abordada neste trabalho, nós geramos proceduralmente dados de elevação com o uso de fractional Brownian motion (fBm), uma técnica popular para gerar detalhes em várias escalas (EBERT, MUSGRAVE, ET AL., 2002). Esta técnica, como mostrado na Figura 25, nos permite gerar dados de elevação em múltiplas escalas, incluindo continentes, montanhas e detalhes em escala de sub-metros, a fim de demonstrar o potencial da nossa técnica. Aplicamos doze camadas de ruído de gradiente para gerar a elevação final, onde quatro camadas geram uma forma grosseira do terreno com baixas frequências que é então usada para modular as oito camadas de alta frequência restantes. Os compute shaders são despachados para gerar os dados necessários, aproveitando o paralelismo maciço fornecido pelas GPUs. Além disso, dada a alta resolução dos dados gerados e, se negligenciarmos o fluxo de dados do disco, nossos testes cobrem com sucesso a renderização de conjuntos de dados em altíssima resolução. Além disso, o raio aproximado de 6371 km da Terra foi usado para modelar o planeta, exigindo que o processo de geração seja implementado com precisão dupla. Finalmente, dado o baixo desempenho de precisão dupla de GPUs de uso comuns e as muitas camadas de geração de conteúdo procedural empregadas, como apresentado na seção a seguir, o kernel de geração procedural é o principal gargalo do nosso sistema quando o observador se movimenta em alta velocidade e baixa altitude pelo terreno.

A técnica proposta foi implementada em C# dentro da engine de jogos Unity. Embora tenhamos conseguido integrar com sucesso o nosso motor de terreno com o pipeline de renderização da Unity, muitas limitações foram encontradas em todo o desenvolvimento. Por exemplo, o culling baseado no frustum não pode ser desativado para uma chamada de desenho específica e torna-se redundante, uma vez que já é aplicado ao selecionar nós para renderização. Além disso, a Unity não fornece meios para definir variáveis uniformes de precisão dupla – um parâmetro de entrada global dos programas de shader – e, por esse motivo, um buffer de memória constante contendo os dados de precisão dupla precisou ser criado e disponibilizado no shader. Além disso, a falta de simulação física de dupla precisão da Unity motivou o uso da biblioteca de física Bullet. O plugin BulletSharpUnity3D (DEANE, 2018) lidou com a integração da Bullet com a Unity e foi modificado para funcionar com uma compilação de precisão dupla da Bullet. Finalmente, o código do shader foi escrito em HLSL, a High Level Shading Language para DirectX, que não fornece função de raiz quadrada de precisão dupla. Nós contornamos esse problema implementando uma função de raiz quadrada de precisão dupla usando o método Babilônico, aplicando o resultado da função de raiz quadrada de precisão simples como o palpite inicial para o método.

Um problema comum encontrado ao renderizar grandes cenários é o erro adicionado por cálculos com variáveis de precisão simples. Isto se manifesta como um forte tremor de vértices e deformação de malhas que se torna impossível ignorar em escalas planetárias. Inspirado por Thorne (2006), abordamos esse problema aplicando um esquema de origem flutuante,

Figura 25 – Geração procedural de dados de elevação é obtida empilhando várias camadas de ruído gradiente com diferentes frequências e amplitudes. O número de camadas empregadas, de cima para baixo, são: 8, 9, 10 e 12, respectivamente.



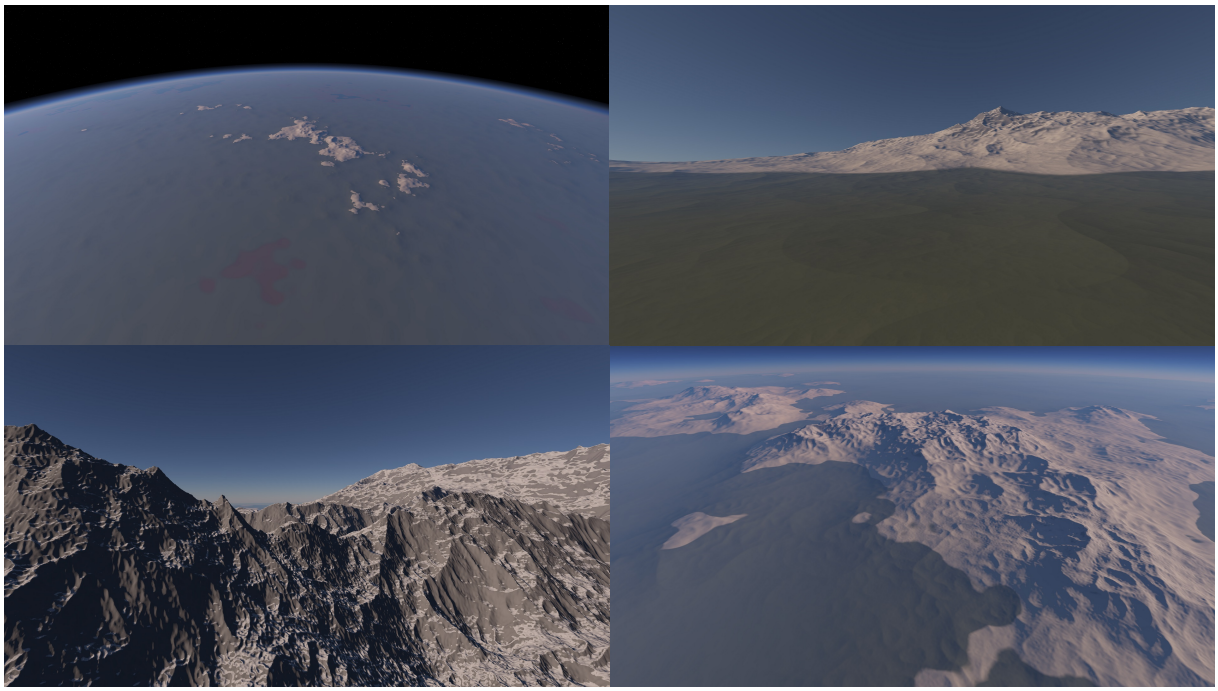
onde a origem do sistema é sempre mantida dentro de uma distância predefinida do observador. Quando o espectador se distancia mais do que essa distância predefinida da origem, nós traduzimos o mundo para que ele se torne centralizado no observador. Isso garante que o observador esteja sempre dentro de um intervalo de precisão suficiente para renderizar objetos próximos sem artefatos.

Nosso sistema fornece vários parâmetros de configuração. A resolução da malha do nó e a profundidade da árvore podem ser ajustadas para controlar a resolução da amostra (em metros) dos nós folha ou, em outras palavras, a resolução máxima exibida quando o visualizador está no nível do solo. Por exemplo, dada uma resolução de amostra de conjunto de dados e uma resolução de malha de nós desejada, a profundidade da árvore necessária pode ser calculada para que os nós folha tenham vértices suficientes para visualizar corretamente o conjunto de dados. Além disso, um multiplicador de alcance de visão pode ser controlado para aumentar o intervalo de seleção de todos os níveis de árvore, aumentando efetivamente a densidade da geometria.

4.1 Análise de Performance

O hardware em que o nosso sistema foi avaliado consiste de uma CPU Intel Core i7-6850k de 3.6GHz com 24GB de RAM e uma GPU Nvidia GTX 1080 Ti (11GB VRAM). Definimos uma animação do observador para ser usada como nosso principal benchmark para melhor avaliar o desempenho do nosso sistema. Ela consiste em uma animação flyover que tenta capturar diferentes situações extremas às quais o sistema poderia ser submetido. A figura 26 mostra diferentes quadros desta animação. Nesta animação, o espectador começa fora da atmosfera, aproxima-se rapidamente do terreno e permanece na mesma posição por alguns segundos. Em seguida, ele passa por uma série de cadeias de montanhas e lentamente se distancia do terreno até parar novamente em uma altitude mais alta. Finalmente, o espectador se move rapidamente para fora da atmosfera, onde a animação termina com uma visão ampla do planeta. Esta animação nos permite analisar as seguintes situações: aproximação rápida do terreno com mudanças constantes e repentinas de altitude; distanciamento do terreno; e vistas estacionárias em diferentes altitudes.

Figura 26 – Quadros da animação de benchmark em 0 (superior esquerdo), 7 (superior direito), 12 (inferior esquerdo) e 18 (inferior direito) segundos.



O desempenho está diretamente relacionado à quantidade e granularidade da geometria a ser renderizada. Profundidades de árvore mais altas exigirão mais etapas para a percorrer e, portanto, é naturalmente limitada pela CPU. No entanto, a profundidade da árvore também afeta o número de nós selecionados para renderização e, ao renderizar um número alto de nós,

a sobrecarga causada pelas chamadas de renderização da API começa a dominar o tempo de quadro final. Além disso, o acoplamento das profundidades de árvore inferiores com a malha de nós de alta resolução diminui a adaptabilidade da estrutura, gerando contagens superiores de triângulos. Portanto, os parâmetros apresentados devem ser ajustados para fornecer o melhor desempenho em cada hardware. Descobrimos que, para nosso hardware de teste, uma resolução da malha dos nós de 128x128 acoplada a uma profundidade de árvore máxima de 20 níveis resultou em um bom equilíbrio entre a quantidade de geometria e o número de nós selecionados a serem renderizados, fornecendo uma resolução de amostra aproximada de 3 cm. Além disso, o número máximo de despachos de geração de dados de elevação e processos de geração simultâneos também pode ser ajustado para evitar a sobrecarga da GPU. Ambos os parâmetros afetam diretamente a interatividade e o popping. Em nossos testes, adotamos os valores 8 e 32, respectivamente.

A Figura 27 mostra os resultados da animação de benchmark com a configuração mencionada anteriormente. Evidentemente, os dados mostram um aumento nos tempos totais e de seleção de nós quando mais nós são selecionados. Esse comportamento pode ser observado em torno de 7 e 18 segundos, onde a câmera está parada e não há processos de geração de nós em execução. Relacionando o número de processos simultâneos de geração de nós e os tempos totais de quadros, como mencionado anteriormente, fica claro que os picos a partir de 5 e 10 segundos são dominados por kernels de geração de dados procedurais. Ainda assim, o desempenho permanece abaixo de 10 ms para toda a animação e abaixo de 4 ms, considerando apenas as visualizações fixas. Isso mostra que nosso sistema é limitado pela disponibilidade de dados de elevação, indicando que, se acoplados a técnicas eficientes de transmissão de dados, os tempos de quadros seriam mais consistentes, deixando espaço suficiente para aumentar os dados de elevação bruta com detalhes processados.

Nós também fornecemos, na Figura 28, os tempos de quadros do nosso sistema simulando dados de elevação instantaneamente disponíveis, a fim de melhor analisar o desempenho da renderização. Ao parar a animação até que todos os processos de geração tenham terminado, podemos selecionar o número correto de nós para cada etapa da animação. Isso pode ser visto claramente ao contrastar a contagem de seleção de nós com a Figura 27 e mostra a quantidade de dados que estão sendo ignorados ao limitar o número de despachos de geração de dados de elevação por quadro. Mais uma vez, trata-se de um uma troca entre interatividade e qualidade visual instantânea, que pode ser reduzida ao otimizar o kernel de geração procedural. Além disso, o tempo de seleção leva em média 20% do tempo total do quadro. Essa contribuição permanece quase constante até mesmo para contagens baixas de seleção de nós, mostrando que os tempos de quadros podem ser melhorados apenas otimizando a renderização de nós.

Figura 27 – Resultados para uma única execução da animação de benchmark em nosso hardware de teste na resolução de tela de 1920x1080. Os dados são plotados em relação ao tempo da animação. O gráfico superior mostra os tempos do quadro, onde a seleção se refere ao tempo total gasto com a seleção de nós para renderização. O segundo gráfico (verde) mostra o número de nós selecionados para renderização. Os dois últimos gráficos consideram a geração procedural de dados de elevação, em que o número de despachados de compute shaders por quadro é mostrado em vermelho e o número de execuções simultâneas de compute shaders por quadro é mostrado em azul.

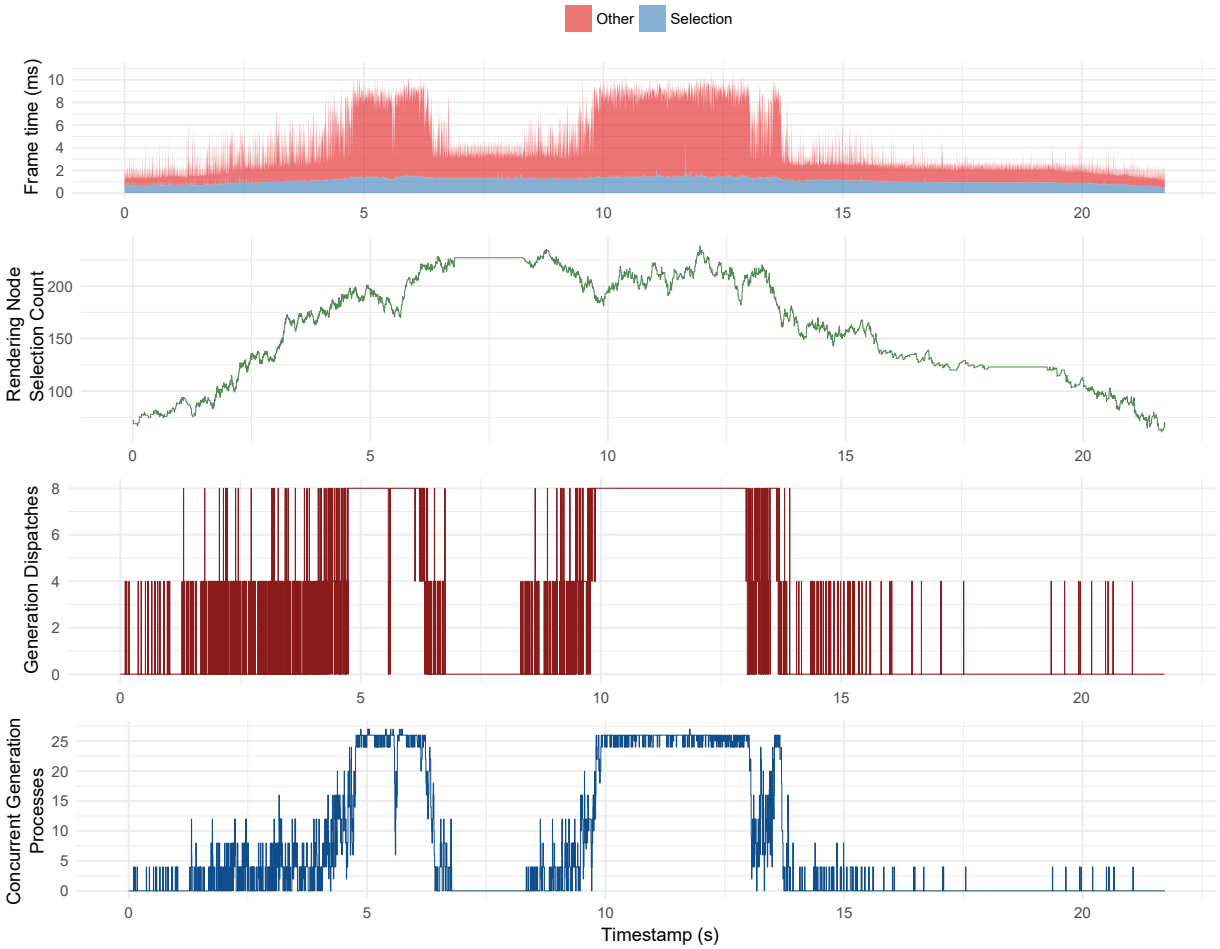
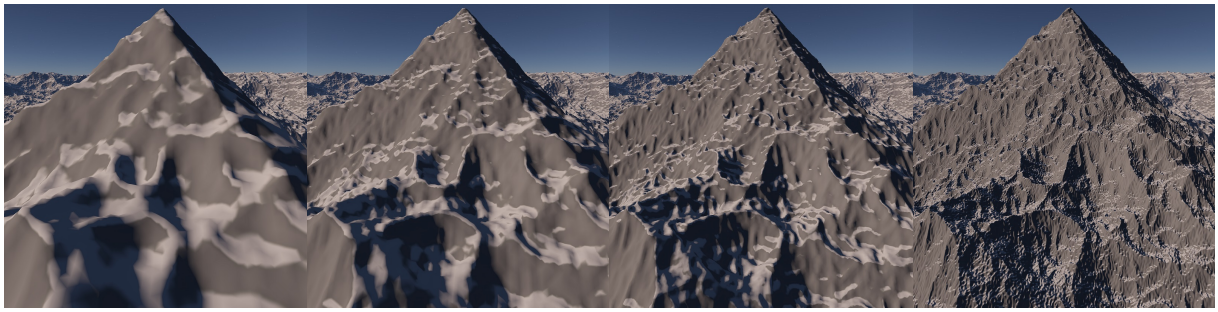


Figura 28 – Dados coletados com a animação animação interrompida em intervalos regulares e aguardando a conclusão de todo o processo de geração. Observe que isso não representa o comportamento normal de um usuário e se destina a fornecer tempos de quadros de renderização que não são afetados pelos kernels de geração procedural. A configuração é a mesma da Figura 27.

4.2 Resultados Visuais

Como mostrado nas Figuras 25, 26 e 30, a técnica proposta é capaz de renderizar planetas altamente detalhados com visualizações próximas e distantes. O uso da transição proposta por Strugar (2009) fornece transições contínuas entre níveis consecutivos de detalhes. Ainda assim, o alto custo de geração de dados procedurais de elevação exigiu o controle do número de processos de geração simultâneos para manter taxas constantes de quadros, resultando em pops de nível de detalhe, o que é ilustrado na Figura 29. Este efeito surge particularmente quando se aproxima do terreno a altas velocidades e é mais pronunciado em hardwares de baixo custo. Além disso, é útil controlar o nível máximo de seleção de renderização para fornecer taxas de quadros em tempo real ao direcionar hardwares de baixo custo. Assim, a figura 29 também mostra o resultado da qualidade visual de reduzir esse parâmetro.

Figura 29 – Quatro quadros consecutivos mostrando o efeito de popping de nível de detalhe produzido pelos voos de alta velocidade pelo terreno.



Um efeito visual que distingue a Terra de outros planetas é a dispersão da luz através de sua atmosfera. Os muitos comprimentos de onda da luz que entram são aleatoriamente espalhados e absorvidos em taxas diferentes pelas partículas que constituem a atmosfera. Este fenômeno é responsável pelas cores distintas encontradas no céu em diferentes momentos do dia e tem uma contribuição impressionante para o realismo das cenas externas, como destacado na Figura 30. Muitas aproximações para tornar isso efetivado foram propostas ao longo dos anos. Adotamos a abordagem proposta por Bruneton and Neyret (2008), que fornece uma representação precisa em tempo real da atmosfera e é responsável pela dispersão múltipla da luz solar.

Figura 30 – A dispersão atmosférica melhora muito o realismo das cenas externas.

5 CONCLUSÃO

Apresentamos uma técnica para renderizar eficientemente terrenos de tamanho planetário em tempo real. Ela fornece uma malha de terreno bastante uniforme usando o triacontaedro rômbico de 30 lados como o poliedro base para a tecelagem da esfera. Além disso, o sistema é capaz de renderizar planetas altamente detalhados, alcançando resolução de dados de elevação abaixo de um metro com nível de detalhe contínuo. Essas características tornam nossa técnica apropriada para aplicações de visualização. Além disso, aumentamos o escopo das possíveis aplicações, apresentando uma descrição detalhada do processo de integração da técnica de renderização proposta com uma biblioteca de física, tornando esse trabalho útil para aplicações que exigem interação entre objetos e o terreno. Além disso, também apresentamos uma abordagem para a geração procedural de dados de elevação, mantendo as taxas de quadros em tempo real.

5.1 DISCUSSÃO E TRABALHOS FUTUROS

Embora os resultados mostrem que nosso sistema pode renderizar com eficiência terrenos detalhados, a renderização de conjuntos de dados de elevação sob demanda não foi abordada neste trabalho e é claramente o recurso essencial ausente do nosso sistema. Como empregamos a tecelagem de esferas proposta em Crusta (BERNARDIN, COWGILL, ET AL., 2011), as mesmas etapas de processamento de dados usadas em seu trabalho podem ser aplicadas ao nosso sistema. Um software de pré-processamento poderia ser desenvolvido para carregar dados brutos de elevação e calcular a profundidade da árvore necessária para corresponder à resolução dos dados de entrada. Os nós folha das quadtrees poderiam então ser progressivamente refinados e as coordenadas cartesianas de seus vértices projetadas para coordenadas geográficas que seriam então usadas para amostrar o conjunto de dados de elevação. A árvore poderia então ser percorrida de baixo para cima com os dados de cada nó sendo obtidos através da redução da amostragem dos nós filhos. Esse processo forneceria uma representação hierárquica do conjunto de dados que corresponderia exatamente ao layout necessário para a renderização. As técnicas de transmissão de dados poderiam então ser empregadas para carregar dados de elevação, conforme exigido pelo aplicativo.

A aproximação de um planeta com uma esfera pode não apresentar precisão suficiente para algumas aplicações e, portanto, nossa solução pode ser adaptada para suportar outras aproximações, como o elipsóide de referência do World Geodetic System 1984 (WGS84). Adicionar suporte a planetas elipsoidais exigiria a redefinição de limites dos nós. A suposição de que o planeta é uniformemente dimensionado nos permite definir os limites como um conjunto de planos. No entanto, essa suposição não vale para os elipsóides, pois os eixos semi-menor e semi-maior são diferentes, escalonando-se não uniformemente o nó. Além disso, uma análise aprofundada das distorções de aspecto e área geradas pelo processo de projeção de dados de

elevação no triacontedro rômbico seria de grande interesse para quantificar com precisão a uniformidade da distribuição da amostra. Dimitrijevic, Lambers, et al. (2016), por exemplo, apresenta uma comparação interessante de projeções esféricas de mapas cúbicos, que podem servir de modelo para avaliar a distorção da projeção de dados.

A decisão de armazenar para cada nó uma malha em espaço global em vez de um heightmap tradicional reduziu com êxito o número de operações necessárias ao renderizar um nó e, assim, aumentou o desempenho da renderização. Ainda assim, essa decisão representa uma troca entre os tempos de renderização e o consumo de memória. Conseguimos diminuir o volume de memória armazenando posições relativas à posição central inicial do nó, o que nos permitiu usar uma malha de precisão simples. Esta solução, no entanto, apresenta limitações para o raio máximo do planeta sendo renderizado, dependendo da precisão requerida. Ao renderizar planetas maiores, um mapa de altura ou malha de precisão dupla deve ser usado.

A geração de dados procedurais na superfície de um planeta exigia o tratamento de problemas de precisão de ponto flutuante. Nós resolvemos esse problema implementando um ruído gradiente 3D de precisão dupla. No entanto, os resultados mostram que nossos compute shaders de geração procedural podem equivaler a metade do tempo de quadros quando um grande número de nós é necessário no mesmo quadro. Isso é causado principalmente pela relação aumentada entre o desempenho de precisão simples e dupla de GPUs padrão que não visam aplicações computação pesadas e, portanto, uma solução mais inteligente para problemas de precisão de geração de procedimentos deve ser investigada.

Embora a solução proposta de controlar manualmente o número de processos de geração de dados de elevação nos permita manter as taxas de quadros em tempo real, o pop de detalhes ocorrerá se não forem gerados nós suficientes em um quadro. Portanto, a aplicação de técnicas de geração procedural mais eficientes melhoraria muito os tempos de renderização e reduziria o pop de detalhes.

O desempenho de renderização do nosso sistema, se negligenciarmos a geração de dados de elevação, pode ser decomposto em três categorias: tempo gasto na seleção de nós, nas chamadas de API de renderização e o tempo gasto pela GPU para realmente renderizar a geometria. Há espaço para melhorias em todas as frentes mencionadas. Por exemplo, a seleção de nós pode ser paralelizada usando os núcleos de CPU disponíveis ou a GPU. A implementação de quadtree totalmente em GPU proposta por Dupuy, Iehl, et al. (2014) poderia ser adaptada para funcionar com nosso sistema e é um interessante tópico de pesquisa. No entanto, seu trabalho não aborda o carregamento de dados sob demanda e, portanto, a quadtree do lado da CPU ainda seria necessária para o gerenciamento do carregamento de dados e da malha de colisão. Além disso, a sobrecarga produzida pelas chamadas da API gráfica pode ser reduzida aproveitando as APIs gráficas modernas, como Vulkan e DirectX 12. Essa sobrecarga ainda pode ser diminuída com o uso de instanciamento em GPU fornecidas pelo OpenGL e DirectX 11. Finalmente, a taxa de geometria processada é limitada pela amostragem de dados de elevação, que é executada duas vezes para vértices em regiões de transição, e poderia ser melhorada ao lidar com

transições entre níveis de forma semelhante a Dupuy, Iehl, et al. (2014).

Utilizamos com sucesso as capacidades de computação das GPUs para gerar dados de elevação procedurais. Consequentemente, juntamente com o fluxo de dados, o processo apresentado também poderia ser usado para aumentar os dados grosseiros de elevação, adicionando detalhes procedurais de forma semelhante a Frasson, Engel, et al. (2016). Essa estrutura poderia ser ainda ampliada empregando dados vetoriais, por exemplo, estradas, rios e lagos, para controlar o processo de geração. Por exemplo, recursos vetoriais representando estradas poderiam ser disponibilizados na GPU para transformar o terreno em tempo de execução.

REFERÊNCIAS

Bernardin, T., E. Cowgill, O. Kreylos, C. Bowles, P. Gold, B. Hamann, and L. Kellogg. **Crusta: A new virtual globe for real-time visualization of sub-meter digital topography at planetary scales**. In: *Computers and Geosciences* 37.1, pp. 75–85. 2011.

Bhattacharjee, S. **Real-time Terrain Rendering and Processing**. PhD thesis. 2010.

Bruneton, E. and F. Neyret. **Precomputed atmospheric scattering**. In: *Computer Graphics Forum* 27.4, pp. 1079–1086. DOI: 10.1111/j.1467-8659.2008.01245.x. 2008.

Cantlay, I. **Directx 11 terrain tessellation**. In: *Nvidia whitepaper* January. 2011.

Cignoni, P., F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. **BDAM - Batched Dynamic Adaptive Meshes for High Performance Terrain Visualization**. In: *Computer Graphics Forum* 22.3, pp. 505–514. DOI: 10.1111/1467-8659.00698. 2003.

Cignoni, P., F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. **Planet-Sized Batched Dynamic Adaptive Meshes (P-BDAM)**. In: *Proceedings of the IEEE Visualization Conference*, pp. 147–154. DOI: 10.1109/VISUAL.2003.1250366. 2003.

Clasen, M. and H.-C. Hege. **Terrain Rendering using Spherical Clipmaps**. In: *EUROVIS'06 Proceedings of the Eighth Joint Eurographics / IEEE VGTC conference on Visualization*, pp. 91–89. DOI: 10.2312/VisSym/EuroVis06/091-098. 2006.

Coumans, E. **Open Source Software: Bullet Physics Library**. URL: http://bulletphysics.org. 2018.

Cozzi, P. and K. Ring. **3D Engine Design for Virtual Globes**. 2011.

Deane, I. **Bullet Physics For Unity**. In: URL: https://assetstore.unity.com/packages/tools/physics/bullet-physics-for-unity-62991. 2018.

DeBoer, W. H. **Fast Terrain Rendering Using Geometrical Mipmapping**. 2000.

Dick, C., J. Krüger, and R. Westermann. **GPU Ray-Casting for Scalable Terrain Rendering**. In: *Eurographics*, pp. 43–50. 2009.

Dick, C., J. Krüger, and R. Westermann. **GPU-Aware Hybrid Terrain Rendering**. In: *Proceedings of IADIS Computer Graphics, Visualization, Computer Vision and Image Processing 2010* 3, pp. 3–10. 2010.

Dimitrijevic, A. M., M. Lambers, and D. D. Ranc. **Comparison of Spherical Cube Map Projections Used in Planet-Sized Terrain Rendering**. In: *Facta Universitatis, Series: Mathematics and Informatics* 31.2, pp. 259–297. 2016.

Dimitrijević, A. M. and D. D. Rančić. **Ellipsoidal Clipmaps - A planet-sized terrain rendering algorithm**. In: *Computers and Graphics (Pergamon)* 52, pp. 43–61. DOI: 10.1016/j.cag.2015.06.006. 2015.

Duchaineau, M., M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. **ROAMing terrain: real-time optimally adapting meshes**. In: *Proceedings of the 8th Conference on Visualization'97*. IEEE Computer Society Press, pp. 81–88. 1997.

Dupuy, J., J. Iehl, and P. Poulin. **Quadtrees on the GPU**. In: *GPU Pro 5: Advanced Rendering Techniques*, pp. 439–449. 2014.

Ebert, D. S., F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley. **Texturing and Modeling: A Procedural Approach**. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 2002.

Fernandes, A. R. and O. Bruno. **GPU Tessellation: We Still Have a LOD of Terrain to Cover**. In: *OpenGL Insights*, pp. 145–161. 2012.

Frasson, A., T. A. Engel, and C. T. Pozzer. **Improving Terrain Visualization Through Procedural Generation and Hardware Tessellation**. In: *Proceedings of XV SBGames*, pp. 218–221. 2016.

Garland, M. and P. Heckbert. **Fast polygonal approximation of terrains and height fields**. In: *Cmu-Cs-95-181*, pp. 95–181. 1995.

Gottschalk, S. **Separating axis theorem**. Tech. rep. Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill. 1996.

Hill, D. **An Efficient, Hardware-Accelerated, Level-of-Detail Rendering Technique for Large Terrains**. In: *Science*. 2002.

Hinsinger, D., F. Neyret, and M.-P. M. Cani. **Interactive animation of ocean waves**. In: ... *symposium on Computer animation* 46.22, pp. 5471–5481. DOI: 10.1145/545287.545288. 2002.

Hitchner, L. E. **Virtual Planetary Exploration : A Very Large Virtual Environment**. In: *IEEE Visualization*, pp. 148–163. 1992.

Kooima, R. **Planetary-scale Terrain Composition**. PhD thesis. 2008.

Lindstrom, P., D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. **Real-Time, Continuous Level of Detail Rendering of Height Fields**. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 109–118. 1996.

Lindstrom, P., D. Koller, W. Ribarsky, L. Hodges, A. Bosh, and N. Faust. **An integrated global GIS and visual simulation system**. Tech. rep. Georgia Institute of Technology. 1997.

Livny, Y., N. Sokolovsky, T. Grinshpoun, and J. El-Sana. **A GPU persistent grid mapping for terrain rendering**. In: *Visual Computer* 24.2, pp. 139–153. DOI: 10.1007/s00371-007-0180-1. 2008.

Löffler, F., H. Schumann, and S. Rybacki. **Error-bounded GPU-supported terrain visualisation**. In: *WSCG proceedings 2009*, pp. 47–54. 2009.

Losasso, F. and H. Hoppe. **Geometry clipmaps: terrain rendering using nested regular grids**. In: *ACM Transaction on Graphics* 1.212, pp. 769–776. DOI: 10.1145/1186562.1015799. 2004.

Mahsman, J. D. **Projective Grid Mapping for Planetary Terrain**. PhD thesis. 2010.

Mistal, B. **Gpu terrain subdivision and tesselation**. In: *GPU Pro 4: Advanced Rendering Techniques*, pp. 3–20. 2013.

Pajarola, R. and E. Gobbetti. **Survey on Semi-Regular Multiresolution Models for Interactive Terrain Rendering**. In: *The Visual Computer* 23.8, pp. 583–605. 2007.

Reddy, M., Y. Leclerc, L. Iverson, and N. Bletter. **TerraVision II: Visualizing massive terrain databases in VRML**. In: *IEEE Computer Graphics and Applications* 19.2, pp. 30–38. DOI: 10.1109/38.749120. 1999.

Strugar, F. **Continuous Distance-Dependent Level of Detail for Rendering Heightmaps**. In: *Journal of Graphics, GPU, and Game Tools* 14.4, pp. 57–74. DOI: 10.1080/2151237X.2009.10129287. 2009.

Tevs, A., I. Ihrke, and H.-P. Seidel. **Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering**. In: *Symposium on Interactive 3D Graphics and Games*, p. 183. DOI: 10.1145/1342250.1342279. 2008.

Thorne, C. **Error minimising pipeline for hi-fidelity, scalable geospatial simulation**. In: *2006 International Conference on Cyberworlds, CW'06*, pp. 81–88. DOI: 10.1109/CW.2006.22. 2006.

Ulrich, T. **Rendering Massive Terrains using Chunked Level of Detail Control**. In: *Proceedings ACM SIGGRAPH 2002* 34. 2002.

Westerteiger, R., A. Gerndt, and B. Hamann. **Spherical Terrain Rendering using the hierarchical HEALPix grid.** In: *Vluds* i, pp. 13–23. DOI: 10.4230/OASIcs.VLUDS.2011.13. 2012.