2

Trabalhos Relacionados

Apesar da grande evolução dos componentes computacionais, como *CPU*, memória e principalmente *GPU*, muito tem sido pesquisado em multi-resolução de terrenos e gerenciamento de terrenos em memória secundária. Isto ocorre porque geralmente é muito grande a quantidade de dados em uma aplicação deste tipo. Além disso, muitos dos trabalhos resolvem apenas uma parte do problema, ou seja, apenas o que interessa a uma aplicação específica. Por exemplo, a maioria dos trabalhos utiliza mapas de alturas para representar terrenos, possuindo, portanto, a limitação de não visualizar terrenos com cavernas. Outros limitam-se a apresentar um algoritmo eficiente de visualização, sem se preocupar se os dados vão ou não caber na memória principal. Existe ainda o problema da aplicação de texturas grandes, que pode ser considerada outra área importante.

Serão apresentados a seguir os trabalhos relacionados mais importantes e recentes que lidam com visualização eficiente de terrenos e acesso aos dados em memória secundária. Para manter o presente estudo conciso e direto, não serão apresentados os conceitos básicos necessários para a total compreensão deste texto. Uma descrição detalhada desses conceitos pode ser encontrada na literatura da área (por exemplo, [12, 26, 19, 22, 13]).

Com uma breve pesquisa, pode-se perceber que os artigos mais reconhecidos e citados em multi-resolução de terrenos baseiam-se em terrenos representados por hierarquias de triângulos retos conhecidas como *árvore binária de triângulos* [16, 11], *triangulação restrita de quadtree* [21], ou ainda *biseção da aresta mais longa* [17, 18]. A idéia principal compartilhada por estes métodos é a de construir uma hierarquia de multi-resolução regular por refinamento ou simplificação. No processo de refinamento um triângulo reto é dividido ou não em duas partes iguais com base no erro do vértice localizado no meio de sua aresta mais longa. Este processo permite cálculos simples de indexação, procedimentos recursivos fáceis e geração natural de malhas sem buracos ou vértices T.

O processo de simplificação é feito de forma inversa. Uma visão geral

sobre os métodos baseados neste esquema de triangulação pode ser encontrada no trabalho de Pajarola [22]. Este método foi primeiramente introduzido por Lindstrom *et al.* [16], cujo terreno é composto de ladrilhos e o processo é baseado em simplificação. Inicialmente, ladrilhos são selecionados no sentido de baixo para cima com base no erro projetado na tela. Posteriormente, cada ladrilho é analisado individualmente. Os vértices que não violam a tolerância de projeção de erro especificada são removidos. Isto é feito utilizando o procedimento de biseção da aresta mais longa. É necessário garantir que não haja vértices T ou falhas entre vértices dentro de um ladrilho, nem entre as bordas dos ladrilhos. Pode-se perceber que é um algoritmo custoso, tanto em processamento quanto em armazenamento. Röttger *et al.* [24] modificaram o algoritmo de Lindstrom *et al.* [16] para uma estratégia de refinamento, além de acrescentar transições suaves na geometria da malha (*geomorphing*), evitando assim o aparecimento repentino de vértices, e propor uma métrica de erro na qual a *quadtree* está sempre balanceada (ou seja, exista diferença de no máximo 1 nível entre nós vizinhos).

Posteriormente, Duchaineau *et al.* [11] propuseram um algoritmo chamado *ROAM*. Apesar de ser apresentado de forma diferente que por Lindstrom *et al.* [16], os algoritmos são similares. A principal diferença entre ambos é a utilização de coerência entre quadros no caso do *ROAM*. Duas filas de prioridade, uma de divisão (*split*) e outra de junção (*merge*), são usadas para atualizar a malha final do terreno a cada quadro. Ambas as filas são ordenadas por prioridade, cuja métrica pode ser ajustada da forma desejada, sendo a mais comum o erro projetado na tela. Como a atualização da malha é incremental, pode-se interromper a execução do algoritmo a qualquer momento, o que permite garantir uma taxa estável de quadros por segundo. Para que não haja falhas ou vértices T na malha, é necessário visitar recursivamente os vizinhos do triângulo em questão e subdividi-los caso seja necessário. Esta operação é muito cara e somente se justifica quando poucos triângulos precisam ser atualizados a cada quadro. Segundo Andrés [3], isto acontece em terrenos de baixa resolução, cuja árvore de dependência não é muito profunda. Em terrenos de alta resolução, devido à profundidade da árvore, ocorrem muitas mudanças. Além disso, em altas velocidades de navegação da câmera (velocidade relativa ao terreno), a malha final torna-se tão diferente que não justifica atualizar a malha de forma tão cara.

Toledo [26] propõe uma estrutura de dados denominada *QuadLOD*, que une as características de uma estrutura hierárquica em *quadtree* e

de malhas progressivas. Dessa forma, é possível aproveitar a capacidade de adaptação à superfície das malhas progressivas com a escalabilidade e simplicidade de uma estrutura como a *quadtree*. Primeiramente, um processo de simplificação é aplicado à *quadtree*, utilizando como métrica de erro a distância projetada na tela do observador até o ladrilho. Posteriormente, cada bloco é individualmente refinado ou simplificado como uma malha progressiva, respeitando uma restrição de fronteira entre blocos vizinhos. Os principais problemas deste trabalho são o alto consumo de *CPU* para a multi-resolução dentro de cada bloco e a qualidade dos triângulos dos blocos de níveis superiores (próximos da raiz), pois a restrição de fronteira acarreta a formação de triângulos muito finos.

Lindstrom *et al.* [17, 18] também utilizam o método de biseção da aresta mais longa, descrevendo uma forma de cálculo de erro projetado na tela de forma monotônica, ou seja, garantindo facilmente que a malha seja sempre contínua. Adicionalmente é proposta uma forma de construção de um *strip* único para todo o terreno. Tal malha é recalculada totalmente a cada quadro, sem a utilização de nenhuma informação de quadros anteriores. Além disso, o trabalho introduz a reorganização dos dados para que a coerência espacial de acesso aos vértices seja aproveitada. Por fim, é apresentada uma maneira simples de execução de *geomorphing*. András [3] mostra uma forma ainda mais eficiente de construir o *strip* de triângulos proposto por Lindstrom. Souza [12] faz um estudo profundo desse trabalho de Lindstrom *et al.* Recentemente, Bao *et al.* [6] apresentaram uma proposta que dá continuidade ao trabalho de Lindstrom *et al.* [18], acrescentando uma estrutura de dados que mantém uma distância de guarda para vértices recentemente utilizados, evitando acessos a dados em disco e cálculos de erro desnecessários. Os vértices são armazenados em um *buffer* em memória de vídeo, evitando transferências repetidas de vértices para a *GPU*.

Trabalhos mais recentes utilizam a idéia de agregar um conjunto de triângulos formando uma sub-malha, sendo cada sub-malha a unidade mínima de cálculo de multi-resolução. Dessa forma, pretende-se aliviar o gargalo entre *CPU* e *GPU*, aproveitando o processamento paralelo entre estes processadores e reduzindo o número de chamadas de desenho, verificação de erro e cálculo de visibilidade. Além disso, pode-se facilmente perceber que, com o poder computacional das *GPUs* de hoje em dia, é menos custoso enviar alguns triângulos desnecessários a mais do que gastar muitos ciclos de *CPU* para eliminá-los. Por fim, o armazenamento das sub-malhas em memória de vídeo permite utilizar toda a capacidade da aceleração das *GPUs* mais modernas. A principal desvantagem de métodos deste tipo é,

além do já mencionado uso de mais triângulos do que o necessário, o aumento da complexidade dos algoritmos, pois garantir a ausência de falhas e vértices T em superfícies deste tipo exige um esforço maior.

Um dos primeiros a propor algo deste tipo foi De Boer [8]. Inicialmente, apresenta-se uma estrutura de *quadtree* contendo caixas envolventes para a eliminação de nós fora do volume de visão. As folhas da *quadtree* contêm um ladrilho ou bloco do terreno. Dentro de cada ladrilho é utilizada uma espécie de *mipmapping* geométrico, chamado por De Boer de *Geomipmapping*. Isto significa que cada ladrilho é reamostrado em resoluções menores, criando diversos níveis de *geomipmapping*. Adicionalmente é necessário um tratamento especial para resolver falhas entre ladrilhos vizinhos que estão em níveis diferentes. Por fim, pode ser feito *geomorphing* entre níveis de *geomipmapping* consecutivos. Apesar de ser uma abordagem simples, não pode ser considerada escalável, pois o máximo de simplificação conseguido é o número de ladrilhos nas folhas da *quadtree*. Além disso, a forma que De Boer propõe para resolver a vizinhança entre níveis diferentes apresenta problemas no formato dos triângulos, ou seja, gera triângulos muito finos.

Pomeranz [2] apresentou uma estrutura de dados chamada *RUSTIC*. *RUSTIC* é um aprimoramento de *ROAM* [11] em que sub-árvores da árvore binária de *ROAM* são, em tempo de pré-processamento, estaticamente criadas e salvas. Levenberg [14] apresenta também outra estrutura de dados chamada *CABTT*, similar à *RUSTIC*, cuja diferença principal está na construção dinâmica, com *cache* e reuso dessas sub-árvores.

A maioria dos trabalhos citados acima não trata os casos de terrenos que não cabem em memória e, quando o fazem, não o fazem de maneira satisfatória. Lindstrom *et al.* [18] propõem utilizar rotinas do sistema operacional para fazer o mapeamento de um arquivo em disco para a memória *RAM*. Embora seja uma solução simples e funcional, possui a limitação de não poder mapear, segundo Cignoni *et al.* [9], arquivos maiores que *2GBytes* (em máquinas 32 bits), sendo portanto uma solução não escalável. Como o algoritmo de Lindstrom possui muitos dados adicionais por vértice, um terreno grande que não exceda este limite dificilmente pode ser criado.

Bao *et al.* [5] propõem uma forma de armazenamento dos dados que minimiza o acesso aos dados em memória secundária. A forma de organização dos dados leva em consideração, além da proximidade espacial, informações de nível de detalhe (*LOD*). Dessa forma, as informações utilizadas para o cálculo de multi-resolução dos vértices do terreno refletem no

agrupamento dos mesmos. O lado negativo é que é necessária uma etapa de pré-processamento mais complicada do que o habitual, além de uma implementação complexa do sistema de gerenciamento de páginas.

Machado [19] apresenta um trabalho cujo foco principal é uma arquitetura cliente-servidor para a visualização de terrenos que utiliza uma *quadtree* como estrutura de multi-resolução. O servidor é responsável por carregar os ladrilhos requisitados pelo cliente. Além disso, o servidor utiliza o sistema operacional para gerenciar o acesso aos dados em disco. O cliente é responsável pela comunicação com o servidor e pela visualização e gerenciamento dos ladrilhos. A comunicação consiste basicamente na requisição e no recebimento de ladrilhos, fazendo reordenação e *timeout*. Na visualização, a eliminação de ladrilhos fora do volume de visão é feita de forma não convencional, projetando a caixa envolvente do ladrilho na janela de visualização. A métrica de erro é executada de forma semelhante, ou seja, não leva em consideração a informação local do ladrilho. Nesse trabalho também é possível obter uma taxa em quadros por segundo adaptada à necessidade da aplicação.

Em dois trabalhos mais recentes, Cignoni *et al.* [9, 10] apresentam uma técnica de visualização eficiente de terrenos em memória secundária denominada *BDAM*. Esta técnica também é baseada em uma árvore binária de triângulos, mas cada triângulo representa uma sub-malha. As sub-malhas são apresentadas na forma de malhas irregulares ou *TINs*, que são construídas e otimizadas em uma fase de pré-processamento. Uma métrica de erro complexa é apresentada para garantir a ausência de falhas e vértices T. O acesso aos dados em disco, assim como no trabalho de Lindstrom *et al.* [18], utiliza rotinas do sistema operacional. Um esforço extra é necessário para vencer a barreira de *2GBytes* mencionada anteriormente, de forma que somente o(s) segmento(s) necessário do arquivo é(são) mapeado(s) em memória. Além disso, é apresentada uma solução, semelhante à da geometria mas em forma de uma *quadtree*, para a aplicação de textura. Ainda assim, apesar da comprovada eficiência destes trabalhos, o principal problema, além de um custoso tempo de pré-processamento, é a complexidade de suas implementações.

Outro foco de interesse deste trabalho é implementar um mecanismo de predição para realizar o carregamento prévio dos dados para a memória principal. Embora existam poucos trabalhos que citem tal assunto especificamente em visualização de terrenos, o uso deste tipo de técnica é imprescindível quando temos acesso aos dados em memória secundária. O benefício principal de se utilizar um mecanismo de predição é sempre ter os dados

necessários disponíveis imediatamente. Isso envolve, no caso de terrenos, uma predição da posição e direção de visualização do observador. Cignoni *et al.* [10] fazem o refinamento da árvore de maneira semelhante ao procedimento de visualização, sendo que ao final as sub-malhas necessárias não são desenhadas. Como utilizam um mapeamento virtual de arquivo em memória, essas sub-malhas são usadas para avisar ao sistema operacional quais páginas de memória serão necessárias em um futuro próximo. Pelo que se sabe, isso só é possível fazer atualmente em *Linux*.

Machado [19] apresenta uma predição de posição utilizando uma fila de prioridades para carregar ladrilhos que tenham maior erro de projeção na tela. Posteriormente, Machado mostra o ganho adquirido com a técnica, tanto em termos de baixa variação da taxa em quadros por segundo, quanto na taxa de acertos em *cache* dos ladrilhos utilizados. Deve-se destacar também que em uma arquitetura cliente-servidor, como a apresentada por Machado, fica mais evidente o ganho apresentado com a predição.

Analisando os trabalhos acima pode-se fazer uma breve comparação entre os métodos que utilizam e os que não utilizam agregação de triângulos, destacando as suas principais vantagens e desvantagens.

Os trabalhos que não utilizam agregação de triângulos ou sub-malhas geralmente apresentam como vantagens a simplicidade de implementação e a geração de malhas com a quantidade de triângulos próxima da ótima. Além disso, a incorporação de *geomorphing* geralmente é simples. As principais desvantagens que pode-se destacar são o grande uso de memória (informações adicionais por vértice ou triângulo para o cálculo de nível de detalhe) e o uso intenso de *CPU* (nível de detalhe e visibilidade calculados por vértice). Além disso, a gerência de dados em memória secundária por vértice ou triângulo pode-se tornar impraticável, dificultando a escalabilidade.

Os trabalhos que agregam triângulos para formar sub-malhas são uma tendência atual. Eles apresentam como principal vantagem uma visualização eficiente, explorando ao máximo os recursos da GPU. Esta característica é importante porque, como foi dito anteriormente, a capacidade de processamento das *GPUs* tem evoluído mais rapidamente do que a das *CPUs*. Além disso, o uso de *CPU* e memória é menor devido à redução dos cálculos e das informações de nível de detalhe e visibilidade. Entretanto, a maioria desses trabalhos apresenta uma implementação demasiadamente complexa, principalmente porque garantir a ausência de falhas e vértices T torna-se mais difícil. Além disso, esses métodos utilizam uma quantidade maior de triângulos do que o realmente necessário para representar o terreno.

A solução proposta neste trabalho busca obter uma implementação simples, eficiente e escalável. Para isso, a *quadtree* foi a estrutura de multi-resolução escolhida. Trata-se de uma estrutura de implementação simples, na qual a eliminação de falhas pode ser feita também de forma simplificada. Além disso, a adaptação de texturas em uma região quadrada fica facilitada. Cada nó da *quadtree* é um ladrilho que contém uma sub-malha da região abrangida pelo nó. Nos próximos dois capítulos serão apresentados, respectivamente, a estrutura de dados utilizada e o algoritmo de visualização sobre esta estrutura, levando em consideração a necessidade de acesso a dados em memória secundária.