Análise de *Walkthroughs* em Cenários 3D de um jogo utilizando Algoritmos de Visibilidade e Estruturas de Particionamento Espacial [*]

Yvens R. Serpa[1], Maria Andréia F. Rodrigues[2]

[1] Centro de Ciências Tecnológicas – Universidade de Fortaleza (UNIFOR)
Caixa Postal 1258– Fortaleza – CE – Brasil

[2] Programa de Pós-Graduação em Informática Aplicada – Universidade de Fortaleza (UNIFOR)
Caixa Postal1 1258– Fortaleza – CE – Brasil

{yvensre, andreia.formico}@gmail.com

Abstract. *In this work, we present a detailed analysis of 3D scene walkthroughs of a digital game, using culling algorithms (View Frustum Culling and Backface Culling) and spatial partitioning structures (Octree and BSP-Tree). Basically, we focus our analysis on the following main aspects: total and per frame processing time spent, effective reduction in the number of triangles sent to the rendering pipeline and memory consumed.*

Resumo. *Neste trabalho, apresentamos uma análise detalhada de walkthroughs em cenários 3D de um jogo digital, usando algoritmos de culling (View Frustum Culling e Backface Culling) e estruturas de particionamento espacial (Octree e BSP-Tree). Basicamente, focamos nossa análise nos seguintes aspectos principais: tempo de processamento total e por quadro gasto, redução eficaz de triângulos enviados ao pipeline de renderização e memória consumida.*

1. Introdução

O interesse cada vez mais crescente dos usuários de aplicações gráficas por *walkthroughs* em ambientes 3D cada vez mais realistas e em tempo real, como acontece nos jogos digitais, tem motivado que novas abordagens seja propostas para reduzir eficientemente o número de triângulos enviados ao *pipeline* de renderização, o tempo de processamento e a memória consumida nessas aplicações. Diversos são os fatores que aumentam a sobrecarga de processamento dos ambientes (modelos de tonalização, texturas, comportamento dinâmico dos objetos, etc.), contribuindo para diminuir o nível de interatividade do usuário e, consequentemente, piorando a experiência do jogo.

---

[*] Versão estendida do trabalho apresentado no WUW/SIBGRAPI 2013 [16], recomendado para publicação na REIC.

Além disso, o desenvolvimento de métodos otimizados para a determinação do conjunto de triângulos potencialmente visíveis (PVS) pela câmera sintética durante *walkthroughs* em ambientes 3D sempre foi uma tarefa pouco trivial e sem solução ótima [1, 2, 3]. Dessa forma, diferentes abordagens para tratar essa questão têm sido constantemente avaliadas, incluindo técnicas como estruturas de particionamento espacial (EDEs) [1, 3, 4] e algoritmos de *Visibility Culling* [5, 6].

Mais especificamente, as EDEs dividem o ambiente em sub-regiões para reduzir o processamento das porções da cena efetivamente visíveis, num dado momento pelo observador. As partições podem ser alinhadas aos eixos cartesianos (ou não) e serem regulares ou irregulares. EDEs fornecem meios mais eficientes para atualização dos dados e recuperação dos mesmos, através de mecanismos eficazes de busca, remoção e inserção [8]. Esse aspecto é estratégico já que em *walkthroughs*, a quantidade de informação espacial (geralmente na forma de vértices e triângulos ao longo da animação) pode crescer exponencialmente, tornando evidente a exploração de métodos e estruturas eficientes para a sua manipulação. Em relação ao armazenamento, as EDEs também devem evitar redundância de elementos, já que esta quantidade é potencialmente grande.

Alguns exemplos comuns de EDEs são as *Octrees* e *BSP-Trees* [3]. A *Octree* representa malhas geométricas 3D de forma hierárquica, realizando subdivisões no espaço, de forma recursiva. A cada nível da hierarquia, o volume é particionado em 8 subvolumes cúbicos idênticos e alinhados aos eixos.

A natureza inflexível das *Octrees* motivou a criação de estruturas de dados mais adaptáveis ao contexto das aplicações [10], como a *BSP-Tree,* a qual realiza a divisão consecutiva do espaço em duas subregiões, através de hiperplanos de corte, não necessariamente, alinhados aos eixos. As vantagens do uso das *BSP-Trees* incluem uma pesquisa espacial eficiente (através de simples comparações envolvendo o lado do hiperplano de corte) e uma maior flexibilidade. Contudo, um número maior de informações relativas aos hiperplanos precisa ser armazenado em cada nó. Assim, uma escolha mal planejada destes pode gerar uma árvore bastante profunda e/ou desbalanceada.

Dentre os algoritmos de visibilidade mais conhecidos destacam-se: o *view-frustum culling* (VFC) e o *backface culling* (BFC) [13]. Em termos gerais, o VFC é usado para descartar os polígonos que não estão contidos no volume de visualização; e o BFC, para identificar e descartar as faces dos objetos que não estão voltadas para o observador. Em particular, o algoritmo de BFC implementado neste trabalho é baseado no trabalho proposto em [15].

De forma geral, as técnicas de particionamento são extensíveis, podendo ser combinadas a algoritmos de *Visibility Culling*, tornando-se atraente para abordagens híbridas [3, 14].

Similarmente às idéias descritas em [10], neste trabalho serão comparados o comportamento de duas das principais EDEs existentes (*Octree* e *BSP-Tree)* e dos algoritmos de visibilidade VFC e BFC para renderização efetiva em *walkthroughs* por ambientes 3D do jogo de computador *Team Fortress* [9], focando nos seguintes aspectos: redução eficaz do número de triângulos enviados ao *pipeline* de renderização,

tempo de processamento gasto e memória consumida. O jogo *Team Fortress* oferece um amplo e diversificado espectro de visões de câmera geradas ao longo das trajetórias, portanto, sendo um interessante estudo de caso a ser explorado.

2. Trabalhos Relacionados

Samet e Webber apresentam um trabalho clássico sobre os fundamentos e aplicações do uso de estruturas de dados hierárquicas e seus respectivos algoritmos [4]. Em outro trabalho, os autores elaboram uma visão geral sobre o uso destas estruturas, organizando seus respectivos dados, relativamente às regiões de ocupação no espaço [8].

Recentemente, Andrysco *et al.* investigam, a partir de EDEs mais comuns, como *Octrees e KD-Trees*, a construção de uma estrutura mais versátil aplicada a uma gama de problemas, assumindo inicialmente que nenhuma EDE é capaz de oferecer uma solução ótima para todos os cenários de renderização [10].

A idéia de PVSs é introduzida por Airey *et al.* em [12], sendo revisitada em outros trabalhos, os quais descrevem como obter estes conjuntos de forma mais eficiente, ao usar algoritmos de VFC [2, 5].

Cohen-Or *et al.* revisam detalhadamente os conceitos fundamentais sobre visibilidade e técnicas de *visibility culling* [1]. Essas técnicas são aplicadas por Bittner *et al.* em conjunto a EDEs, na tentativa de melhor determinar a porção efetivamente visualizada pelo observador no processo de *culling* [6].

A principal fonte de inspiração para o algoritmo de BFC desta pesquisa foi o trabalho de Zhang *et al*. [15], o qual apresenta a ideia de máscaras de *bit* para categorizar as normais das porções da malhas geométricas do ambiente, calculadas durante uma fase de pré-processamento. Os autores propõe, em tempo de execução, categorizar a normal da câmera sintética e utilizá-la para, através da comparação com as máscaras pré-definidas das malhas, definir quais porções estão (ou não) voltadas ao observador, a fim de removê-las.

3. Cenários de Testes

Vários testes de *walkthroughs* pelos ambientes do *Team Fortress* foram realizados em um *notebook* com processador Intel Core i5-2450m 2.5 GHz, 4 GB DDR3 SDRAM, Placa Gráfica Intel(R) HD Graphics 3000. Nosso principal interesse foi o estudo comparativo entre o número de triângulos submetidos ao *pipeline* de renderização, uso de memória e tempo dispendido no processo de remoção dos mesmos; bem como a avaliação dos principais fatores que interferem no desempenho da aplicação durante o *walkthrough*.

3.1 Ambientes 3D

Sete ambientes 3D diferentes foram preparados para os testes, a saber: *Nucleus, Coal Town, Ravine, Gorge, Viaduct, Saw* e *Gold Rush* (Figura 1 e Tabela 1). Os critérios para a seleção desses ambientes desse jogo foram os seguintes: (1) complexidade geométrica e tamanho em MB variados (incluindo ambientes mais simples e menores, contendo cerca de 60 mil vértices e 1.7MB, a ambientes maiores e mais complexos, com aproximadamente 400 mil vértices e 11MB); (2) *walkthroughs* mistos (alternando

trechos internos e externos, com aclives, declives, escadas, curvas acentuadas, janelas, etc.); (3) popularidade elevada do *Team Fortress* (em torno de 50 mil usuários simultâneos por dia [11]); e por ser (4) gratuito e multiplataforma.

(a) Nucleus (b) Coal Town (c) Ravine (d) Viaduct (e) Gorge (f) Saw (g) Gold Rush

Figura 1 *Walkthroughs* por ambientes 3D do jogo *Team Fortress*

Nucleus	Coal Town	Ravine	Viaduct	Gorge	Saw	Gold Rush
59.889 vértices	77.682 vértices	100.266 vértices	169.800 vértices	166.542 Vertices	181.596 vértices	394.794 vértices
1.7 MB	2.1 MB	2.9 MB	5.2 MB	5.2 MB	5.3 MB	11 MB

Tabela 1: Nome, número de vértices e tamanho dos ambientes 3D do jogo

3.2 Metodologia

Os testes realizados incluíram duas EDEs (Octree nível 4 e BSP-Tree com fator mínimo 200, isto é, seus nós folhas tem, no mínimo, 200 triângulos), combinadas a dois algoritmos de visibilidade (VFC e BFC). As duas estruturas de dados foram desenvolvidas para melhor adequar-se às ideias propostas em [15]. Desta forma, os nós folhas de ambas as EDEs possuem máscaras de *bits* representando as diferentes categorias de normais presentes em um nó. A classificação das normais é um processo simples que envolve descobrir o ângulo formado pela normal entre os eixos *x* e *y*, e seu respectivo octante, no círculo unitário. Para nossos testes, consideramos a classificação das normais, de acordo com a Figura 2.

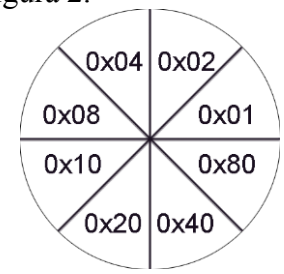

Figura 2 Máscaras de *bit* para a classificação das normais

Para ambas as estruturas, as combinações realizadas foram BFC, VFC e BFC+VFC. Inicialmente propomos mais um algoritmo de *culling*, o *Occlusion Culling*, no entanto, nos testes iniciais esta abordagem não se mostrou competitiva, além de ser inadequada para ser combinada à *BSP-Tree*, isto porque os hiperplanos de corte, durante a geração da BSP-Tree, podem provocar o corte de potenciais oclusores, além de potencialmente aumentar o número de polígonos a serem testados, devido ao particionamento dos mesmos, degradando o desempenho do algoritmo como um todo.

Definimos e gravamos sete diferentes *walkthroughs*, cada um formado por 2.000 quadros (cada quadro guardando sua posição e orientação), um para cada ambiente (Figura 1), com o objetivo de garantir o controle dos experimentos e facilitar a reprodução dos testes. Comparamos o tempo de processamento dos algoritmos de visibilidade e EDEs, bem como o número de triângulos enviados ao *pipeline* de renderização e as influências no desempenho das estruturas. Cada teste foi executado uma única vez. Os dados numéricos coletados ao longo de cada *walkthrough* foram consolidados a cada 20 quadros, com os respectivos valores sendo usados para a geração das curvas correspondentes.

Mais especificamente, os arquivos originais dos ambientes 3D foram pré-processados pelos respectivos algoritmos de cada EDE, gerando um arquivo específico, dado de entrada no *framework* de testes utilizado. Estes arquivos gerados contêm os dados relativos ao ambiente 3D (lista de vértices e de conexões), informações a respeito da EDE específica e os valores paras as máscaras de normais. O *framework* de testes que usamos foi originalmente implementado por Moreira [7], disponibilizando a implementação da leitura dos arquivos (Figura 2).

Testes sistemáticos foram realizados contendo informações relevantes como: tempo gasto em *ms*; uso de memória RAM consumida; e número de triângulos enviados ao *pipeline* de renderização a cada quadro da trajetória, isto é, a cada passo da câmera. Entende-se por passo, qualquer movimento de translação ou rotação realizado pelo observador, não havendo a ocorrência duas transformações simultâneas durante um mesmo passo. Os ambientes contêm texturas simples, tendo o modelo de tonalização de *Gouraud* ativado.

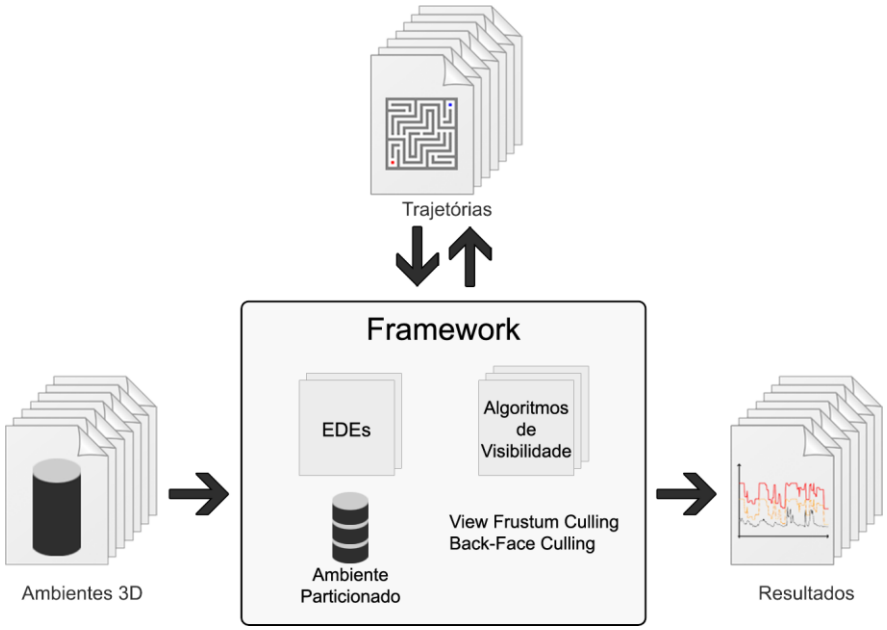

Figura 2. *Framework* de testes

4. Resultados e Discussão

Inicialmente, as abordagens que utilizaram somente o algoritmo de BFC tiveram um desempenho bastante inferior em comparação às abordagens que utilizavam o algoritmo

de VFC. Isto porque, na ordem de execução do algoritmo, o VFC elimina nós não visíveis pela câmera sintética e o processamento do BFC é limitado aos nós contidos no campo de visão do observador. Quando o VFC não está presente, o processamento do BFC é realizado para todos os nós da estrutura, degradando o desempenho. Além disso, como o algoritmo utilizado é baseado na proposta conservativa de Zhang [15], a remoção de triângulos é bastante sutil, de forma que muito triângulos ainda são enviados ao *pipeline* de renderização (Figura 3).

Em contrapartida, as abordagens que utilizavam o VFC mostraram desempenho competitivo e redução eficiente de triângulos. A combinação entre os dois algoritmos de *culling* mostrou os melhores resultados (em relação ao tempo de processamento) na maioria dos casos, empatando em alguns casos com a abordagem sem o algoritmo de BFC (Figura 3).

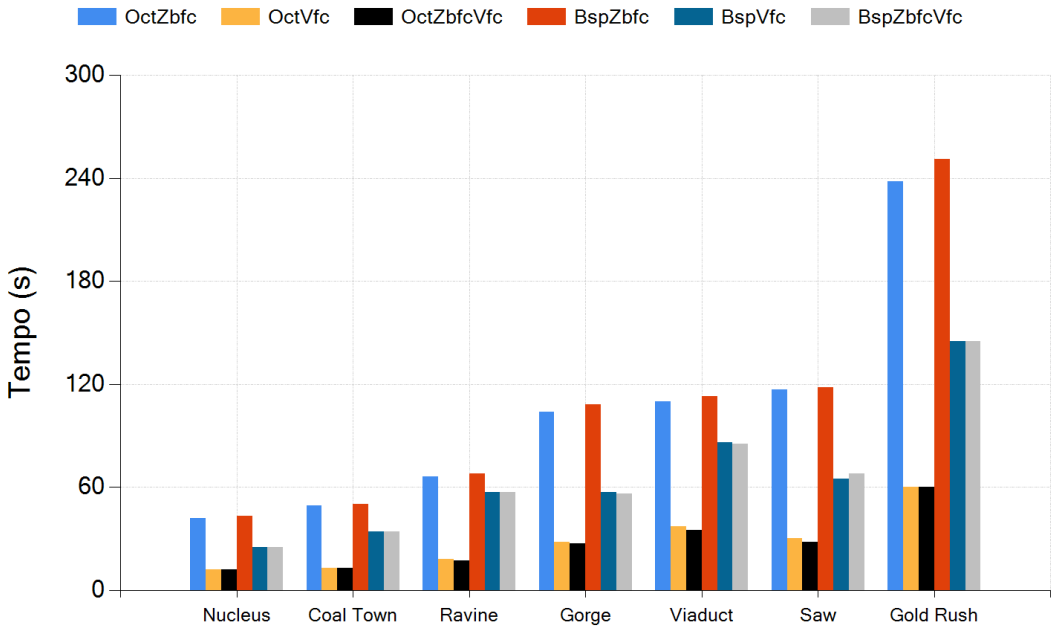

Figura 3. Tempo total de processamento para as combinações, em cada cenário 3D

As estruturas de dados testadas mostraram desempenho bastante diferentes entre si, no entanto, mantendo o mesmo comportamento diante da combinação com os algoritmos de *culling*.

A comparação mais simples entres as EDEs trata-se da combinação com o BFC, apresentando desempenho inferior, prejudicado pelo número de nós e custo de navegação pela estrutura da *BSP-Tree*. A *Octree* utilizada possui profundidade fixa (nível 4), possuindo o máximo de 4096 nós folhas a serem percorridos e processados pelo BFC. No entanto, a *BSP-Tree* não possui profundidade fixa, dependendo do valor de polígonos mínimos presentes em seus nós, isto é, um nó é definido como folha caso possua um número de triângulos inferior ou igual ao número mínimo estipulado para os testes (em nossa implementação, o valor foi 200). Assim, em cenários bastante complexos, a tendência é que a profundidade da estrutura seja potencialmente grande, tornando-se mais custosa em termos de espaço em memória e seu tratamento torna-se menos eficiente. Na Figura 3, mostra-se que há um aumento do tempo de processamento

para todas as abordagens, à medida que a complexidade do cenário aumenta, tornando-se mais evidente a diferença de tempo entre as abordagens *Octree*+BFC e *BSP-Tree*+BFC.

Quando combinadas ao algoritmo de VFC, ambas as EDEs tornam-se mais competitivas, com diminuições bastante consideráveis nos tempos totais de processamento. A *Octree*, em cenários simples, como o *Nucleus*, obtém resultados cerca de 3,5 vezes mais rápidos e, em cenários mais complexos, como o *Gold Rush*, aproximadamente 3,9 vezes mais rápidos em comparação à abordagem *Octree*+BFC. Mesmo em uma proporção inferior, a *BSP-Tree* atinge resultados (para os mesmos cenários mencionados anteriormente), cerca de 1,71 e 1,79 vezes mais rápidos, respectivamente.

Ao compararmos os tempos de processamento entre as EDEs, nota-se que em todos os casos a *Octree* apresenta resultados melhores do que aqueles apresentados pela outra estrutura. Por exemplo, no cenário *Ravine*, o melhor resultado é obtido pela combinação *Octree*+BFC+VFC, realizando o *walkthrough* em cerca de 17 segundos, cerca de 3,35 vezes menos tempo do que a mesma combinação utilizando a estrutura da *BSP-Tree*, demorando cerca de 57 segundos para realizar o mesmo trajeto.

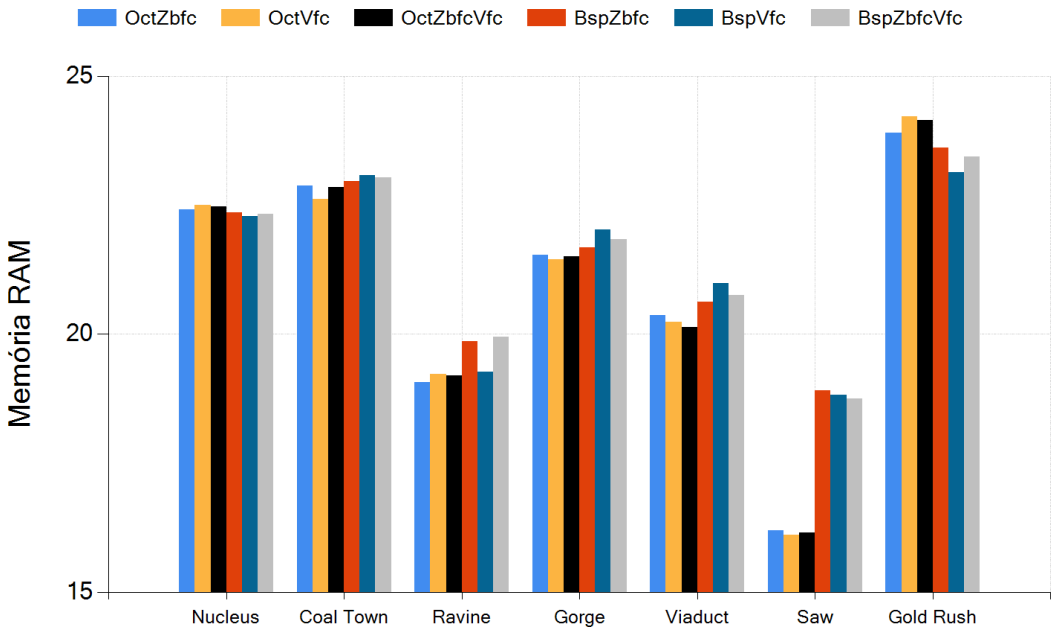

Figura 4. Memória RAM média (MB) utilizada por combinação, por mapa

Em termos de memória RAM consumida, todas as combinações mostraram-se bastante próximas, com variações pequenas entre 4 MB para mais ou para menos, de acordo com o cenário. Em alguns casos, como no cenário *Saw*, a diferença entre o custo em memória das abordagens *Octree* e *BSP-Tree* são mais evidentes devido à complexidade do cenário (por exemplo, malhas geométricas complexas formando componentes do cenário, como detalhes arquitetônicos das estruturas), provocando situações desafiadoras ao processo de geração de planos de cortes da estrutura.

Adicionalmente, percebe-se que não necessariamente o custo em memória dos algoritmos está diretamente relacionado à complexidade dos cenários, visto que o ambiente mais simples, *Nucleus*, apresentou pouca diferença em relação ao mais complexo, *Gold Rush*. A explicação para tal reside no *trade off* que existe entre complexidade da malha geométrica e a utilização de algoritmos e estruturas de particionamento. Em cenários simples, com poucas dificuldades e malhas geométricas bem distribuídas, é possível que a utilização de técnicas de visibilidade degrade o desempenho geral da aplicação. Por outro lado, em cenários mais complexos, a não utilização das mesmas tornará mais evidente o custo computacional de tais características. Desta forma, conforme há uma maior complexidade e sofisticação dos ambientes, a necessidade da utilização de técnicas de visibilidade torna-se fundamental, até um ponto em que o ambiente de entrada é tão complexo que o ganho, ainda significativo, não é o suficiente, de modo que há sempre a necessidade de técnicas ainda mais robustas para lidar com tais situações. Nota-se que os cenários mais ao centro da Figura 4 (*Ravine, Gorge, Viaduct,* Saw) possuem custo de memória e tempo de processamento mais baixos em relação aos cenários localizados nas extremidades, evidenciando o *trade off* mencionado.

No início do *walkthrough* pelo cenário *Nucleus*, a câmera está posicionada em uma plataforma central, em um amplo abismo conectado a uma ponte, em áreas de interseção entre regiões das EDEs. Tratando-se de um ambiente interno, o cenário apresenta dificuldade para a geração de planos de cortes da *BSP-Tree*, que, além de mostrar tempo de processamento superior às abordagens da outra EDE (Figuras 3 e 5), envia um número superior de triângulos ao *pipeline* de renderização, em todos os momentos, em comparação às combinações mais competitivas, como a *Octree+BFC* (Figura 12). A *Octree*, por ser indiferente às complexidades da malha do ambiente consegue resultados competitivos, embora, esta mesma característica regular da estrutura constrói nós folhas com uma grande variedade de categorias de normais, o que torna o processo de BFC ineficiente, visto que quanto maior o número de diferentes categorias de normais em um nó, maior a probabilidade de pelo menos um deles seja visível ao observador, dada a categoria da normal da câmera, de forma que todo o nó seja considerado visível.

O *Coal Town*, um ambiente simples, com casas regulares, se beneficia das regiões regulares e cúbicas da *Octree*, garantindo um excelente desempenho (Figura 6). Estas características levaram a geração de nós folhas com menor diversidade de categorias de normais o que trouxe para o algoritmo BFC uma redução, ainda pouco significativa, mas presente, para ambas as estruturas de particionamento (Figura 13). No entanto, o terreno acidentando e a presença de janelas são características que dificultam a geração de planos de corte da *BSP-Tree*, resultando em um desempenho inferior ao da *Octree*.

Embora contendo uma quantidade média de triângulos, *Ravine* é bem mais complexo do que os ambientes anteriormente apresentados. Inclui torres, potenciais pontos de complexidade para o *visibility culling*, pois permitem que o observador se posicione em regiões elevadas, com amplo campo de visão. Além disso, durante a geração da *Octree*, forçam que as *Axis-Aligned Bounding Box* (*AABBs*) de seus nós sejam maiores, decorrente da altura máxima elevada do ambiente. Consequentemente,

geram uma maior variedade de categorias de normais, prejudicando o funcionamento do BFC, que mesmo assim ainda consegue realizar, em alguns momentos, remoções de triângulos (Figura 14). Em todas as fases do *walkthrough*, em que o observador passa por áreas elevadas e/ou realiza rotações maiores do que 90º, todas as abordagens deterioram seu o desempenho devido à avaliação de diferentes regiões do ambiente (Figura 7).

No trajeto pelo *Viaduct* há duas situações de principal dificuldade. O terreno bastante irregular serve, ora como oclusor (quando à frente do observador, como uma grande rampa), ora como ampliador do campo de visão (quando o observador está em posições mais elevadas). Consequentemente, o processo de *visibility culling* torna-se custoso e, em situações de amplo campo de visão, faz com que muitos nós das estruturas de particionamento sejam processados, degradando o desempenho do sistema (Figura 8). A presença de elementos como escadas vazadas, cercas de diferentes tamanhos, suportes e plataformas mostrou-se uma dificuldade aos processo de geração dos planos de corte da *BSP-Tree* (Figura 15).

Constituído de três andares, *Gorge* tem uma construção complexa e fechada em seu centro, mas a navegação é simples, oferecendo dificuldades apenas nos momentos de transição entre os nós das estruturas (Figura 9). Em particular, em um trecho curto, as combinações com *BSP-Tree* atingem resultados superiores àquelas combinadas à *Octree*, dada a geometria bastante regular da região, em contraste com a grande quantidade de nós visíveis pela estrutura, forçando-a a percorrer uma grande variedade de nós, degradando o desempenho (Figura 9). No entanto, em termos gerais, o comportamento é similar aos testes anteriores, nos quais as abordagens utilizando a *Octree* se mostram mais velozes em termos de tempo de processamento e com redução mais eficiente no número de triângulos enviados ao *pipeline* de renderização (Figura 16).

O *walkthrough* pelo cenário *Saw* atravessa um casarão indo em direção a um túnel. Tem baixa complexidade, com campo de visão reduzido da câmera e às regiões relativamente homogêneas e planas (Figura 17). Há também outro túnel, com paredes extremamente irregulares, que conduz a um campo aberto, gerando dificuldades e curvas de baixo desempenho, no intervalo entre o quadro 400 e 900 da Figura 10.

O maior ambiente testado, em termos de espaço ocupado em disco e com a maior quantidade de triângulos, foi o *Gold Rush*. O *walkthrough* (Figura 11) leva a situações críticas em que o campo de visão contém uma diversidade de nós das EDEs (o observador situa-se próximo a janelas, entre os limites das construções e da região externa). Os testes de interseção contudo são bastante rápidos, garantindo o desempenho competitivo da *Octree,* mesmo em um ambiente grande e complexo. Devido ao tamanho do ambiente, os nós da *Octree* são potencialmente grandes, o que produz nós com alta variedade de categorias de normais, degradando o desempenho do processo de BFC (Figura 18).

5. Conclusões e Trabalhos Futuros

Podemos concluir que o estudo comparativo conduzido reforça o fato da importância em se identificar uma abordagem competitiva o bastante, envolvendo algoritmos de *visibility culling* e EDEs, capaz de, durante a execução real de um jogo digital (cujo

custo por triângulo enviado é maior), mostrar desempenho mais competitivo em relação ao não uso de tais abordagens. Durante a execução de algoritmos de visibilidade baseados em *Octrees* a eliminação dos nós não-visíveis resultou, na maioria dos casos, na eliminação de grandes regiões, tornando o processo geral mais eficiente. Além disso, sua estrutura mostrou boa sinergia com ambos algoritmos de *culling* testados, inclusive com o BFC inspirado pelo trabalho de Zhang [15]. Em contrapartida, embora a *BSP-Tree* seja uma estrutura mais flexível, não se mostrou eficiente diante da complexidade dos estudos de caso e, em diversas ocasiões, produziu nós-folhas com grande diversidade de categorias de normais, tornando o BFC ineficiente (diferente da *Octree* que, em muitos casos, possuía nós com poucas normais, realizando um processamento mais eficiente).

Um ponto de fragilidade encontrado nos testes foi a grande diversidade de categorias de normais presentes nos nós folhas das EDEs estudadas, o que torna ineficiente o processo de BFC, isto porque a probabilidade de que pelo menos uma máscara seja compatível com o campo de visão do observador é o suficiente para classificá-lo como visível e enviá-lo ao *pipeline*. Dessa forma, uma que envolva métodos probabilísticos e estatísticos aplicados ao grau de variância das normais na definição dos nós e seus respectivos tamanhos poderia levar à geração de uma EDE mais eficiente em relação ao tempo de processamento, memória e remoção de triângulos.

Como trabalhos futuros, um estudo mais detalhado sobre heurísticas para a escolha dos planos de corte da *BSP-Tree* poderia levar a uma melhor construção da árvore, conservando a sua velocidade, mas realizando o processo de remoção das áreas invisíveis de forma tão eficiente quanto a *Octree*. Ainda há também espaço para propor uma abordagem híbrida, que utilize o melhor dos dois métodos. O mesmo se aplica aos algoritmos de visibilidade. Por exemplo, um estudo mais detalhado a respeito das técnicas de BFC e a inclusão de novos algoritmos de *culling*, como o *Occlusion Culling*, poderiam também gerar abordagens mais robustas e flexíveis, diante de uma variedade de cenários 3D.

Referências

[1] Cohen-Or, Y. Chrysanthou, C. Silva, F. Durante. “A Survey in Visibility for Walk-through Applications”. In IEEE TVGCG Journal, v.9, n. 3, p. 412-431, 2003.

[2] M. Laakso. “Potentially Visible Set (PVS)”. Technical Report, Helsinki Univertiy of Technology Telecommunications Software and Multimedia Laboratory, 2003.

[3] W. Silva, M.A.F. Rodrigues. Interactive Rendering of Indoor and Urban Environments on Handheld Devices by Combining Visibility Algorithms with Spatial Data Structures. International Journal of Handheld Computing Research, v. 2, p. 55-71, 2011.

[4] H. Sammet, R.E. Webber. “Hierarchical Data Structures and Algorithms for Computer Graphics”. In IEEE Computer Graphics and Applications, v. 8, n. 3, p. 48-68, 1988.

[5] J. Bittner. “Hierarchical Techniques for Visibilty Computations”. PhD Thesis, Faculty of Electrical Enginnering, Czech Technical University, Prague, 2002.

[6] J. Bittner, V. Havran, P. Slavík. “Hierarchical Visibility Culling with Occlusion Trees”. In Proceedings of Computer Graphics International, p. 77-80, 1998.

[7] R.S. Moreira. “V-Mask: Uma Abordagem Híbrida e Conservativa Utilizando Bitmasks para Visibilidade Volumétrica em *Smartphones*”. Dissertação de Mestrado. PPGIA, UNIFOR. Fortaleza-CE, Brazil, 2011.

[8] H. Samet “Spatial Data Structures”. In Modern Database Systems: The Object Model, Interoperability and Beyond, Addison-Wesley, p 361-385, 1995.

[9] VALVE. *Team Fortress*. Disponível em: <www.teamfortress.com>. Último acesso em: 25/04/2013.

[10] N. Andrysco, X. Tricoche. “Implicit and Dynamic Trees for High Performance Rendering”. In Proceedings of Graphics Interface, p. 143-150, 2011.

[11] STEAM. Steam Powered - Top Played Games Status. Disponível em: <http://store.steampowered.com/stats/ >. Último acesso em 3/5/2013.

[12] J. Airey, J. Rohlf, Jr.F. Brooks. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. ACM Computer Graphics (Proceedings of Symposium on Interactive 3D Graphics), 24:41-40, 1990.

[13] Cohen-Or, D., Chrysanthou, Y., Silva, C.T., Durand, F. (2002). “A Survey of Visibility for Walkthrough Applications”, IEEE Transactions on Visualization and Computer Graphics, v. IX, p. 412–431.

[14] Wald, T. Ize, A. Kensler, A. Knöll, S.G. Parker. “Ray Tracing Animated Scenes using Coherente Grid Traversal”. In ACM Transactions on Graphics, p. 485-493, 2006.

[15] Zhang, H. Hoff III, K.E. “Fast Backface Culling Using Normal Masks”. In Proceedings of the 1997 Symposium on Interactive 3D Graphics, p. 103-106, 1997.

[16] Serpa, Y.R., Rodrigues, M.A.F. Comparando Estruturas de Particionamento Espacial e Algoritmos de Visibilidade em *Walkthroughs* por Ambientes 3D de um Jogo. In Workshop of Undergraduate Works (WUW), SIBGRAPI, 2013.

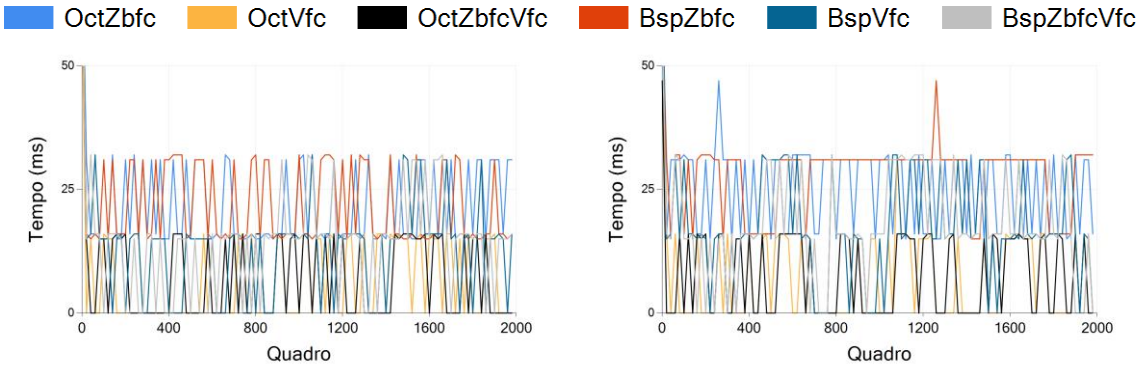

Figura 5 Tempo por quadro no cenário *Nucleus*

Figura 6 Tempo por quadro no cenário *Coal Town*

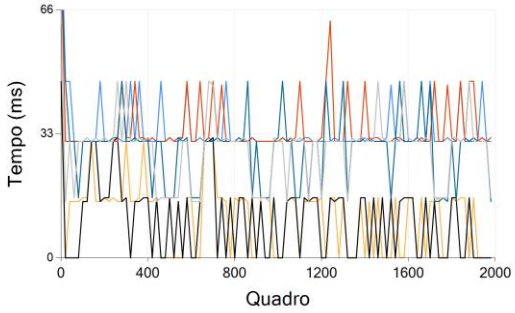

Figura 7 Tempo por quadro no cenário *Ravine*

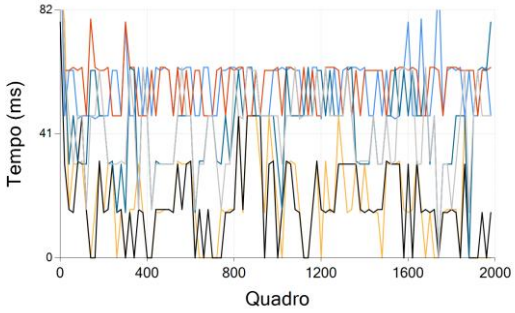

Figura 8 Tempo por quadro no cenário *Viaduct*

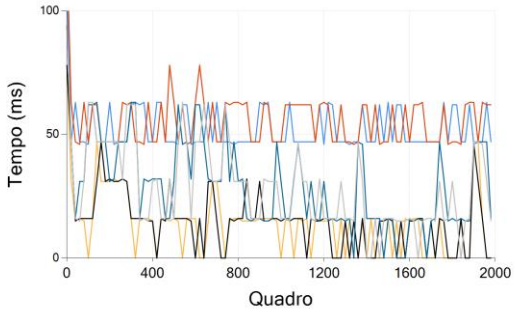

Figura 9 Tempo por quadro no cenário *Gorge*

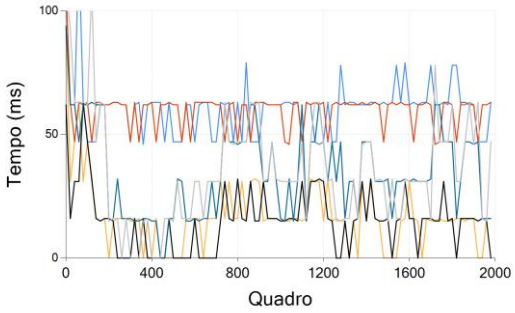

Figura 10 Tempo por quadro no cenário *Saw*

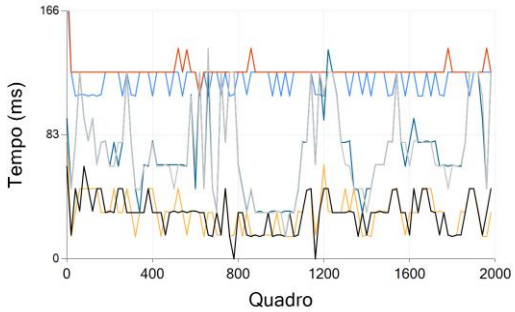

Figura 11 Tempo por quadro no cenário *Gold Rush*

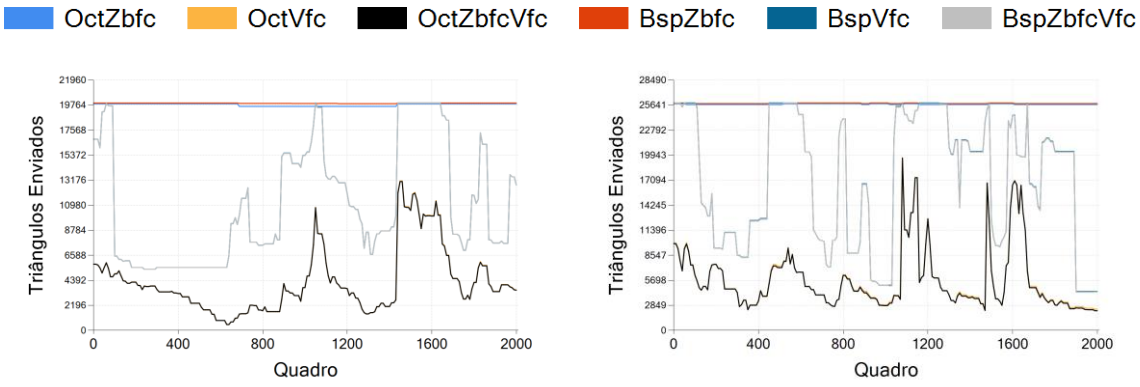

Figura 12 Triângulos enviados por quadro, no cenário *Nucleus*

Figura 13 Triângulos enviados por quadro, no cenário *Coal Town*

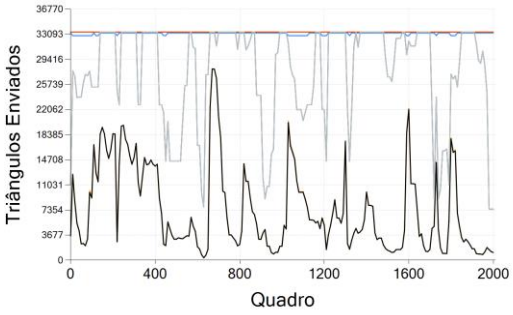

Figura 14 Triângulos enviados por quadro, no cenário *Ravine*

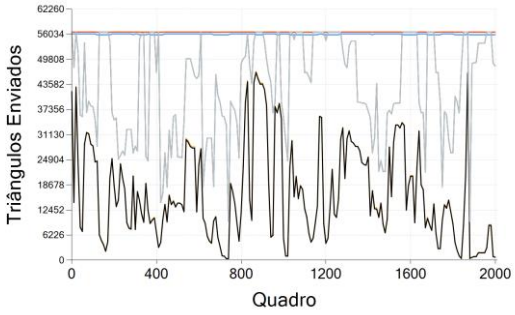

Figura 15 Triângulos enviados por quadro, no cenário *Viaduct*

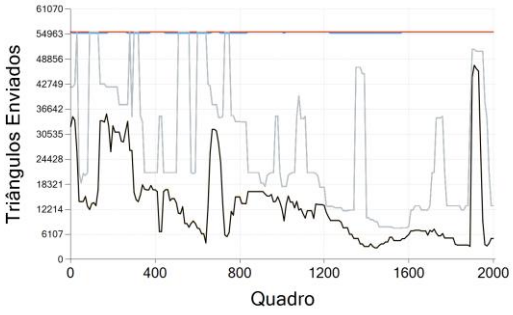

Figura 16 Triângulos enviados por quadro, no cenário *Gorge*

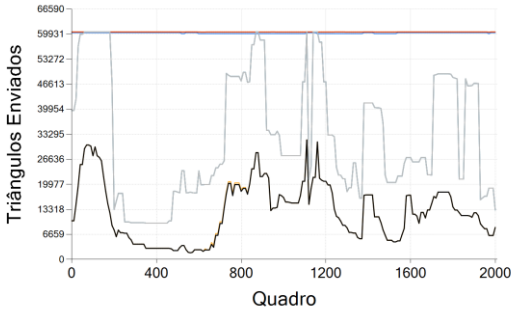

Figura 17 Triângulos enviados por quadro, no cenário *Saw*

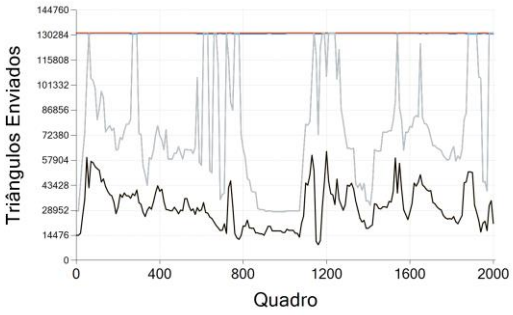

Figura 18 Triângulos enviados por quadro, no cenário *Gold Rush*