Exploração da Remoção Hierárquica por Oclusão em Simulações em Tempo Real

Vítor Cunha Miguel Leitão
DEE/ISEP
Av. Dr. António Bernardino de Almeida, Porto, Portugal
{vrc,jml}@isep.ipp.pt

Resumo

As aplicações de simulação gráfica em tempo real podem apresentar um fraco desempenho devido à complexidade de ambientes virtuais de muito grandes dimensões. Um exemplo é a simulação de condução em ambientes urbanos onde se utilizam cenas extensas com densidade de geometria tipicamente muito elevada. Nesta perspectiva, o desenvolvimento de métodos que permitem a selecção apenas da geometria visível, eliminado toda a restante, é de importância crucial pois irá diminuir a quantidade de trabalho a realizar pelo hardware gráfico. Neste grupo de métodos, que determinam a visibilidade da geometria mediante um ponto de vista sobre a cena, existe um conjunto cujo propósito é o de determinar se um dado objecto no ambiente virtual se encontra ocluso por outro(s), logo, não visível. Este tipo de algoritmos de remoção por oclusão, em particular os que usam Hardware Occlusion Queries, tornou atractiva a determinação de visibilidade em tempo real sem necessidade de extensos cálculos em pré-processamento. Neste artigo é abordada a influência na performance que o uso da remoção por oclusão, quando aplicada a cenas organizadas em estruturas de dados hierárquicas, tem em simulações de ambientes virtuais complexos de muito grandes dimensões. A aplicação de suporte desenvolvida usa as funcionalidades base que a API OpenSceneGraph disponibiliza para o render da cena e determinação de visibilidade.

Palavras-Chave

visibility, occlusion culling, real-time rendering, hardware occlusion query.

1. INTRODUÇAO

A crescente necessidade de modelos mais detalhados nem sempre é acompanhada pela evolução das capacidades do hardware gráfico. A elevada densidade de geometria (complexidade) em modelos que se estendem por vários quilómetros (dimensão) poderá comprometer a cadência de imagens e consequentemente degradar a experiência do utilizador, exemplo de modelos deste tipo são os ambientes urbanos.

Neste contexto o desenvolvimento de técnicas que permitam diminuir a quantidade de geometria a desenhar em cada imagem ganha bastante relevância. Toda a geometria que é desenhada mas que não contribui para a imagem final representa um desperdício dos recursos de processamento ao nível do CPU e GPU. Uma forma de abordar a questão é remover toda a geometria não visível a partir do ponto de vista actual, ou seja, toda a geometria que não contribui para a imagem final. Neste conjunto pode ser incluída geometria que se encontra fora do *view-frustum*, faces ocultas de objectos visíveis ou geometria oclusa por outra que se encontre mais próxima do ponto de vista.

O clássico *view-frustum culling* é uma técnica simples e rápida que evita o envio de geometria que se encontra fora do *view-frustum* para a *pipeline* gráfica mas que não elimina os objectos oclusos nele contidos. Da mesma forma, a remoção prévia das faces de polígonos visíveis cujas normais não se encontrem orientadas para o ponto de vista actual, não resolve o problema da geometria oclusa. Assim, mesmo com a aplicação destas técnicas, abordadas em grande detalhe em [Akenine-Möller08], será desenhada geometria desnecessária levando a que a mesma área da imagem seja redesenhada várias vezes.

Para lidar com este problema foram propostos ao longo dos anos vários algoritmos para determinação de visibilidade, em particular, os algoritmos de remoção por oclusão que se destinam a identificar geometria que apesar de presente no *view-frustum* não contribui para a imagem final por se encontrar oclusa. Este grupo de algoritmos evoluiu significativamente com a introdução e desenvolvimento de uma extensão do OpenGL, genericamente designada por *Hardware Occlusion Query* e denominada por **Teste de Oclusão** no restante artigo. Este permite aceder ao mecanismo do *hardware* gráfico que determina a visibilidade de uma dada geometria face a outra(s) previamente desenhada(s).

Apesar de na sua essência ser um mecanismo simples, o uso dos Testes de Oclusão de forma a obter um significante ganho na performance em simulações em tempo real ainda apresenta desafios. A latência introduzida pela

ausência imediata dos resultados de um teste durante a etapa de *Remoção* pode originar modestas performances quando comparado com a aplicação do simples *view-frustum culling* [Bittner04]. Diversos algoritmos, analisados na Secção 2, propõem estratégias para minimizar os problemas associados aos Testes de Oclusão onde o denominador comum é a diminuição do número de testes realizados durante uma simulação.

A realização de remoção por oclusão de uma forma hierárquica deverá levar a uma diminuição do número de Testes de Oclusão necessários. Se depois de aplicado o teste de visibilidade a um nó pai este for dado como ocluso, é removido de posteriores processamentos assim como toda a sua árvore filha poupando em processamento e testes. Neste artigo é analisada a influência que a remoção por oclusão tem quando aplicada a diferentes níveis hierárquicos de uma cena estática complexa de muito grandes dimensões. Com as simulações realizadas são identificadas as vantagens e fraquezas da aplicação dos testes em cada nível hierárquico e em combinações destes.

O restante artigo encontra-se organizado da seguinte forma: na Secção 2 é dada uma visão geral sobre algoritmos de visibilidade e de remoção por oclusão que usam Testes de Oclusão e serão discutidas as principais vantagens e desvantagem deste mecanismo. Na Secção 3 é apresentado o OpenSceneGraph. A Secção 4 analisa a aplicação dos Testes de Oclusão em diferentes níveis hierárquicos da cena. Na Secção 5 são descritas as características dos grafos de cena criados e a forma como foram realizadas as simulações. A Secção 6 apresenta os resultados experimentais obtidos nas simulações e a sua análise. Por fim, na Secção 7 são discutidas as conclusões e desenvolvimentos futuros.

2. TRABALHO RELACIONADO

Ao longo dos anos têm sido apresentados variados algoritmos direccionados ao problema que persiste desde os inícios da computação gráfica, a determinação de visibilidade. A publicação dos detalhados estudos por Cohen-Or *et al.* [Cohen-Or03] e Bittner e Wonka [Bittner03] forneceu uma taxinomia que permite a comparação das variadas abordagens ao problema da visibilidade.

Considerando o domínio da determinação de visibilidade, os diferentes métodos podem ser classificados como algoritmos que determinam a visibilidade a partir de um ponto ou de uma região. Os algoritmos que determinam a visibilidade para uma região trabalham numa fase de pré-processamento para determinar o Conjunto Potencialmente Visível. Um exemplo desta abordagem é o método de células e portais para modelos arquitectónicos [Teller91]. Algumas considerações podem ser feitas relativamente a este tipo de algoritmos:

- O conjunto encontrado é apenas válido para a configuração original da cena para o qual foi calculado, logo pouco adequado a cenas dinâmicas;
- O conjunto calculado para um ponto no interior de uma célula é demasiadamente conservador;
- Fazer a sua determinação para todas as células representa uma tarefa computacionalmente dispendiosa, realizada em pré-processamento significa longos períodos de inicialização da aplicação.
- É difícil implementar a determinação de um Conjunto Potencialmente Visível preciso para uma cena de carácter geral. Enquanto em modelos de interiores de edifícios as paredes actuam como divisores naturais em cenas exteriores a tarefa pode-se revelar mais difícil.

Os métodos que determinam a visibilidade a partir de um ponto trabalham em *runtime* [Greene93, Zhang97, Wonka00, Klosowski01, Hillesland02, Bittner04, Guthe06, Mattausch08]. Estes métodos exigem mais cálculos em cada *frame* mas apresentam uma maior flexibilidade permitindo cenas dinâmicas e realizam naturalmente a fusão de oclusores devido a operarem no espaço das imagens. Este tipo de algoritmos resolve os quatro problemas associados à determinação do Conjunto Potencialmente Visível.

Um algoritmo de remoção por oclusão conceptualmente importante mas que nunca foi implementado na totalidade em *hardware* foi apresentado por Greene *et al.* [Greene93] com a designação de *Z-buffer* Hierárquico. Apesar da variedade de propostas, antes da existência de *hardware* dedicado os algoritmos de remoção por oclusão eram considerados demasiado custosos para serem usados na prática, com algumas excepções como o *Hierarchical Occlusion Maps* de Zhang *et al.* [Zhang97] e o dPVS de Aila *et al.* [Aila04].

Com a introdução dos Testes de Oclusão por *hardware*, a remoção por oclusão tornou-se atractiva de ser realizada em *runtime*. Este mecanismo, que retorna o número de píxeis visíveis de uma geometria simples sem a necessidade de ler de volta o *Z-buffer*, apareceu pela primeira vez no *hardware* gráfico VISUALIZE fx da Hewlett-Packard [Scott98]. Mais tarde, em 2002, foi introduzida pela NVIDIA com a placa aceleradora gráfica GeForce3 uma extensão de OpenGL mais evoluída denominada de *GL_NV_occlusion_query* [NV02]. Actualmente também existe como uma extensão oficial ARB [ARB03].

2.1 Teste de Oclusão

De uma forma simples, com os Testes de Oclusão a aplicação pode questionar o *hardware* para saber se um determinado conjunto de polígonos é visível quando comparados com o conteúdo actual do *Z-buffer*. Esses polígonos são frequentemente os limites dos volumes envolventes de um objecto mais complexo. Se nenhum desses polígonos for visível, então o objecto pode ser removido de posteriores processamentos. A implementação em *hardware* realiza a *scan-convertion* dos polígonos a testar e compara as suas profundidades ao *Z-buffer*. Se todas as profundidades estiverem escondidas, todos os polígonos estão oclusos. Portanto, estes testes operam no espaço da imagem [Cohen-Or03].

O Teste de Oclusão introduzido pela Hewlett-Packard apresenta duas grandes limitações [Akenine-Möller08].

Em primeiro lugar apenas retorna verdadeiro ou falso para indicar a visibilidade do objecto questionado. Um objecto que apenas contribua com alguns píxeis para a imagem final pode ser, eventualmente, removido na sua totalidade sem que o resultado final sofra significativamente. Em segundo lugar, a extensão *GL_HP_occlusion_test* da HP [HP97] é implementada como um mecanismo de pára-e-espera, a aplicação deverá suspender a sua execução enquanto espera pelo resultado do teste. Esta característica impede que os testes sejam serializados o que condiciona a exploração do paralelismo entre CPU e GPU.

A introdução da extensão pela NVIDIA veio resolver as limitações da extensão da Hewlett-Packard. Com a *GL_NV_occlusion_query* é possível realizar testes em série e obter os respectivos resultados numa fase posterior. Assim o CPU fica livre para realizar outros processamentos necessários à aplicação. O Teste de Oclusão também retorna um número inteiro que indica quantos píxeis passaram no teste de profundidade, i.e., a quantidade de píxeis visíveis. Se esse número for inferior a um determinado limite de píxeis, então algumas aplicações podem decidir não realizar o *rendering* dos objectos desse volume envolvente. Neste caso sacrifica-se a correcção da imagem final pelo desempenho. Quando os Testes de Oclusão são mencionados no contexto da computação gráfica moderna, a extensão da NVIDIA está usualmente implícita.

Apesar da evolução dos testes o seu uso em algoritmos de remoção por oclusão continua a não ser trivial devido ao custo que representa para a aplicação. Para além do custo associado ao teste em si existe a latência entre a realização do teste e a disponibilidade do resultado. Esta latência decorre do atraso no processamento do teste em longas *pipelines* de *rendering*, o tempo de processamento do teste e o custo de devolver o resultado ao CPU. Esta situação causa dois problemas quando os testes são usados de forma sequencial [Bittner04]: CPU *stalls* e GPU *starvation*.

Após a emissão de um teste o CPU fica à espera do seu resultado e não fornece mais dados ao GPU para processamento. Quando o resultado fica finalmente disponível, a *pipeline* do GPU poderá já estar vazia. Como consequência, o GPU terá que esperar que o CPU processe o resultado do teste antes que este lhe envie mais trabalho. Para evitar os CPU *stalls*, a melhor estratégia é preencher os tempos de latência com outros processamentos necessários á aplicação incluindo a realização de outros testes, Figura 1. Dado estes poderem correr simultaneamente, não é necessário esperar pelo resultado de um antes da realização de um novo teste, sendo o seu resultado usado posteriormente quando disponível [Kovalcík05].

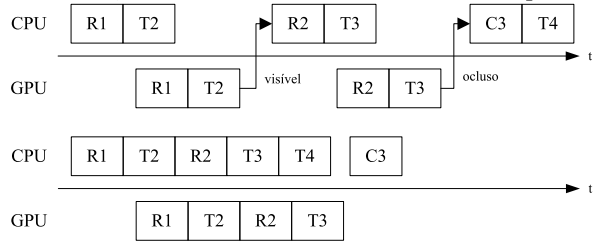

Figura 1 - Comparação de esquemas de gestão de testes onde Rn, Tn e Cn representam as tarefas de *rendering*, emissão de testes e o *culling* do nó n. No topo, a abordagem pára-e-espera onde se verifica o CPU *stall* e o GPU *starvation*. Em baixo, um possível esquema para preencher os tempo de latência, usando os resultados dos testes quando disponíveis. Neste exemplo o nó 2 é assumido como visível, procedendo com o seu render antes mesmo da disponibilidade do resultado do teste. Figura adaptada de [Bittner04].

Apesar dos problemas descritos, o uso dos Testes de Oclusão como mecanismo que auxilia em *runtime* a decisão de que geometria remover apresenta várias vantagens:

- Qualquer objecto pode ser usado como oclusor. Como para o teste é usado o conteúdo do *Z-buffer*, qualquer geometria da cena que já tenha sido desenhada na altura do teste funciona como oclusor;
- Fusão de oclusores implícita, pelo princípio descrito no ponto anterior;
- A flexibilidade de oclusores também se aplica à geometria oclusa, visto que para o teste podem ser usados os volumes envolventes ou a geometria em si.
- Como é um mecanismo simples, pode ser facilmente integrado em algoritmos de *rendering* existentes. O custo de implementações baseadas neste mecanismo é bastante inferior a algoritmos que recorrem apenas ao CPU para determinação de visibilidade.

No entanto existem situações em que o uso destes testes não é adequado mesmo se a cena for complexa. Por exemplo, um simulador de voo não é um bom candidato a usar este mecanismo pois existirá muita geometria visível sem que haja possibilidade de a remover por oclusão.

2.2 Remoção por Oclusão

O uso de Testes de Oclusão para determinação de visibilidade é tentador. Os algoritmos que os usam tentam contornar os problemas descritos anteriormente minimizando o número de testes realizados através da exploração da coerência temporal em conjunto com outras técnicas.

Hillesland *et al.* [Hillesland02] apresentou um algoritmo denominado de *Fast and Simple Occlusion Culling* (FSOC). Este algoritmo é um dos primeiros a usar a extensão da NVIDIA e divide a cena usando uma estrutura em grelha regular. Na altura do render de uma *frame* a grelha é processada por camadas no sentido de frente para trás relativamente ao ponto de vista actual. Os autores salientam o problema da latência dos testes e a solução por eles apresentada é de simplesmente realizar *N* testes de cada vez e depois obter as correspondentes *N* respostas, onde *N* é um número que tem que ser ajustado manualmente para se atingir a melhor performance possível. Este algoritmo não usa nenhum tipo de coerência temporal na determinação de visibilidade e apresenta limitações em cenas com densidade de geometria variável.

O algoritmo *Coherent Hierarchical Culling* (CHC) apresentado por Bittner *et al.* [Bittner04] explora a coerência

espacial e temporal para evitar quebras na *pipeline* gráfica. Durante o render de uma *frame* a estrutura de dados em árvore é varrida hierarquicamente da frente para trás e termina ou em nós folha ou em nós internos que sejam determinados como oclusos. Estes nós são denominados de nós terminadores e os nós interiores classificados como visíveis são denominados de nós abertos. No decorrer de uma *frame* o algoritmo não realiza testes aos nós abertos da última *frame* fazendo-o apenas aos nós terminadores. Dentro deste conjunto, se o nó se encontrava visível na *frame* anterior (nó folha) é realizado o teste e prossegue com o *render* da geometria associada. Se o nó se encontrava ocluso é realizado o teste adiando o processamento da árvore filha para quando o resultado se encontrar disponível. É também usado um método de propagação de visibilidade onde o estado de um nó pai é alterado desde que todos os nós filhos tenham o mesmo estado de visibilidade oposto ao do pai. O algoritmo reduz o número de testes necessários evitando testes a árvores filhas de nós oclusos (coerência espacial), não realizando testes aos nós abertos e assumindo que um nó visível permanece visível durante um número pré-definido de *frames* (coerência temporal). Como limitações apresenta o facto de ser necessário realizar testes para todos os nós folha anteriormente visíveis de forma a correctamente actualizar a informação de visibilidade na hierarquia através da propagação do estado de visibilidade. Este problema foi abordado por Staneker *et al.* [Staneker04] que propôs um método que evita Testes de Oclusão a objectos que são visíveis. Este método usa diversos testes em *software* (e.g., Mapa de Ocupação) para manter um mapeamento das zonas do ecrã já desenhadas. Outra situação é quando um nó transita de removido do *view-frustum* para removido por oclusão, nesta transição um teste é realizado para todos os nós que estavam removidos do *view-frustum* na *frame* anterior. Estes comportamentos originam testes desnecessários.

Apresentado em 2006 por Guthe *et al.* [Guthe06], o algoritmo *Near Optimal Hierarchical Culling* (NOHC) partilha semelhanças com o algoritmo CHC mas adopta uma abordagem mais analítica. O princípio de funcionamento deste algoritmo baseia-se na aplicação de uma função custo/benefício a todos os nós que se encontram no *view-frustum* sendo realizado um teste se a função retornar que o custo associado à realização do teste é menor que o benefício esperado ou se o nó pertencer a alguma categoria especial (previamente ocluso). A função que calcula o custo/benefício envolve a estimação do tempo que levaria a fazer o *render* directo do nó em processamento em comparação com o tempo de realização do teste mais o tempo do *render* caso fosse determinado que o nó é visível. De forma semelhante ao algoritmo CHC, são sempre realizados testes aos nós que anteriormente foram determinados como oclusos. A um nó anteriormente visível, o teste é realizado apenas se esse nó já foi desenhado sem lhe tenha sido aplicado o teste durante n *frames*, tal que o custo do teste seja compensado pelo benefício de uma possível oclusão. O algoritmo NOHC mostra uma nítida melhoria em termos de performance relativamente ao algoritmo CHC e ao contrário deste último, nunca mostra um desempenho inferior ao uso do simples *view-frustum culling* mesmo em cenas com pouca geometria em profundidade. Uma desvantagem deste algoritmo é a sua dependência de uma etapa de calibração realizada em pré-processamento onde são medidos diversos parâmetros do *hardware*. Depender de medidas de performance de *hardware* também significa que o real ganho de performance será variável para diferentes plataformas de *hardware* tornando difícil a sua precisa previsão.

Recentemente o algoritmo CHC foi revisto por Mattausch *et al.* [Mattausch08] adquirindo uma nova designação, CHC++. Com este novo algoritmo os autores pretendem melhorar diversos aspectos do original, em particular a redução de: mudanças de estado, Testes de Oclusão desnecessários e geometria desenhada. Para conseguir reduzir o número de mudanças de estado, os nós que o algoritmo decida que necessitam de teste são colocados em filas. Várias filas são usadas para acumular nós previamente visíveis e nós previamente oclusos. Estas filas são usadas para realizar lotes de testes em vez de os realizar isoladamente, assim, o estado é apenas alterado uma vez por cada lote. Tal como o CHC, este algoritmo realiza sempre testes aos nós anteriormente oclusos, colocando-os na respectiva fila. Se um nó for anteriormente visível o algoritmo também prossegue com o seu render, mas ao contrário do CHC, não realiza imediatamente o teste. Em vez disso coloca-o na fila correspondente. Para diminuir o número de testes realizados, este algoritmo apresenta dois métodos: o primeiro é a realização de um único Teste de Oclusão englobando vários nós oclusos. A este único teste é dado o nome de *multiquery*. Para isolar grupos de nós presentes na fila de nós oclusos que sejam candidatos a constituírem uma mesma *multiquery* é usada uma função de custo/benefício. O segundo método é a distribuição dos testes realizados a nós visíveis de forma aleatória através de um intervalo de *frames*. Este método evita o alinhamento temporal que ocorria no CHC devido ao número de *frames* usado como intervalo entre a realização do teste. Para lidar com a diminuição de geometria desenhada os autores usam uma técnica que permite usar volumes envolventes mais justos à geometria que envolvem, no entanto esta funcionalidade explora características próprias da estrutura de organização de dados espaciais usada. Este algoritmo revela uma performance superior ao NOHC, CHC e *view-frustum culling* em todas as situações da simulação. De todos os métodos utilizados, a realização de testes em lote é o que produziu maior aumento relativo de performance. No entanto, o facto de ser necessário aguardar que um número pré-definido de nós esteja disponível para realizar os testes em lote, leva a um atraso na disponibilidade dos resultados, o que significa que alterações na visibilidade podem ser detectadas mais tarde do que seria desejável. Além do mais, nesta situação como um nó continuaria a ser considerado ocluso, mais testes seriam realizados sobre ele o que ainda mais aumentaria a latência.

Os algoritmos descritos pretendem ser genéricos e aplicáveis a cenas genéricas. Apesar de conscientes da importância da aplicação hierárquica dos Testes de Oclusão não foi tornado evidente nos artigos analisados o impacto que os respectivos algoritmos teriam se aplicados, na mesma cena, em diferentes níveis hierárquicos ou em combinações destes.

3. OPENSCENEGRAPH (OSG)

O OSG [OSG] é um projecto *open source* que se destina a desenvolver um conjunto de ferramentas para o desenvolvimento de aplicações gráficas de alto desempenho multi-plataforma. A iniciativa remonta aos finais dos anos 90 e actualmente conta com uma considerável comunidade de utilizadores bastante activa. Em [Martz07] é feita a descrição da arquitectura e principais características do OSG.

Na sua essência o OSG baseia-se no conceito de grafos de cena, adopta uma arquitectura Orientada a Objectos tendo o OpenGL como base. Escrito inteiramente em ANSI C++ e OpenGL, usa a *Standard Template Library* (STL) e *Design Patterns* [Gamma95] para disponibilizar bibliotecas de funcionalidades que permitem o desenvolvimento de aplicações 3D de alto desempenho. O OSG é *middleware*, situando-se na camada intermédia acima da API de baixo nível e abaixo da aplicação 3D. As vantagens apresentadas pelo OSG são o seu desempenho, escalabilidade, portabilidade e uso de grafos de cena.

Na arquitectura do OSG (Figura 2) o núcleo é o conjunto de bibliotecas que fornecem funcionalidades tanto à aplicação como aos *NodeKits*. Em conjunto, as bibliotecas do núcleo e os *NodeKits* constituem a API do OSG. As bibliotecas nucleares fornecem funcionalidades essenciais de grafos de cena e de *rendering*, para além de funcionalidades adicionais que tipicamente as aplicações 3D necessitam. Os *NodeKits* expandem as funcionalidades das classes dos nós fornecidas pelas bibliotecas nucleares. Os *Plug-ins* são bibliotecas que lêem e escrevem imagens 2D e modelos 3D. As bibliotecas de interoperabilidade permitem que o OSG seja facilmente integrado em novos ambientes.

3.1 Teste de Oclusão no OSG

Com uma abordagem orientada aos objectos aos grafos de cena, o OSG disponibiliza uma variedade de tipos de nós. Todos os nós partilham uma classe comum (*osg::Node*) com as funcionalidades especializadas de cada um definidas nas classes derivadas. A variedade de tipos de nós e a sua capacidade implícita de organização espacial permitem capacidades de armazenamento de informação que não estão disponíveis nas tradicionais APIs de baixo nível.

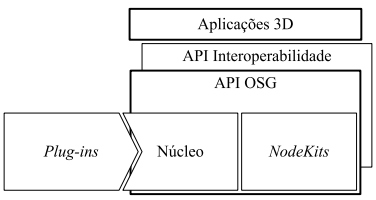

Figura 2 – Arquitectura do OSG

A biblioteca nuclear *osg* contém as classes dos nós usados para a criação do grafo de cena no OSG [Martz07]. Uma das classes derivadas da *osg::Node* é a *osg::Group* que define uma classe para qualquer tipo de nó que possa ter filhos. Como classe derivada desta última existe a classe *osg::OcclusionQueryNode*. Este tipo de nó, doravante denominado por **nó OQ**, permite realizar o Teste de Oclusão à sua árvore filha usando o correspondente volume envolvente (caixa alinhada com os eixos) e disponibilizar o estado de visibilidade à aplicação.

O nó OQ pode ser aplicado isoladamente a nós durante o processo de *rendering* ou incluídos na estrutura hierárquica do grafo de cena. Nos algoritmos apresentados anteriormente, o teste de visibilidade consiste numa operação realizada sobre nós da cena sempre que o algoritmo assim o decida. Com a possibilidade de inserir nós OQ na estrutura hierárquica da cena podem ser implementados métodos que em pré-processamento analisam o grafo e decidam, com base em determinados critérios, a inserção de nós OQ apenas em determinados locais do grafo. Assim durante o *render* de uma *frame*, o algoritmo que decide a realização ou não de Testes de Oclusão apenas irá abranger estes nós enquanto os restantes serão processados normalmente.

Durante o varrimento de *Remoção* ao grafo de cena, o nó OQ é visitado no processo de *render* de uma *frame*. Na versão do OSG usada (v 2.8.1) três situações podem ocorrer:

- O nó OQ e toda a sua árvore filha é removida de posteriores processamentos devido a encontrar-se fora do *view-frustum*.
- Para um nó determinado como ocluso, novo teste será realizado e a árvore filha do nó não é processada. No entanto, é necessário que um número pré-definido de *frames* de intervalo entre testes, para o nó em questão, tenha sido excedido.
- A terceira situação ocorre quando o nó em processamento é determinado como visível. Para que isso aconteça, o resultado do teste (nº de píxeis visíveis) terá que ser superior a um valor pré-definido de píxeis que define o limite de visibilidade para o nó em questão. Neste caso o processamento prossegue para os nós filhos. O nó também é considerado visível se o ponto de vista actual se encontrar dentro da sua esfera envolvente. Para actualizar a informação de visibilidade é realizado novo Teste de Oclusão segundo os mesmo critérios que para os nós oclusos.

Analisando o que poderá acontecer durante o processamento de um nó OQ, verifica-se que este algoritmo realiza sempre um teste a nós anteriormente visíveis e oclusos desde que já tenham decorrido n *frames* desde o último teste. No entanto, para nós determinados como visíveis na *frame* anterior o algoritmo prossegue com o processamento dos seus filhos. A abordagem de apenas realizar um novo teste quando um intervalo pré-definido de *frames* (configurável para cada nó OQ) tenha decorrido origina um menor número de testes relativamente a algorit-

mos que realizam sempre um novo teste se o nó for dado como ocluso na *frame* anterior. Por outro lado, este intervalo fará com que a disponibilidade de informação para determinação de visibilidade possa estar desactualizada na *frame* actual. Ao aumentar o valor n está-se a diminuir o número de testes realizados o que se reflecte numa imagem final menos correcta, em termos de geometria desenhada. Se o valor de n for diminuído (pior caso n=1) o aumento de testes não significa que a qualidade melhore pois com esta alteração também aumentam os custos acumulados devido ao uso de mais testes. Outro problema que advém desde intervalo é o alinhamento temporal dos novos Testes de Oclusão. No algoritmo CHC este alinhamento verifica-se nos testes realizados aos nós visíveis, enquanto no OSG esta situação também ocorre devido aos testes feitos aos nós oclusos. O efeito do alinhamento de testes é mais penalizador quando uma grande quantidade de nós transita entre visível/ocluso num curto espaço de tempo.

4. INFLUÊNCIA DA HIERARQUIA NA REMOÇÃO POR OCLUSÃO

Os grafos de cena são estruturas de organização de dados espaciais em árvore que para além da geometria da cena são enriquecidas com texturas, transformações, níveis de detalhe, estados, fontes de luz, entre outros. Sendo uma estrutura hierárquica onde vários nós podem apontar para um mesmo nó filho (instanciação), este tipo de representação de informação é chamada de Grafo Acíclico Direccionado [Akenine-Möller08]. Uma das principais motivações para o uso de estruturas de organização de dados hierárquicas é o custo de processamento que poderá passar de O(n) para O(log n). Usando este princípio, a aplicação dos Testes de Oclusão de forma hierárquica deverá apresentar ganhos devido a testes não realizados e a geometria não desenhada.

Um grafo de cena demasiado profundo significa que cada Teste de Oclusão abrange menos geometria, nesta situação o custo de desenhar a geometria fica mais próximo do custo da realização do teste. Por outro lado, um grafo de cena plano para a mesma quantidade de objectos significa que cada nó agrupa muita geometria. Se por um lado esta situação origina menos Testes de Oclusão, por outro significa que a área abrangida por toda a geometria será maior, logo mais dificilmente se encontrará totalmente oclusa.

Neste sentido, a importância de grupos de geometria advém de estes permitirem remover vários objectos com um único Teste de Oclusão. Quando os testes apenas são aplicados ao nível da geometria todos os nós OQ presentes no *view-frustum* necessitam de ser testados. O teste a vários níveis permite que grupos oclusos eliminem de posteriores processamentos todos os seus nós filhos. No entanto, quando os grupos são visíveis é possível continuar a remover por oclusão nós filho testando cada um individualmente. O Teste de Oclusão quando realizado a grupos em níveis hierárquicos superiores permite remover grandes quantidades de geometria com um único teste. No entanto, a probabilidade destes grupos serem dados como oclusos é menor que a dos grupos dos níveis inferiores devido à dimensão do volume envolvente testado.

Ao aplicar Testes de Oclusão a vários níveis hierárquicos será necessário considerar:

- A aplicação de testes a níveis hierárquicos superiores apenas apresentará vantagens em cenas de grandes dimensões e elevada complexidade.
- A realização de testes ao nível da geometria irá aumentar significativamente o número de testes realizados mesmo quando usados em conjunto com níveis hierárquicos superiores.
- A aplicação de testes a níveis hierárquicos não adjacentes levará a grandes variações no desempenho provocadas pela súbita remoção/inclusão de grandes quantidades de geometria.
- Em cenas com baixa densidade de geometria é necessária precaução na aplicação dos testes de oclusão. O custo acumulado dos testes pode não compensar o ganho obtido quando a geometria é determinada como oclusa.
- Em cenas de grandes dimensões e elevada densidade a aplicação de testes aos nós internos em conjunto com o teste ao nível da geometria permite uma remoção com diferentes níveis de resolução o que deverá ser benéfico para o desempenho da aplicação.

A dimensão do volume envolvente testado face à densidade de geometria na cena é preponderante para a eficiência do algoritmo de remoção por oclusão. A selecção do número de filhos de cada nó e a forma como a geometria é organizada no grafo de cena, deve ter em consideração a dimensão e localização (coerência espacial) dos objectos para uma remoção mais eficiente.

5. CENAS E SIMULAÇÕES

Para comprovar a influência prevista da hierarquia na remoção por oclusão e avaliar a sua utilização mais eficiente, foi desenvolvida uma aplicação de teste e 2 cenas. De uma forma genérica, todas as cenas foram criadas distribuindo aleatoriamente objectos num plano. Estes objectos constituem os nós folha.

A estrutura hierárquica do grafo de cena compreende 4 níveis: nó raiz, dois níveis de nós grupo e nós folha. Para atribuir nós folha aos nós grupo imediatamente acima na hierarquia são inseridos objectos, em número configurável, em áreas quadradas no plano. Estas áreas dividem o plano uniformemente e são de dimensão e em quantidade configuráveis. O nível hierárquico de nós grupo, pais dos nós folha, é por sua vez agrupado por um número também configurável de nós grupo, os quais são todos filhos do nó raiz.

Em cada cena foram usadas 1024 áreas, i.e., nós grupo acima dos nós folha. Estes nós grupo são, por sua vez, agrupados em conjuntos de 4x4 pelos nós grupo do nível acima. As quantidades de objectos inseridos (em todas as áreas), mostrados na Tabela 1, permitem criar as duas cenas usadas nas simulações. Estas têm a mesma dimensão e diferentes níveis de densidade geométrica.

	#edifícios	#outros	#vértices	#primitivas
Cena 1	2 028	4 056	14 920 409	5 165 020
Cena 2	25 426	10 862	160 950 781	45 527 127

Tabela 1 – Estatísticas das cenas usadas nas simulações. As cenas são constituídas por dois tipos de objectos: edifícios e pequenos objectos (outros).

Os objectos usados na Cena 1 ocupam sensivelmente 2% do plano enquanto os da Cena 2 ocupam 22%, proporcionando uma boa oclusão. A recolha de dados das simulações ocorre durante um percurso previamente gravado onde a câmara percorre a cena na forma de *walkthrough*. Este trajecto é realizado a velocidade constante e são visitadas zonas de diferente complexidade em profundidade da cena. A Figura 3 apresenta um ponto de vista sobre a Cena 2 no inicio do trajecto onde é possível verificar a oclusão que a densidade de geometria usada nesta cena permite.

A existência de edifícios proporciona boa oclusão mas simultaneamente são estes objectos que interessam remover por oclusão visto serem os que mais contribuirão para a imagem final. Os edifícios são introduzidos na cena com posição, dimensão e orientação aleatórias. Nas simulações realizadas aos pequenos objectos da cena não é aplicado o Teste de Oclusão quando realizado ao nível da geometria. O custo de processamento destes objectos é baixo quando comparado com o dos edifícios. Caso lhes fosse aplicado o teste de forma individual isso adicionaria a cada *frame* o custo desses testes sem que o benefício, quando oclusos, fosse significativo.

Para analisar a influência que os Testes de Oclusão têm na performance da aplicação quando aplicados em diferentes níveis do grafo de uma cena, em cada simulação realizada foram introduzidos nós OQ com uma das configurações ilustradas nas Figura 4. Uma configuração representa o conjunto de níveis hierárquicos ao qual é aplicado o algoritmo de remoção por oclusão. Cada simulação consiste em usar uma das configurações, para cada uma das duas cenas. Nos níveis intermédios os nós grupo são substituídos por nós OQ e ao nível da geometria são adicionados nós OQ como pais dos nós folha.

Figura 3 – Cena 2: constituída por edifícios, árvores (*billboards*) e pequenos mascote Tux. Neste caso encontram-se visíveis os volumes envolventes.

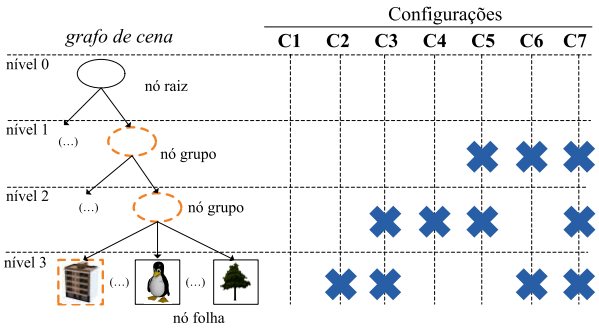

Figura 4 – Configurações de nós OQ usadas nas simulações de cada cena. Com C1 apenas é realizado o *view-frustum culling* para estabelecer uma base de comparação.

A principal métrica adoptada para avaliar o desempenho da aplicação é a duração de cálculo de cada *frame* ao longo da simulação realizada a *frame rate* constante. Como descrito na Secção 3.1 os nós OQ permitem a configuração de dois parâmetros, um dos quais é o intervalo entre Testes de Oclusão definido em número de *frames*. No entanto, em situações onde a aplicação não consegue atingir a *frame rate* proposta o número de testes realizados decai o que afecta a determinação de visibilidade. Este problema foi contornado com a alteração para um intervalo de tempo entre Testes de Oclusão. O valor usado foi de 0,167 segundos o qual representa um período equivalente a um intervalo de 5 frames (valor padrão usado pelo OSG) numa simulação realizada a 30 fps constantes. O restante parâmetro configurável, o número de píxeis a partir do qual um volume envolvente testado é considerado visível, foi definido de forma conservadora com o valor de 1 píxel.

A aplicação 3D criada para a realização das simulações foi desenvolvida em C++ fazendo uso da API OSG. A necessidade de recolher dados específicos de performance e a implementação de novas funcionalidades implicou algumas modificações na própria API. Todas as simulações descritas neste artigo foram conduzidas num PC com um CPU Intel Core2 Duo a 2,53GHz, 3GB de memória RAM e uma placa aceleradora gráfica GeForce 9650m GT com 1GB de memória dedicada.

6. ANÁLISE DE RESULTADOS

Nesta secção são apresentados os resultados das simulações realizadas para estudar a influência que a complexidade da cena e a aplicação hierárquica dos Testes de Oclusão têm no desempenho da aplicação 3D usando a abordagem de remoção de geometria oclusa proposta pelo projecto OSG, descrita na Secção 3.1.

No primeiro conjunto de simulações foi usada a Cena 1. Neste caso, a baixa densidade de edifícios não favorece a fusão de oclusores e permite que objectos distantes sejam facilmente visíveis. Assim, a aplicação de Testes de Oclusão a níveis hierárquicos que envolvam geometria dispersa por uma grande área não se revela proveitoso. A Figura 5 ilustra o resultado de quatro simulações onde cada curva representa a aplicação do algoritmo de remoção por oclusão do OSG para as configurações C1 a C4 descritas na Figura 4.

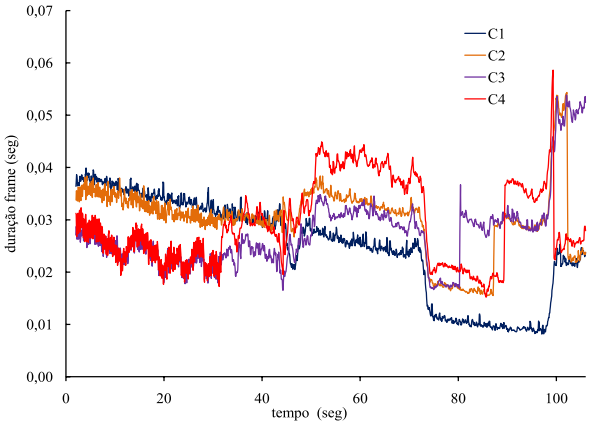

Figura 5 – Cena 1: resultados da aplicação do algoritmo de remoção por oclusão a diferentes níveis hierárquicos. Na simulação com C1 apenas é usado o *view-frustum culling*.

Nos primeiros 50 segundos da simulação, onde a densidade de geometria é maior, a duração de cálculo de uma *frame* beneficia com a aplicação dos testes. No restante tempo, a aplicação apenas do *view-frustum culling* apresenta melhor desempenho que qualquer outra configuração. Neste caso, como os volumes envolventes são dificilmente determinados como oclusos os Testes de Oclusão que resultam maioritariamente em geometria visível, o que se revela penalizador. Assim, para a Cena 1 apenas são apresentados os resultados referentes à aplicação dos testes aos níveis hierárquicos mais próximos da geometria. Os resultados obtidos nas simulações referentes às restantes configurações de nós OQ podem ser consultados na Tabela 2 e mostram que todas apresentam um desempenho médio inferior ao *view-frusutm culling* (C1).

Apesar dos resultados anteriores evidenciarem tendências positivas nas zonas de maior densidade de geometria não são conclusivos quanto às vantagens de realizar Testes de Oclusão de uma forma generalizada. No segundo conjunto de simulações foi usada a Cena 2. Nesta, estão presentes 12 vezes mais edifícios que na cena usada anteriormente. A Figura 6 apresenta o resultado da aplicação das configurações C1 a C4 à Cena 2. Deste conjunto de simulações observa-se:

- O evidente problema do alinhamento temporal dos Testes de Oclusão em C1. Nesta configuração não existe a possibilidade de remover por oclusão todo um grupo. Assim todos os nós OQ que se encontram inicialmente no *view-frustum* são testados. No decorrer da simulação, como os testes são realizados com um intervalo fixo, os testes posteriores serão realizados ao mesmo tempo influenciando o tempo de cálculo da *frame*.
- A aplicação não hierárquica dos Testes de Oclusão (C2) apresenta um ganho modesto face a C1. Apesar de conseguir remover por oclusão alguma geometria, o elevado número de testes realizados por *frame* reflecte-se na performance.
- A aplicação de Testes de Oclusão revela-se vantajosa nas secções onde a complexidade é mais elevada. A maior diminuição do tempo de cálculo das *frames* verifica-se quando é usada uma configuração que inclua o nível hierárquico 2 (C3 e C4). Também se observa que nestas configurações os tempos das *frames* são mais constantes ao longo da simulação quando comparados com C1 e C2.

Dos pontos enunciados destaca-se o último que observa que o grande ganho de performance acontece quando é aplicado o Teste de Oclusão ao nível hierárquico 2, neste caso com a realização de um único teste é possível remover todos os nós de geometria por este englobados. A diferença observável entre as configurações C3 e C4 localiza-se na zona de menor complexidade. Enquanto ambas as configurações conseguem remover grupos, com C3 é possível remover por oclusão nós de geometria numa situação onde C4 terá que desenhar todos esses nós.

As restantes configurações (C5, C6 e C7) aplicam o Teste de Oclusão ao nível hierárquico 1. A inclusão deste nível nos testes realizados não proporciona ganhos na mesma ordem de grandeza dos verificados com as configurações C3 e C4. Os valores apresentados na Tabela 2 demonstram que as configurações C5 e C7, em média, permitem melhorias nos tempos das *frames*, no entanto, ambas as configurações usam os testes no nível hierárquico 2. Em C6 o mesmo não acontece, como tal, a duração média das *frames* sobe para valores superiores a qualquer das configurações que usa os testes no nível hierárquico 2.

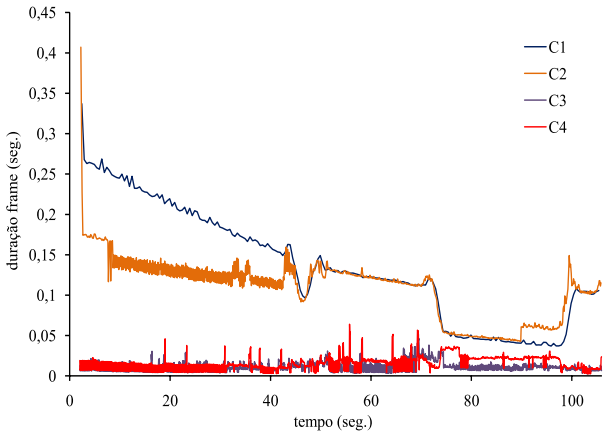

Figura 6 – Cena 2: resultados das simulações realizadas com as configurações C1 a C4.

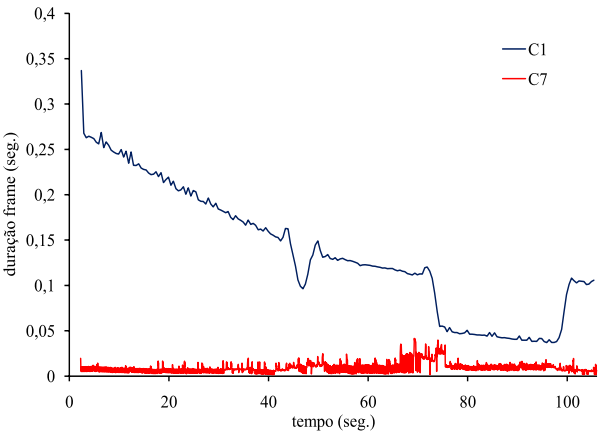

Figura 7 – Cena 2: resultados das simulações realizadas com as configurações C1 e C7.

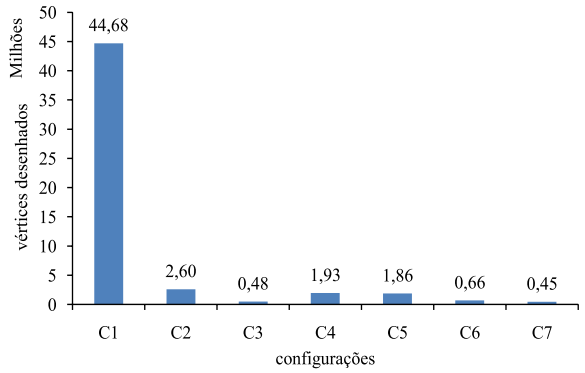

Figura 8 – Cena 2: média de vértices desenhados por *frame* para as diferentes configurações.

Como anteriormente previsto, a realização de testes em níveis hierárquicos não adjacentes implica uma grande oscilação na performance verificável pelo valor do desvio padrão apresentado pela configuração C6. Como resultado da aplicação dos testes em todos os níveis hierárquicos e beneficiando das vantagens de cada um, a configuração C7 apresenta o melhor e mais equilibrado desempenho durante a simulação com a Cena 2. As Figuras 7 e 8 ilustram esse comportamento.

Todas as simulações apresentadas foram realizadas usando um intervalo fixo entre testes aos nós visíveis e oclusos, seguindo a abordagem original do OSG. Uma implicação directa do uso de um intervalo entre Testes de Oclusão é a quantidade de resultados disponíveis para determinar a visibilidade. A aplicação do princípio da coerência temporal visa diminuir a influência negativa que os testes têm no desempenho da aplicação. No entanto, a ausência de resultados pode ter um efeito mais penalizador que a realização de testes com maior frequência.

O OSG, ao aplicar a coerência temporal a nós anteriormente oclusos está a assumir que estes permanecem oclusos durante um período de tempo. Se durante este período a geometria se torna visível, como ainda é considera oclusa, não é desenhada. Nesta situação, para além da imagem apresentada ao utilizador não ser correcta, a ausência da geometria implica que esta não será usada como oclusora o que pode levar a que seja desenhada geometria desnecessária. A somar ao custo de desenhar esta geometria há o custo da realização de Testes de Oclusão associados a esta, também desnecessários. Assim, o aumento da frequência de testes permite a determinação da geometria realmente visível com mais exactidão onde o custo acrescido a cada *frame* devido ao processamento dos testes pode ser minimizado pela geometria oclusa não desenhada.

Para testar esta abordagem, o algoritmo de remoção por oclusão do OSG foi modificado para realizar testes em todas as *frames* aos nós anteriormente oclusos, mantendo a coerência temporal para nós visíveis. Aplicado à Cena 2 (configuração C7), verifica-se uma redução de cerca de 20 mil vértices desenhados em média por *frame* sem que o aumento do número de testes tenha afectado significativamente o tempo médio por *frame* (0.0117 seg.).

		média	máxima	desvio padrão
Cena 1	C1	0.0247	0.0430	0.0094
	C2	0.0300	0.0556	0.0069
	C3	0.0285	0.0571	0.0084
	C4	0.0292	0.0593	0.0080
	C5	0.0299	0.0808	0.0082
	C6	0.0301	0.0592	0.0071
	C7	0.0289	0.0556	0.0081
Cena 2	C1	0.1361	0.3370	0.0701
	C2	0.1047	0.4070	0.0369
	C3	0.0113	0.0515	0.0057
	C4	0.0148	0.0640	0.0081
	C5	0.0121	0.0527	0.0081
	C6	0.0178	0.0808	0.0103
	C7	0.0096	0.0416	0.0056

Tabela 2 – Valores em segundos das *frames* relativas às diferentes simulações. São também apresentados os valores da *frame* máxima e do desvio padrão para evidenciar a amplitude de variação que uma configuração origina.

7. CONCLUSÃO E TRABALHO FUTURO

Este artigo apresenta os aspectos relacionados com a remoção hierárquica de objectos oclusos que não contribuem para a imagem apresentada ao utilizador. O mecanismo de *hardware* utilizado pelos algoritmos de remoção por oclusão analisados, o Teste de Oclusão, parece ser actualmente a melhor solução para o problema da visibilidade de um objecto.

Mesmo recorrendo a um algoritmo com as limitações identificadas, os ganhos de performance obtidos são bastante consideráveis especialmente quando comparados com o uso apenas do *view-frustum culling*. Relativamente a este, verificou-se um desempenho 10 vezes superior quando os testes são aplicados de forma não hierárquica (i.e., apenas aos nós folha) e 14 vezes superior quando os testes são aplicados a todos os níveis hierárquicos do grafo. No entanto, estes ganhos só são possíveis quando a densidade de geometria da cena oferece um bom poder oclusor. Nestes casos, a vantagem de aplicar os Testes de Oclusão revela-se ao permitir eliminar grandes quantidades de geometria com um único teste. Como evidenciado pelos resultados das simulações com a Cena 1, em situações de baixa densidade de geometria a aplicação dos Testes de Oclusão em níveis superiores da hierarquia não se revela melhor que o simples *view-frustum culling*.

A metodologia proposta para a exploração da coerência temporal comprovou permitir resultados semelhantes ao OSG, em termos de tempo de cálculo por *frame*, diminuindo os problemas de perda de objectos deste.

Existem várias direcções que podem ser seguidas em termos de trabalho futuro. O algoritmo pode ser melhorado incorporando algumas ideias propostas nos artigos analisados, em especial, Kovalcik e Sochor [Kovalcík05] e Mattausch *et al.* [Mattausch08] onde são aplicados métodos que com base na previsão de visibilidade, reduzindo

o número de testes realizados a nós visíveis. Outra possibilidade é o desenvolvimento de um método de análise do grafo de cena que baseado em determinados critérios (complexidade, dimensão, dispersão, etc.) introduza automaticamente nós OQ numa etapa de pré-processamento. Para além do desenvolvimento do algoritmo, também se pretende usar cenas utilizadas nos testes de alguns dos algoritmos descritos na Secção 2.2 de forma a poder analisar o desempenho da abordagem descrita neste artigo com os resultados publicados pelos respectivos autores.

8. REFERÊNCIAS

[Aila04] T. Aila, V. Miettinen. dPVS: An occlusion culling system for massive dynamic environments. *IEEE Computer Graphics and Applications*, vol. 24, nº 2, p. 86-97, Março 2004. ISSN 0272-1716.

[Akenine-Möller08] T. Akenine-Möller, E. Haines, N. Hoffman. *Real-Time Rendering (Third Edition)*. A K Peters Ltd, 2008, p. 658-680. ISBN 978-1-56881-424-7.

[ARB03] Architecture Review Board (OpenGL extension specification). *GL_ARB_occlusion_query*. 2003. <http://www.opengl.org/registry/specs/ARB/occlusion_query.txt>

[Bittner03] J. Bittner, P. Wonka. Visibility in computer graphics. *Environment and Planning B: Planning and Design*, vol.5, nº 30, p. 729-756, Setembro 2003. ISSN 0265-8135.

[Bittner04] J. Bittner, M. Wimmer, H. Piringer, W. Purgathofer. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum (Eurographics 2004)*, vol. 23, nº 3, p. 615-624, Setembro 2004.

[Cohen-Or03] D. Cohen-Or, Y. Chrysanthou, C. Silva, F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, nº 3, p. 412-431, 2003. ISSN 1077-2626.

[Gamma95] E. Gamma, R. Helm, R. Johnson, J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995. ISBN 978-0-201-63361-0.

[Guthe06] M. Guthe, A. Balázs, R. Klein. Near optimal hierarchical culling: Performance driven use of hardware occlusion queries. *Proceedings of Eurographics Symposium on Rendering 2006*. The Eurographics Association, Junho 2006.

[Greene93] N. Greene, M. Kass, G. Miller. Hierarchical Z-buffer visibility. *Computer Graphics (Proceedings of SIGGRAPH 93)*, p. 231-238, 1993. ISBN 0-89791-601-8.

[HP97] Hewlett-Packard (OpenGL extension specification). *GL_HP_occlusion_test*. 1997. <http://www.opengl.org/registry/specs/HP/occlusion_test.txt>

[Hillesland02] K. Hillesland, B. Salomon, A. Lastra, D. Manocha. *Fast and Simple Occlusion Culling Using Hardware-Based Depth Queries*. Department of Computer Science, University of North Carolina at Chapel Hill, 2002. Relatório técnico.

[Kovalcík05] V. Kovalcík, J. Sochor. Occlusion culling with statistically optimized occlusion queries. *Proceedings of WSCG (Short Papers)*, p. 109-112, 2005.

[Klosowski01] J. T. Klosowski, C. T. Silva. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, nº 4, p. 365-379, Outubro 2001. ISSN 1077-2626.

[Martz07] P. Martz. *OpenSceneGraph Quick Start Guide*. Skew Matrix Software LLC, 2007. <http://www.lulu.com/content/767629>

[Mattausch08] O. Mattausch, J. Bittner, M. Wimmer. CHC++: Coherent Hierarchical Culling Revisited. *Computer Graphics Forum (Proceedings Eurographics 2008)*, vol. 27, nº 2, p. 221-230, Abril 2008. ISSN 0167-7055.

[NV02] NVIDIA (OpenGL extension specification). *GL_NV_occlusion_query*. 2002. <http://www.opengl.org/registry/specs/NV/occlusion_query.txt>

[OSG] OpenSceneGraph. <http://www.openscenegraph.org>

[Scott98] N. Scott, D. Olsen, E. Gannett. An overview of the VISUALIZE fx graphics accelerator hardware. *Hewlett-Packard Journal*, vol. 49, nº 2, p. 28-34, Maio 1998.

[Staneker98] D. Staneker, D. Bartz, W. Straßer. Occlusion Culling in OpenSG PLUS. *Computers & Graphics*, vol. 28, nº 1, p. 87-92, Fevereiro 2004.

[Teller91] S. Teller, C. H. Séquin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 91)*, vol. 25, nº 4, p. 61-70, Julho 1991. ISSN 0097-8930.

[Wonka00] P. Wonka, M. Wimmer, D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. *Rendering Techniques 2000 (Proceedings of the Eurographics Workshop on Rendering 2000)*, p. 71-82, 2000. ISBN 3-211-83535-0.

[Zhang97] H. Zhang, D. Manocha, T. Hudson, K. E. Hoff III. Visibility culling using hierarchical occlusion maps. *Proceedings of SIGGRAPH 97*, p. 77-88, 1997. ISBN 0-89791-896-7.