Renderização Interativa em Dispositivos Móveis utilizando Algoritmos de Visibilidade e Estruturas de Particionamento Espacial

Wendel B. Silva[1], Maria Andréia F. Rodrigues (orientadora do mestrado)[2]
[1]Bolsista Mestrado FUNCAP-CE (2006-2007), [2]Bolsista DT CNPq

Mestrado em Informática Aplicada (MIA) – Universidade de Fortaleza (UNIFOR)
Av. Washington Soares, 1321 – Bloco J(30) – Fortaleza – CE – Brazil
{wendelbsilva@gmail.com, andreia.formico@gmail.com}

Resumo. *Desenvolvemos um sistema para renderização interativa em dispositivos móveis usando a API OpenGL ES. Implementamos várias combinações de algoritmos de visibilidade (view-frustum culling, occlusion culling, backface culling e um novo algoritmo, simples e rápido, nomeado backface culling conservativo). Associamos esses algoritmos a diferentes estruturas de particionamento espacial (Grid Irregular, BSP-Trees, Octrees e Portal Octrees) e comparamos seus desempenhos via testes extensivos usando diferentes ambientes 3D (da ordem de milhares de triângulos). Os resultados mostram que taxas interativas podem ser obtidas em dispositivos móveis usando-se geometria, ao invés de renderização baseada em imagem ou em ponto.*

Abstract. *We developed a system for interactive rendering on mobile devices using the OpenGL ES API. We implemented various combinations of visibility algorithms (view-frustum culling, occlusion culling, backface culling, and a new and simple, yet fast algorithm, called conservative backface culling). We associated these visibility algorithms with different settings of spatial data structures (non-uniform Grids, BSP-Trees, Octrees, and Portal-Octrees) and compared their performance through extensively testings using different 3D environments (in the order of thousands of triangles). The results show that interactive frame rates on mobile devices can be obtained using geometry rather than image-based rendering or point-based rendering.*

1. Introdução

Muitas técnicas de aceleração têm sido desenvolvidas para aumentar a velocidade de renderização em ambientes gráficos 3D [Akenine-Möller e Haines 2002], dentre essas técnicas, estão os algoritmos de visibilidade [Cohen-Or *et al.* 2002]. Estes visam a remoção eficiente de objetos (ou partes de objetos), não-visíveis ao observador, para que não sejam processados pelo *pipeline* de renderização. Vários fatores influenciam no desempenho dos algoritmos de visibilidade: número de pixels rasterizados e de objetos, distribuição dos objetos no ambiente gráfico, complexidade geométrica do ambiente, etc. Visibilidade é, portanto, um problema complexo, o qual demanda um grande poder de processamento, muitas vezes, sem uma solução ótima. O desafio é constante, devido à demanda sempre crescente por aplicações gráficas cada vez mais realistas, especialmente, quando a plataforma de execução é um dispositivo móvel, devido às restrições de processamento e memória da maioria dos modelos fabricados em grande escala.

Na área de visibilidade, os algoritmos podem ser executados em fase de pré-processamento (*offline*) ou em tempo de execução da aplicação (*online*). Operam no

espaço do objeto (usando informações 3D do ambiente) ou no espaço da imagem (usando uma representação 2D) [Bittner *et al.* 2004, Cohen-Or *et al.* 1996] do espaço 3D [Klosowski e Silva 2000, Klosowski e Silva 2001]. Aplicações gráficas em Medicina, Engenharia e Entretenimento vêm utilizando ambientes 3D amplos, com uma quantidade massiva de dados. Assim, implementar otimizações no processo de renderização é importante para garantir taxas interativas de quadros por segundo (q/s).

Dentre os algoritmos de visibilidade mais conhecidos destacam-se: o *backface culling*, o *view-frustum culling* e o *occlusion culling* [Cohen-Or *et al.* 2002]. Em termos gerais, o *backface culling* é usado para identificar e descartar as faces dos objetos que não estão voltadas para o observador; o *view-frustum culling,* para descartar os polígonos que não estão contidos no volume de visualização; e o *occlusion culling*, para descartar os objetos que estão ocultos por objetos opacos, ou seja, que não estão visíveis ao observador.

Na dissertação [Silva 2008], abordamos os seguintes problemas: o problema de visualização de aplicações gráficas 3D em dispositivos móveis com baixo poder de processamento, garantindo a renderização de quadros a taxas interativas; o problema da navegação em tempo-real nos ambientes gráficos, exibidos nas aplicações desenvolvidas para esses dispositivos; e o problema de armazenamento de cenas 3D (contendo um número massivo de triângulos) em dispositivos móveis, com pouca capacidade de memória. Assim, os resultados apresentados são relativos ao desenvolvimento, otimização e análise de desempenho de um sistema para renderização interativa de ambientes 3D (internos e externos), em diferentes dispositivos móveis (*Pocket PC iPaq* hx2490b e celular *Nokia n82*). As estratégias para solucionar os problemas mencionados, são especificadas a seguir.

Para o problema de visualização de aplicações gráficas 3D em pequenos dispositivos, apresentamos um novo algoritmo de *backface culling* conservativo (simples, porém eficiente), bem como implementamos e combinamos os algoritmos *view-frustum culling*, *backface culling* e *occlusion culling.* Para os problemas de navegação em tempo-real em ambientes contendo um número massivo de triângulos e de armazenamento de cenas 3D em dispositivos móveis, implementamos estruturas de dados espaciais (*Grids*, *BSP-Trees*, *Octrees* e *Portal-Octrees*, essas duas últimas, com diferentes níveis de profundidade) [Samet 1990]. Utilizamos então essas estruturas para representar recursivamente partições do ambiente e, adicionalmente, para buscar informações espaciais das cenas. Além disso, associamos essas estruturas a várias combinações de algoritmos de visibilidade para otimizar o desempenho da renderização e da navegação no sistema gráfico implementado.

Gostaríamos de ressaltar resumidamente as principais diferenças da dissertação em relação aos trabalhos existentes. Os algoritmos implementados por Silva [Silva 2008] operam no espaço do objeto, usando informações 3D do ambiente. Os algoritmos de visibilidade são executados em nível de *software* e a remoção da geometria não-visível ocorre no próprio dispositivo móvel. Faz-se uso da coerência espacial, de forma a otimizar a representação espacial e a busca de informações 3D, usando estruturas de particionamento espacial. Adicionalmente, até onde temos conhecimento, em nenhum dos trabalhos relacionados, algoritmos de visibilidade foram combinados entre si e associados a estruturas de dados espaciais para otimizar, analisar e demonstrar ganhos no desempenho de aplicações gráficas para ambientes 3D internos e externos, buscando a geração de taxas de renderização de q/s interativas, utilizando dispositivos móveis.

O mestrado foi concluído com as seguintes publicações científicas derivadas deste trabalho de pesquisa: 1 artigo em periódico nacional [Rodrigues *et al*. 2006] (versão estendida do tutorial apresentado no XIX *Brazilian Symposium on Computer Graphics and Image Processing*), 1 artigo em periódico internacional [Rodrigues *et al*. 2007] e 2 artigos em congressos internacionais [Rodrigues *et al*. 2008] e [Silva e Rodrigues 2009]. O texto completo da dissertação está em http://andreia.formico.googlepages.com/Dissertacao.pdf

2. Visão Geral do Sistema

O sistema gráfico proposto e implementado é composto por um conjunto de pacotes de classes (as quais incluem os algoritmos de visibilidade *view-frustum culling*, *occlusion culling, backface culling* e *backface culling* conservativo, combinados a *Grids*, *BSP-Trees*, *Octrees* e *Portal-Octrees*, estas últimas, com diferentes níveis de profundidade). Em particular, as profundidades da *Octree* e da *Portal Octree* podem ser especificadas pelo próprio desenvolvedor (maiores detalhes podem ser obtidos em [Silva 2008]).

O teste de intersecção entre um objeto e o *frustum* de visualização pode variar, dependendo da estrutura espacial escolhida. Assim, no algoritmo de *view-frustum culling* implementamos três tipos de testes: 1) entre o *frustum* e um ponto; 2) entre o *frustum* e uma esfera; e 3) entre o *frustum* e um cubo. Por exemplo, na *Octree* implementada, os testes são realizados entre as caixas envoltórias de cada nó e o *frustum*; já na *BSP-Tree*, são realizados entre o *frustum* e o plano de partição. O sistema apresenta duas implementações do *backface culling*, uma baseada no algoritmo tradicional [Akenine-Möller e Haines 2002] e outra disponível na API OpenGL ES [Wang *et al.* 2007]. No algoritmo tradicional, para cada triângulo calculamos o produto escalar entre o vetor normal de cada face dos objetos e o vetor *look-at da* câmera. Para acelerar o desempenho, no algoritmo de *backface culling* conservativo implementado, realizamos os cálculos em fase de pré-processamento. O sistema de coordenadas *x, y* é dividido em regiões. As regiões apontadas pelas direções dos vetores normais às faces dos triângulos são identificadas em tempo de execução. Se estas forem as mesmas do *look-at* da câmera, o triângulo é descartado; caso contrário, é renderizado. Utilizamos Java 1.6 para a implementação do módulo executado em fase de pré-processamento (o qual usa a API Java3D) e C++ no módulo da aplicação (este último, com a API OpenGL ES). A implementação está disponível para *Smartphones* e *Pockets PC* com *Windows Mobile* e celulares com *Symbian OS*.

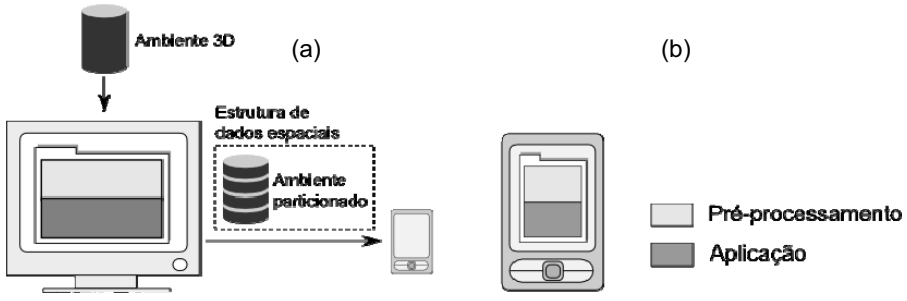


Figura 1. Em (a), o módulo de Pré-processamento (que gera o ambiente 3D particionado) é executado no computador pessoal e os algoritmos de visibilidade no dispositivo móvel. Em (b), os dois módulos executam no dispositivo móvel.

2.1. Arquitetura

É composta de dois módulos básicos: 1) o que particiona o ambiente 3D em estruturas de dados espaciais; e 2) o que contém os pacotes dos algoritmos de visibilidade e das estruturas de particionamento espacial. O primeiro e o segundo módulos podem ser executados tanto em computadores pessoais quanto em dispositivos móveis (Figura 1). Entretanto, o primeiro módulo, executado em fase de pré-processamento, depende da capacidade de processamento da plataforma de execução; já o segundo, pode ser executado em diferentes plataformas, desde que apresentem uma implementação da API OpenGL ES.

A arquitetura do sistema segue o padrão *Model-View-Control* [Buschmann *et al*. 1996], usado para particionar o ambiente em: Modelo, Visão e Controlador do Dispositivo. O Modelo administra os dados correntes do sistema e o comportamento dos objetos 3D, disponibiliza os dados para a Visão e executa as instruções que foram interpretadas pelo Controlador do Dispositivo (Figura 2). As estruturas de particionamento espacial e os algoritmos de visibilidade estão implementados no Modelo. A Visão contém informações básicas para a renderização de cenas na tela do dispositivo e relativas à câmera (ângulo de visão, razão de aspecto, planos próximo e distante). A cada quadro gerado durante a locomoção da câmera, a Visão solicita ao Modelo os dados a serem renderizados. Outros perfis de visão podem ser criados e/ou anexados, por exemplo, buscando a geração de imagens com diferentes níveis de detalhamento, dependendo dos recursos computacionais disponíveis no dispositivo. No sistema implementado, a API gráfica OpenGL ES pode ser substituída por outras, desde que apresentem características e funcionalidades similares (tipos de primitivas geométricas e formas de agrupamento, mapeamento e normalização entre sistemas de coordenadas, etc). O Controlador do Dispositivo interpreta a entrada de operações (comandos de teclado, caneta e *joystick*), transferindo as instruções já traduzidas para o Modelo. Na realidade, o Controlador do Dispositivo corresponde à interface de interação entre o usuário e a aplicação. Durante a execução do sistema, o usuário pode navegar pelo ambiente utilizando o *joystick,* bem como ativar/desativar algoritmos de visibilidade e combinações destes. Além da navegação guiada pelo usuário, o sistema oferece a opção de gravação de uma trajetória qualquer pelo ambiente, reproduzida automaticamente.

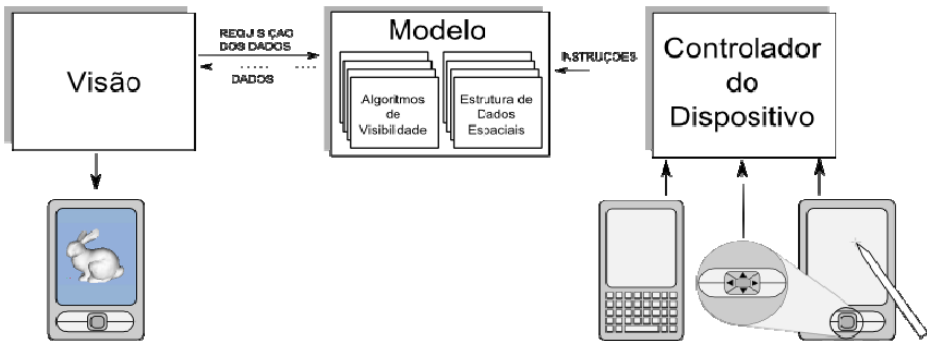


Figura 2. A arquitetura do sistema, baseada no padrão de software MVC.

3. Testes e Resultados

Realizamos 720 testes (cada um, executado 5 vezes, com médias usadas para a geração dos resultados), para a análise de desempenho das estruturas de dados e dos algoritmos de

visualização implementados no sistema. Dois modelos contemporâneos de dispositivos móveis foram usados nos testes: *PocketPC iPaq* hx2490b (128mb de memória interna, 64mb de memória heap, 520MHz PXA270 processador Intel, sem GPU) e celular *Nokia n82* (100mb de memória interna, 128mb de memória *heap*, processador ARM 11 332MHz, com GPU).

Os testes iniciais realizados nos dois dispositivos móveis incluíram oito combinações de algoritmos de visibilidade: 1) *backface culling*, 2) *backface culling* conservativo, 3) *backface culling* com *backface culling* conservativo, 4) *view-frustum culling*, 5) *view-frustum culling* com *backface culling*, 6) *view-frustum culling* com *backface culling* conservativo, 7) *view-frustum culling* com *backface culling* e *backface culling* conservativo, e 8) sem o uso de algoritmo de visibilidade. As combinações de algoritmos com melhor desempenho nos testes iniciais foram submetidas a testes adicionais, associando-as a quatro tipos de estruturas de particionamento espacial (*Grid* Irregular, *BSP-Tree*, *Octree* e *Portal Octree*, estas duas últimas, com diferentes níveis de profundidade). Estas variações foram sistematicamente testadas em seis ambientes gráficos (Tabela 1): 4 internos (Mundos 1, 2, 3 e 4) e 2 externos (Mundos 5 e 6). Os Mundos apresentam tamanhos e níveis de complexidade geométrica distintos, com um número variado de triângulos e objetos (abajur, bule, coelho, dragão, flor, pingüim, taça, vaca e mesa), distribuídos de formas diferentes.

Em fase de pré-processamento (Figura 1.a), os seis ambientes 3D da Tabela 1 foram particionados, usando-se as quatro estruturas de dados espaciais anteriormente mencionadas, ocupando espaço de armazenamento nos dispositivos. Definimos e gravamos duas trajetórias de locomoção da câmera para garantir o controle dos experimentos e facilitar a reprodução dos testes: uma para os ambientes internos e outra para os externos, contendo 1.476 e 1.999 quadros, respectivamente. Para os seis Mundos, quanto às estruturas, o nível de profundidade influenciou no desempenho da *Octree* e da *Portal Octree.* Nos testes realizados nos ambientes internos (Mundos 1, 2, 3 e 4) e externos (Mundos 5 e 6) os melhores desempenhos foram obtidos com o telefone celular *Nokia n82*. Informações mais detalhadas podem ser encontradas em [Silva 2008].

Para o Mundo 1, o melhor desempenho foi obtido associando-se os algoritmos de visibilidade *view-frustum culling, backface culling* e *backface culling* conservativo a *Octrees* de nível 4. Obtivemos taxas aproximadas de 65 e 11q/s, no melhor e pior caso, respectivamente, e taxas médias de 31q/s. Ao longo da trajetória, observamos que o tempo de processamento do *view-frustum culling* e o número de triângulos enviado ao *pipeline* de renderização (cuja correlação é de +0,97) influenciaram no desempenho das estruturas.

No Mundo 2, a combinação *view-frustum culling, backface culling* e *backface culling* conservativo associada a *Octrees* de nível 5 obteve o melhor desempenho (com taxas aproximadas de 32 e 1q/s, no melhor e pior caso, respectivamente, e taxas médias de 8q/s). Observamos que o número de triângulos enviado ao *pipeline* de renderização teve mais influência no desempenho das estruturas do que o tempo gasto na execução do *view-frustum culling*, devido à grande quantidade de triângulos presente no ambiente (102.232), com uma correlação de +0,99 em relação ao tempo de processamento para a renderização.

Nos Mundos 3 e 4, a melhor combinação foi o *view-frustum culling* com o *backface culling* conservativo, utilizando *Octrees* com 5 níveis. No Mundo 3, obtivemos taxas de 32 e 1q/s, no melhor e pior caso, respectivamente, com taxas médias de 9q/s. Já no Mundo 4, conseguimos taxas de 65 e 3q/s no melhor e no pior caso, respectivamente, com taxas médias de 17q/s. Como nos Mundos 2 e 3, no Mundo 4, o número de triângulos

enviado ao *pipeline* de renderização teve mais influência no desempenho das estruturas do que o tempo gasto para a execução do *view-frustum culling*. Nos Mundos 3 e 4, a correlação entre o número de triângulos e o tempo de renderização foi +0,99.

A melhor recomendação para o Mundo 5 foi o *view-frustum culling* associado ao *Grid* Irregular atingindo taxas de 66 e 20q/s, no melhor e pior caso, respectivamente, com taxas médias de 37q/s. Como no Mundo 1, tanto o número de triângulos quanto o tempo de processamento influenciaram no desempenho das estruturas. No Mundo 6, a melhor combinação foi o *view-frustum culling* e o *backface culling* conservativo associada a *Octrees* com 4 níveis de profundidade. Obtivemos taxas de 66 e 8q/s, no melhor e no pior caso, respectivamente, com taxas médias de 19q/s. Como nos Mundos 2, 3 e 4, o número de triângulos teve mais influência no desempenho das estruturas do que o tempo gasto na execução do *view-frustum culling*, com correlação de +0.99.

5. Conclusão e Trabalhos Futuros

A dissertação apresentou um sistema gráfico 3D para renderização interativa em dispositivos móveis, utilizando a API OpenGL ES. Várias combinações de algoritmos de visibilidade (*view-frustum culling*, *occlusion culling*, *backface culling* e um novo algoritmo, nomeado de *backface culling* conservativo) associadas a diferentes estruturas de particionamento espacial (*Grid* Irregular, *BSP-Trees*, *Octrees* e *Portal Octrees*, estas duas últimas, com diferentes níveis de profundidade) foram implementadas para otimizar o tempo de processamento gasto na renderização. Variando-se os ambientes gráficos 3D modelados (internos e externos, com diferentes níveis de complexidade geométrica e de distribuição dos objetos), as trajetórias da câmera pelos ambientes e as plataformas de execução (PocketPCs e celulares), diferentes combinações de algoritmos e estruturas foram testadas sistematicamente e análises de desempenho foram realizadas. Mostramos que a renderização interativa de ambientes 3D (mesmo contendo um número grande de polígonos), executados em dispositivos móveis, pode ser obtida com sucesso, se combinações de algoritmos de visibilidade e estruturas de particionamento espacial forem devidamente associadas. Mais especificamente, obtivemos taxas interativas (em torno de 31, 9, 17, 37 e 19q/s em ambientes contendo 6.199, 102.232, 30.199, 2.548 e 10.040 triângulos, respectivamente), com o celular Nokia n82. Em síntese, as principais contribuições da dissertação são:

- Levantamento e comparação de algoritmos de visibilidade propostos na literatura e de estruturas de particionamento espacial para a renderização de ambientes 3D e desenvolvimento de aplicações gráficas interativas;
- Realização de um estudo comparativo detalhado sobre a API gráfica OpenGL ES;
- Especificação e implementação de um novo algoritmo para *backface culling* (simples, porém eficiente), chamado de *backface culling* conservativo;
- Projeto, implementação e validação de um sistema para renderização interativa durante a navegação por ambientes 3D em dispositivos móveis, englobando, combinando e estendendo as principais propostas disponíveis na literatura; e
- Especificação e execução de uma metodologia para análise de desempenho do sistema, modelando e testando sistematicamente ambientes 3D com características geométricas variadas (número de triângulos, distribuições de objetos, trajetórias de locomoção nos ambientes, mundos internos e externos, etc), executados em

diferentes modelos contemporâneos de dispositivos móveis (PocketPC iPaq e telefone celular Nokia n82).

Como trabalhos futuros, recomendamos o armazenamento e acesso otimizado da estrutura de particionamento espacial na memória do dispositivo. Adicionalmente, alguns dispositivos não apresentam suporte à aritmética de ponto flutuante e, durante a execução da aplicação, suas variáveis são convertidas para ponto fixo, fazendo uso do processador. Desta forma, recomendamos a conversão do sistema atual para ponto fixo. Também seria interessante a implementação de obstáculos virtuais no algoritmo de *occlusion culling*. Outros algoritmos de visibilidade também poderiam ser implementados e comparados. Quanto às estruturas de particionamento, os desempenhos da *BSP-Tree* e da *Portal Octree* podem ser otimizados, por exemplo, implementando-se um conjunto de polígonos potencialmente visíveis e de um algoritmo de células e portais, respectivamente. Para finalizar, outras estruturas de particionamento espacial, como a *Kd-Tree,* poderiam ser testadas no sistema.

Referências

Akenine-Möller, T., Haines, E. (2002). "Real-Time Rendering", A. K. Peters, 2nd edition.

Bittner, J., Wimmer, M., Piringer, H., Purgathofer, W. (2004). "Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful", CGF, v. XXIII, p. 615–624.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996). "Pattern-Oriented Software Architecture", v. I, John Wiley & Sons.

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.

Cohen-Or, D., Rich, E., Lerner, U., Shenkar, V. (1996). "A Real-time Photo-realistic Visual Flythrough", IEEE Trans. on Visualization and Computer Graphics, v. II, p. 255–264.

Klosowski, J.T., Silva, C.T. (2000). "The Prioritized-Layered Projection Algorithm for Visible Set Estimation", In IEEE Transactions on Visualization and Computer Graphics, p. 108–123.

Klosowski, J.T., Silva, C.T. (2001). "Efficient Conservative Visibility Culling Using the Prioritized-Layered Projection Algorithm", IEEE Transactions on Visualization and Computer Graphics, v. VII, p. 365–379.

Rodrigues, M.A.F., Barbosa, R.G., Silva, W.B. (2006). “Desenvolvimento de Aplicações 3D para Dispositivos Móveis utilizando as APIs M3G e OpenGL ES”, *RITA,* v. XIII, p. 65-96.

Rodrigues, M.A.F., Rocha, R.S., Silva, W.B. (2008). “Interactive and Accurate Collision Detection in Virtual Orthodontics”, Em 14th *Eurographics Symposium on Virtual Environments (EGVE),* Netherlands. Eurographics Association, p. 65-72.

Rodrigues, M.A.F., Silva, W.B., Barbosa Neto, M.E., Gillies, D., Ribeiro, I.M.M.P. (2007) “An Interactive Simulation System for Training and Treatment Planning in Orthodontics”, *Computers & Graphics*, v. 31, p. 688-697.

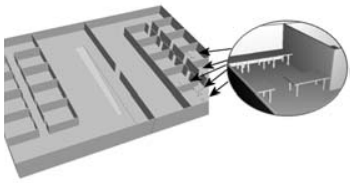
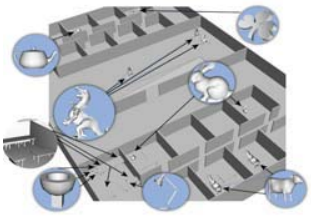
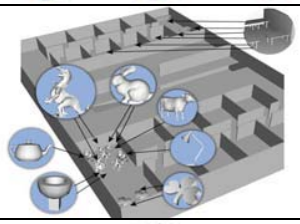
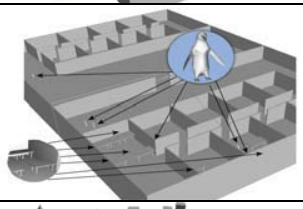
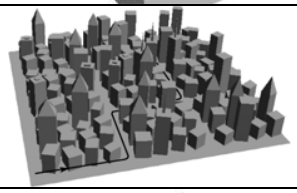
Samet, H. (1990). "The Design and Analysis of Spatial Data Structures". Addison-Wesley.

Silva, W.B. (2008). “Renderização Interativa em Dispositivos Móveis utilizando Algoritmos de Visibilidade e Estruturas de Particionamento Espacial”, Dissertação de Mestrado, MIA-UNIFOR, Fortaleza-CE. Disponível em http://andreia.formico.googlepages.com/Dissertacao.pdf

Silva, W.B. e Rodrigues, M.A.F. (2009). “A Lightweight 3D Visualization and Navigation System on Handheld Devices”, Em Anais do 24th *Annual ACM Symposium on Applied Computing (ACM SAC)*, *Track on Mobile Computing and Applications*, USA: ACM Press, p.162-166.

Wang, R.Y., Pulli, K., Popović, J. (2007). "Mobile 3D Graphics with OpenGL ES and M3G", Morgan Kauffman.

Tabela 1: Detalhamento dos ambientes modelados e utilizados nos testes de análise de desempenho.

Ambiente Gráficos	N°. de triângulos	Tamanho (Kbytes)	Tipo de ambiente	Distribuição dos objetos
Mundo 1 	6.199	214	interno	2 andares com 40 salas alinhadas aos eixos de coordenadas, acrescidos de 6 mesas
Mundo 2 	102.232	4.144	interno	2 andares com 40 salas alinhadas aos eixos de coordenadas, acrescidos de 17 objetos (bule, dragões, flor, coelhos, vacas, taças, luminária e mesas), uniformemente espalhados nas salas
Mundo 3 	102.232	4.178	interno	2 andares com 40 salas alinhadas aos eixos de coordenadas, acrescidos de 17 objetos (bule, dragões, flor, coelhos, vaca, taças, luminária, mesas), agrupados em regiões
Mundo 4 	30.199	1.048	interno	2 andares com 40 salas alinhadas aos eixos de coordenadas, acrescidos de 12 objetos (pingüins e mesas), uniformemente espalhados nas salas
Mundo 5 	2.548	108	externo	9 quarteirões, cada um com 16 edifícios não-alinhados, com diferentes orientações e alturas, uniformemente espalhados
Mundo 6 	10.040	511	externo	9 quarteirões, cada um com 16 edifícios não-alinhados, com diferentes orientações e alturas, acrescidos de 5 objetos (vacas e dragões) uniformemente espalhados