INF 2063 – Tópicos em CG III

Visualização de Modelos Massivos

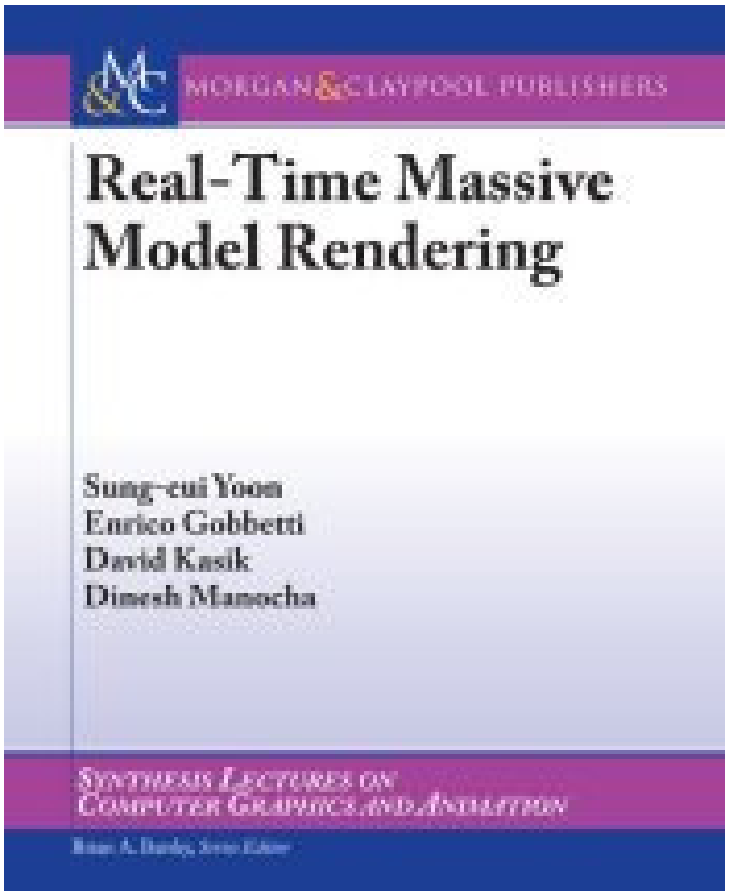
Prof. Alberto Raposo
Tecgraf / DI / PUC-Rio

Aula 02

Visibility for Walkthrough Applications

Alberto Raposo - 2010

Livro



Real-Time Massive Model Rendering

Sung-eui Yoon, Enrico Gobbetti, David Kasik, and Dinesh Manocha

2008, 122 pages
Morgan & Claypool Publishers

Tecgraf PUC-RIO VR GROUP

Organização do Livro

1. Introdução *√ (aula 1)*
2. **Visibilidade (hoje)**
3. Simplificação e Níveis de Detalhe
4. Representações Alternativas
5. Cache-Coherent Data Management

Outras referências importantes

- “A Survey of Visibility for Walkthrough Applications”. Daniel Cohen-Or, Yiogios L. Chrysanthou, Cláudio T. Silva, Frédo Duran. *IEEE Transactions on Visualization and Computer Graphics*, 9(3): 412-431. July-September 2003.
- “Visibility in Computer Graphics”. J. Bittner, P. Wonka. *Journal of Environment and Planning B: Planning and Design*, 30(5): 729-756. 2003.

Algoritmos de Visibilidade

- Abordam o problema de determinar quais superfícies ou primitivas são visíveis a partir de um certo ponto de vista ou região.
  - Determinar geometria visível exata (*visible set*) é muito complexo
  - Algoritmos de visibilidade para modelos massivos tendem a usar aproximações rápidas e conservadoras (*potentially visible set – PVS*)

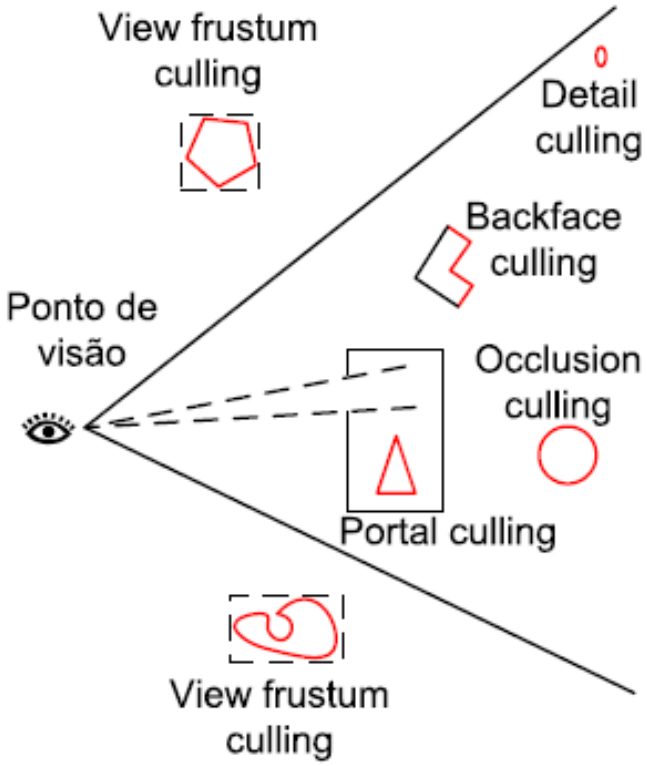
(Potentially) Visible Set

- Ideia básica: rejeitar grandes partes da cena que não estejam visíveis, para reduzir a complexidade da renderização ao conjunto de primitivas (potencialmente) visíveis
- O processo de criação do subconjunto (potencialmente) visível da cena é chamado *Visibility Culling*

Visibility Culling

- Objetivo: trazer o custo de renderização da cena para a complexidade da porção visível da cena
  - Altamente desejável em ambientes “densamente ocludido”
- Juntamente com técnicas de simplificação e representações alternativas (LOD), Visibility Culling é ingrediente essencial na visualização em tempo-real de modelos massivos

Técnicas típicas de visibility culling



HSR vs Visibility Culling

- Hidden Surface Removal (HSR) investem recursos computacionais para descobrir as partes exatas dos polígonos que não estão visíveis.
- Visibility culling simplesmente identificam polígonos que não estão visíveis, sem lidar com o nível de "subpolígono"
  - Algoritmos conservativos superestimam os polígonos visíveis, ao contrário de HSR
- Distinção entre HSR e Visibility Culling nem sempre é clara
  - Algoritmo de HSR pode ter culling como parte integrante
  - Alguns métodos colocam culling e HSR no mesmo nível

HSR e Modelos Massivos

- Atualmente, apenas 2 classes de algoritmos conseguem lidar com modelos massivos
  - Rasterização com Z-buffer
    - Em teoria, trabalha em tempo linear ao número de objetos da cena
      - Tempo sub-linear pode ser alcançado com o uso de divisões espaciais, visibility culling, etc
  - Ray casting / Ray tracing
    - Nos últimos anos atingiu capacidade de tempo real, implementação em hardware, etc

Pré-processamento

- Geralmente modelos massivos pedem algum tipo de pré-processamento para poderem executar o Visibility Culling
  - Pré-computação do próprio PVS (caso de portal culling, por ex.)
  - Organização espacial das primitivas geométricas em estuturas que facilitem os testes de visibilidade

Subdivisão Espacial dos Objetos (1)

- BVH (Bounding Volume Hierarchies)
  - Cada volume encapsula grupos de objetos do próximo nível da árvore
  - Árvore pode ser atravessada top-down
    - Se nó pai for testado e considerado totalmente invisível, descarta todos os filhos
    - Se for visível ou parcialmente visível, testa os nós do próximo nível
  - Sugestão de leitura:
    - C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. **Fast BVH construction on GPUs**, Proc. Eurographics 2009. http://mgarland.org/files/papers/gpubvh.pdf

Subdivisão Espacial dos Objetos (2)

- Spatial partitioning
  - Divide a cena em células não sobrepostas
  - Exs.: hierachical grids, octress, kd-trees
  - Kd-trees
    - Axis-aligned binary space partitioning
    - Se a partição das bounding boxes cobrirem bem as primitivas da cena, kd-trees geralmente apresentam eficiência de culling superior às outras técnicas de subdivisão espacial

Subdivisão Espacial em Modelos Massivos (1)

- Problemas:
  - Milhões de primitivas
  - Distribuição não homogênea na cena
  - Tipicamente não cabem na memória
- Desafio: encontrar o balanço entre eficiência na construção da estrutura, e eficácia da mesma para o visibilty culling.

Subdivisão Espacial em Modelos Massivos (2)

- Kd-tree
  - Escolha mais comum para modelos estáticos
  - Padrão *de-facto* para obtenção da divisão espacial é a SAH (Surface Area Heuristics), que tenta minimizar a probabilidade de um raio atravessar os vários ramos da hierarquia, a partir das áreas das superfícies das bounding boxes dos nós.
    - Sugestão de Leitura:
      - S. Popov, J. Günther, H.-P. Seidel, P. Slusallek. "*Experiences with streaming construction of SAH kd-trees*". Proc. IEEE Symp. on Interactive Ray Tracing 2006, pp. 89-94.
- BVH
  - Boa opção para modelos dinâmicos (menores)
    - Obs: modelos mais complexos dinâmicos ainda estão longe de atingir taxas interativas

Taxionomia para Algoritmos de (Occlusion) Culling (1)

- From region
  - Calculam PVS para regiões fixas (células) da cena
  - Geralmente PVS são calculados em pré-processamento
  - PVS se mantém por alguns quadros (diminui custo de processamento)
  - Boa capacidade de predição (bom para aplicações em rede e gerência de cache)
  - Só servem para aplicações especializadas: modelos de arquitetura, cenários urbanos.
  - Não lida com objetos móveis
- From point
  - Aplicados online, a partir de cada viewpoint, para computar o PVS from scratch
  - Adequado para cenas mais gerais

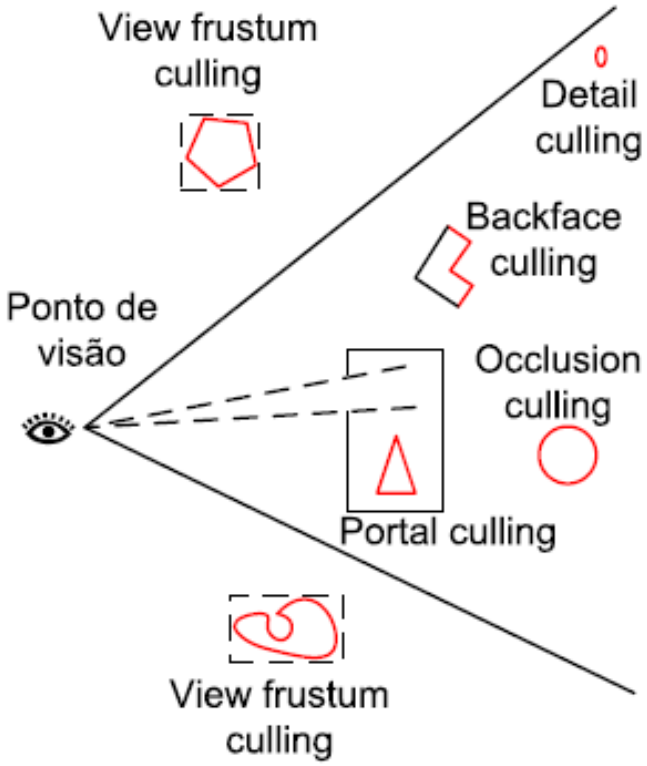
Taxionomia para Algoritmos de (Occlusion) Culling (2)

- Exato
  - Determinação de quais os polígonos estão visíveis é feita de forma precisa
  - Caro; típico de HSR
- Conservativo
  - Remove grande parte dos polígonos não visíveis, porém nem todos
  - Polígonos enviados erroneamente são tratados pela GPU posteriormente
  - Constitui grande parte dos algoritmos de visibilidade
- Aproximado / Agressivo
  - Pode remover polígonos que estejam visíveis, produzindo imagens erradas
  - Podem ser interessante para certos tipos de aplicação, pois tem melhor desempenho que os exatos e conservativos

Taxionomia para Algoritmos de (Occlusion) Culling (3)

- Pré-processado x online
- Espaço do objeto x espaço da imagem
- Software x hardware
- Cenas estáticas x cenas dinâmicas
- Específicas de occlusion culling
  - Oclusores totais x parciais
    - Todos os objetos da cena são oclusores, ou apenas alguns deles?
  - Oclusores individuais x colapsados
    - Utiliza-se objetos individuais como oclusores, ou colapsam diversos objetos para formar um oclusor?

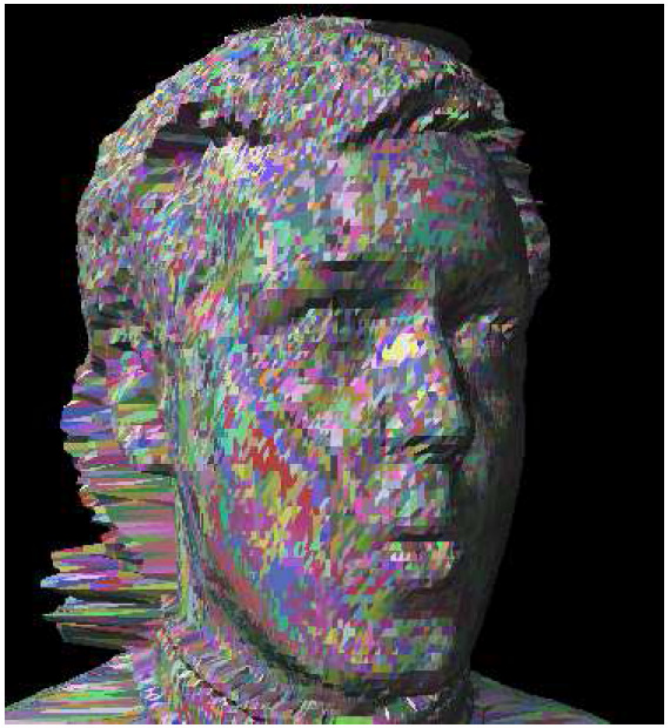
Voltando às técnicas de visibility culling



From point

View Frustum Culling e Back-face Culling

- Geralmente são as técnicas de determinação de visibilidade mais efetivas para modelos massivos
  - Por ex., em modelos de baixa "depth complexity[1]", occlusion culling costuma não ser muito eficiente
- View Frustum Culling
  - Ver apresentação Eduardo T. Carlos (aula anterior)
- Back-face Culling
  - Faces facilmente descartadas com produto escalar da normal
  - Hierarchical back-face culling
    - Faces agrupadas em clusters pela proximidade física ou proximidade no espaço das normais
    - S. Kumar, D. Manocha, W. Garret, M. Lin. "Hierarchical back-face computation". *Computers & Graphics*, 25(5): 681-692, 1999.



(c) PLB (904 clusters)

[1] número médio de polígonos mapeando em 1 pixel da imagem

Occlusion Culling (1)

- Para melhorar desempenho da renderização em modelos com alta “depth complexity”, algoritmos de occlusion culling colocam uma “terminação” no traversal da cena em caso de oclusão
  - Subdivisão espacial são essenciais, pois implementam uma travessia “de frente pra trás” de uma forma eficiente

Occlusion Culling e Ray Tracing

- Estrutura espacial hierárquica permite terminação simples
  - Uma vez que se encontra um hitpoint de um raio, pode-se ter certeza que todas as outras primitivas atrás daquela serão invisíveis
- Tendência em ray tracing em tempo real: traçado para “pacotes” de raios usando SIMD

Occlusion Culling e Rasterização com Z-Buffer

- Algoritmos modernos exploram organização espacial para atravessar a cena “de frente pra trás”
  - Testa-se cada nó visitado contra o Z-buffer
  - Termina o percorrimento quando detecta oclusão, i.e., quando os valores Z da caixa estiver atrás do Z-Buffer armazenado

Occlusion Culling em Hardware

- Artigos recentes sobre o tema:
  - Oliver Mattausch, Jirí Bittner, Michael Wimmer. “CHC++: Coherent Hierarchical Culling Revisited”. EUROGRAPHICS 2008. *Computer Graphics forum*, 27(2): 221-230. 2008.
  - Anish Chandak, Lakulish Antani, Micah Taylor, Dinesh Manocha. “FastV: From-point Visibility Culling on Complex Models”. EUROGRAPHICS 2009. *Computer Graphics forum*, 28(4). 2009.
    - *We present an efficient technique to compute the PVS of triangles in a complex 3D scene from a viewpoint. (…) Our approach traces a high number of small, volumetric frusta and computes blockers for each frustum using simple intersection tests. In practice, the algorithm can compute the PVS of CAD and scanned models composed of millions of triangles at interactive rates on a multi-core PC. We also use the visibility algorithm to accurately compute the reflection paths from a point sound source(…)*

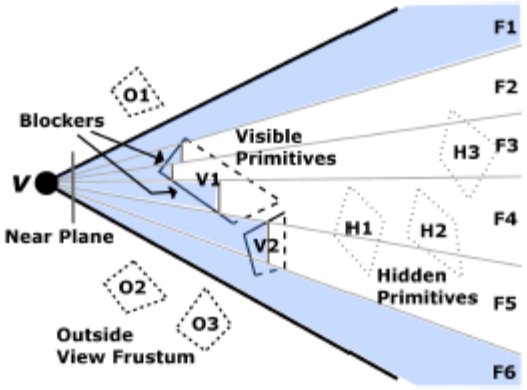


Figure 2: *Overview: We divide the view-frustum with an apex at v, into many small frusta. Each frustum is traced through the scene and its far plane is updated when it is blocked by a connected blocker. For example, frustum F_5 is blocked by primitives of object V_2 but frustum F_1 has no blockers.*

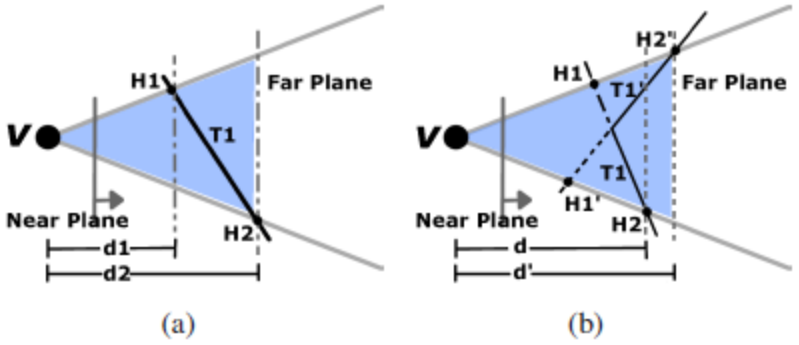


Figure 5: ***Updating Far Plane Depth****: (a) Frustum lies completely inside triangle T_1. The depth of the far plane is set to the maximum of $\mathbf{d_1}$ and $\mathbf{d_2}$. (b) Triangles T_1 and T_1' constitute the blocker. We compute the far plane depths of each triangle and use the maximum of the depth values.*

Anish Chandak, Lakulish Antani, Micah Taylor, Dinesh Manocha.
"FastV: From-point Visibility Culling on Complex Models".
EUROGRAPHICS 2009. *Computer Graphics forum*, 28(4). 2009.

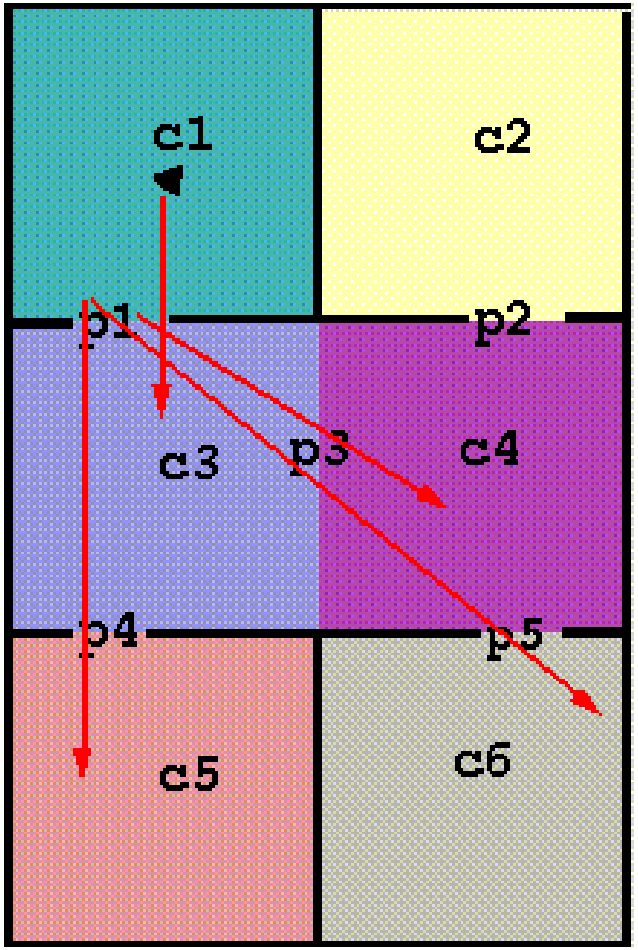
From region

Células e Portais

- Origem:
  - D. P. Luebke, C. Georges. “Portals and mirrors: Simple, fast evaluation of potentially visible sets”,, *Proc. Symp. Interactive 3D Graphics*, pp. 105-106. 1995.
- Nossa implementação:
  - R. J. M. SILVA, G. N., WAGNER, A. B., RAPOSO, M. GATTASS. Experiência de Portais em Ambientes Arquitetônicos Virtuais. *VI Symposium on Virtual Reality - SVR 2003*, pp.117-128. 2003.

Célula / Portal - Conceito

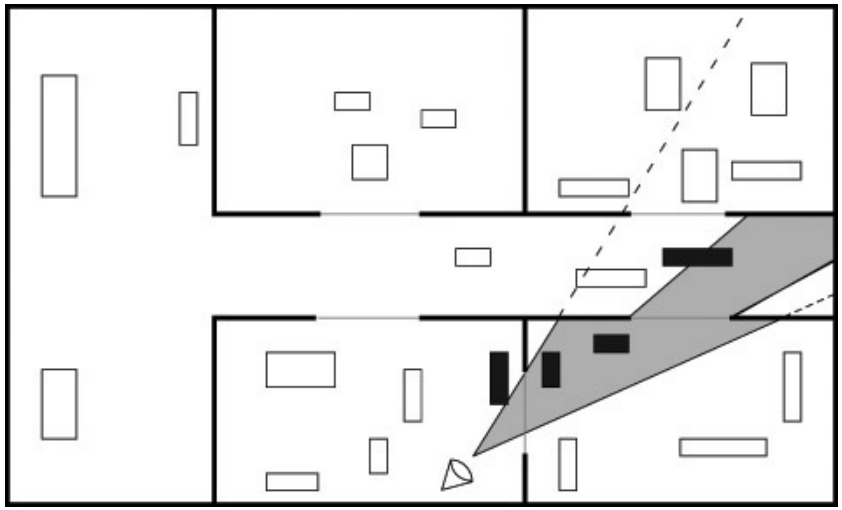
- Em um ambiente fechado, objetos presentes em quartos (células) distantes não podem ser vistos através das paredes (oclusores) e os objetos nos quartos adjacentes são vistos somente através das portas ou janelas (portais).



Célula / Portal – Pré-processamento

- O ambiente deve ser pré-processado para se obter as células e os portais:
  - Uma forma de se obter essa divisão é através do cálculo de uma BSP-Tree ou uma kd Tree;
  - O nosso trabalho parte do princípio de que as células e os portais já foram pré-calculados;
- Cálculo da Visibilidade:
  - Calcular o conjunto de objetos potencialmente visíveis (PVS): Teller / Séquin, Quake™;
  - ***Não calcular o PVS: a determinação é feita dinamicamente - Luebke***

Portais



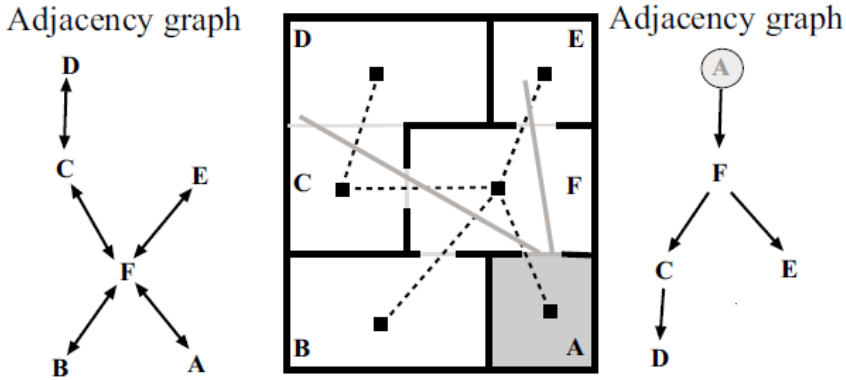


Figure 10: Cells and portals: the adjacency graph and stab tree

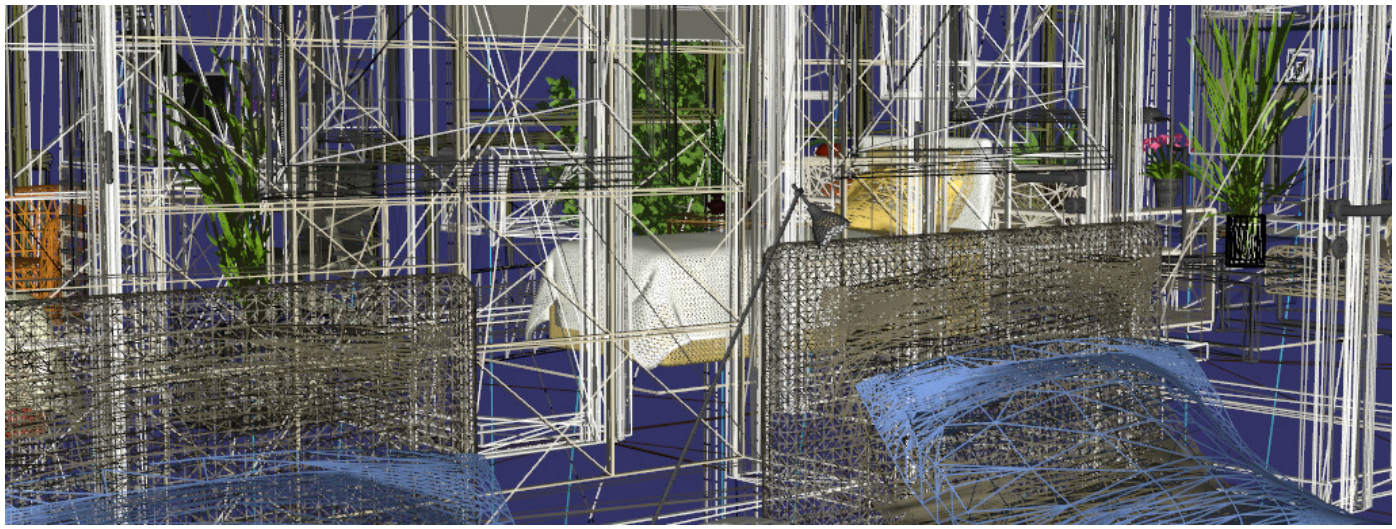
- Determinar o conjunto potencialmente visível:
  - A célula onde está o observador mais todas as células adjacentes.
- Luebke e Georges:
  - Montar o grafo de cena a partir da informação de conectividade;
  - Percorrendo o grafo, determinar quais células estão visíveis;
  - Adequado para implementação em grafos de cena: Performer / OpenSceneGraph

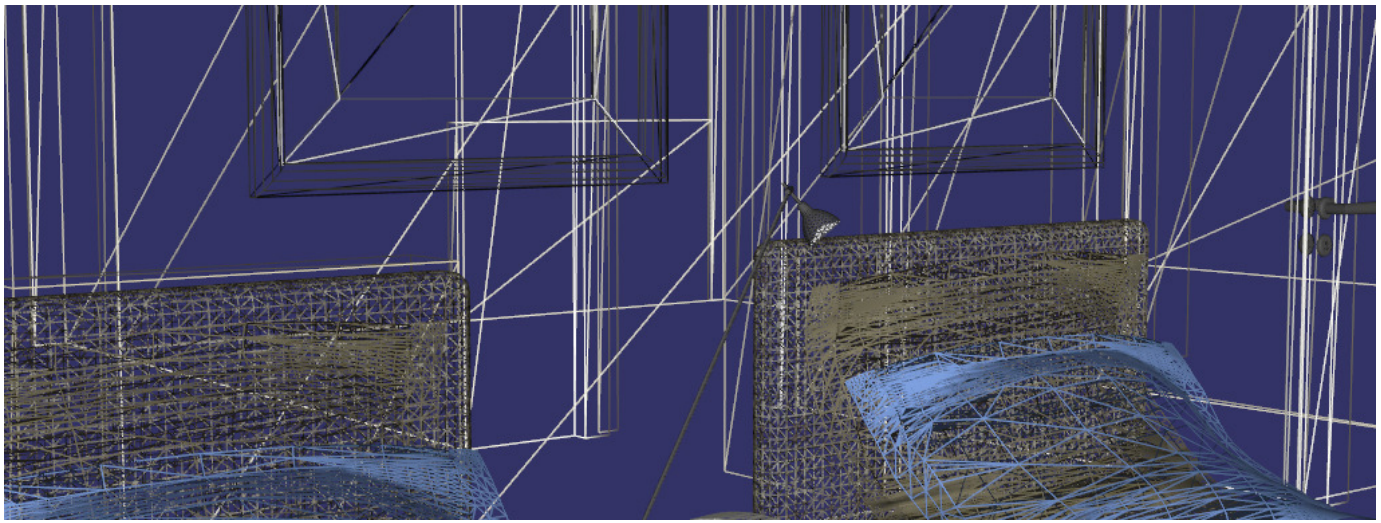
Algoritmo

- Passo 1: Determinar a célula onde se encontra o observador;
- Passo 2: Para cada célula *C* adjacente não visitada, marcá-la como visitada;
- Passo 3: Renderizar os objetos da célula *C* aplicando o algoritmo de view-frustum culling (nativo do OSG);
- Passo 4: Para cada portal *P* de *C* voltado para o observador:
  - *Passo 4.1: Projetar P na tela;*
  - *Passo 4.2: Determinar o retângulo R que envolve P:*
  - *Passo 4.3: Calcular a interseção I entre R e a projeção do frustum atual;*
  - *Passo 4.4: Se I for não vazia, calcular o novo frustum formado pela posição do observador e os quatros vértices de I e defini-lo como frustum atual;*
  - *Passo 4.5: Recursivamente retornar ao passo 2;*

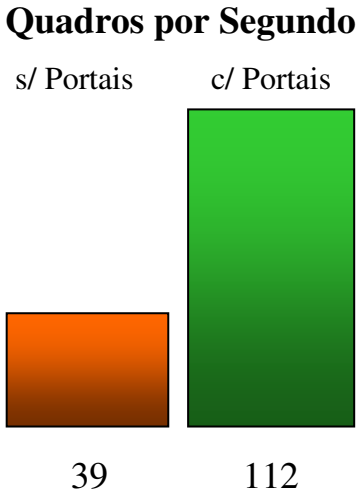
•Apesar do retângulo envolvente ser maior do que a própria projeção do portal, o descarte ainda é vantajoso. Esse fato fornece ao algoritmo sua característica conservadora.

Resultados



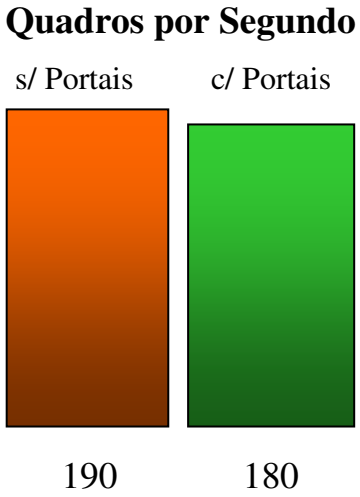


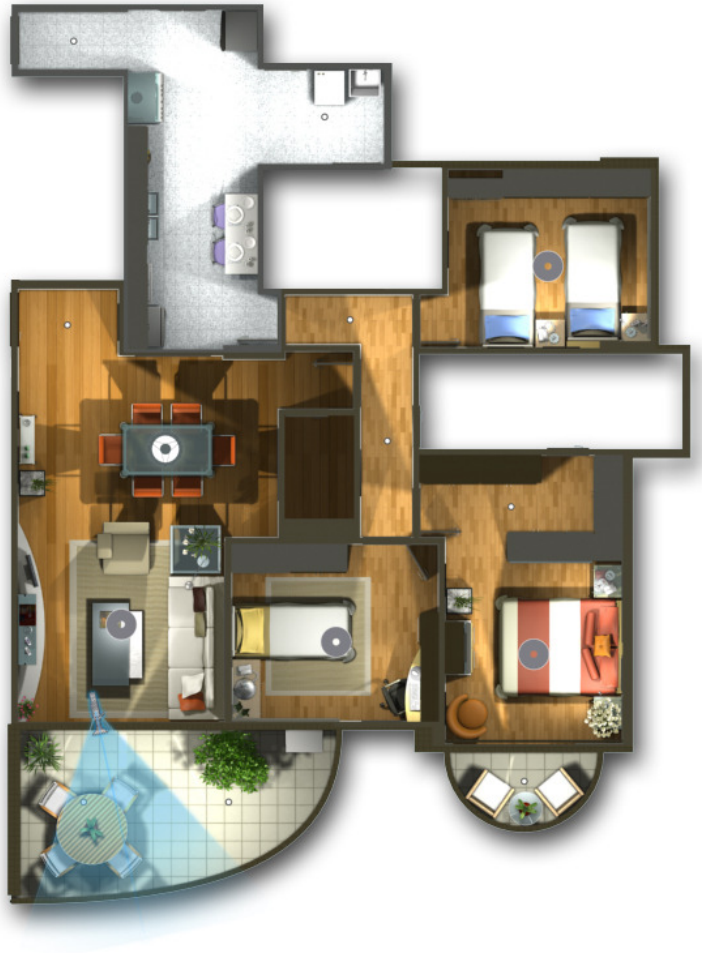
Teste de Performance (1)



Tecgraf PUC-RIO VR GROUP

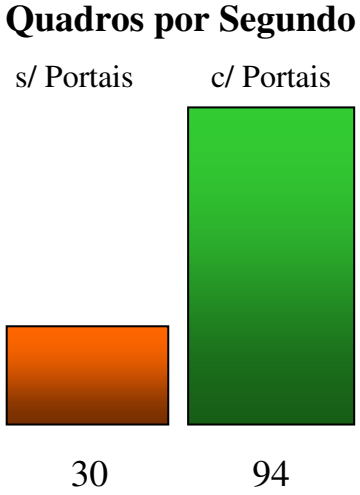
Teste de Performance (2)





Tecgraf PUC-RIO VR GROUP

Teste de Performance (3)

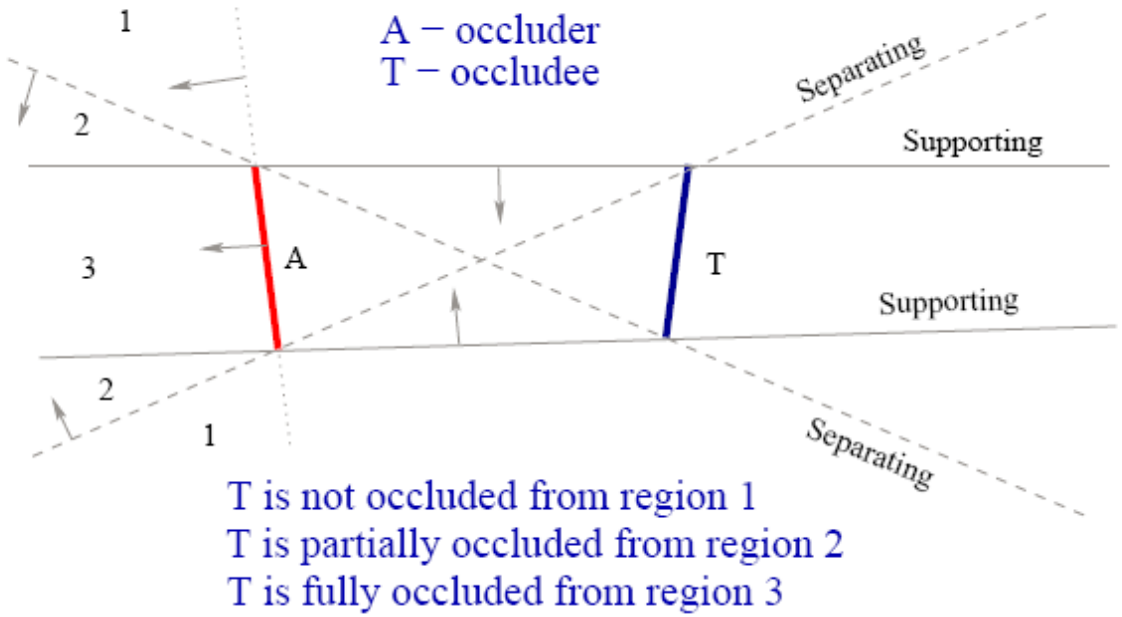


Tecgraf PUC-RIO VR GROUP

Outros Occlusion Culling Conservadores

- Usando shadow frusta
  - Assume-se que não se pode ver um objeto que esteja na sombra gerada por um objeto oclusor

- Assumindo a existência de grandes oclusores convexos



Occlusion Culling Agressivo (1)

- Dada a complexidade da geração do PVS, sua computação é geralmente baseada em amostragem
  - Não garante que 100% dos objetos visíveis aparecerão no PVS

Occlusion Culling Agressivo (2)

- Exemplo de abordagem:
  - Considerar criação do PVS para uma região, e não para um ponto único

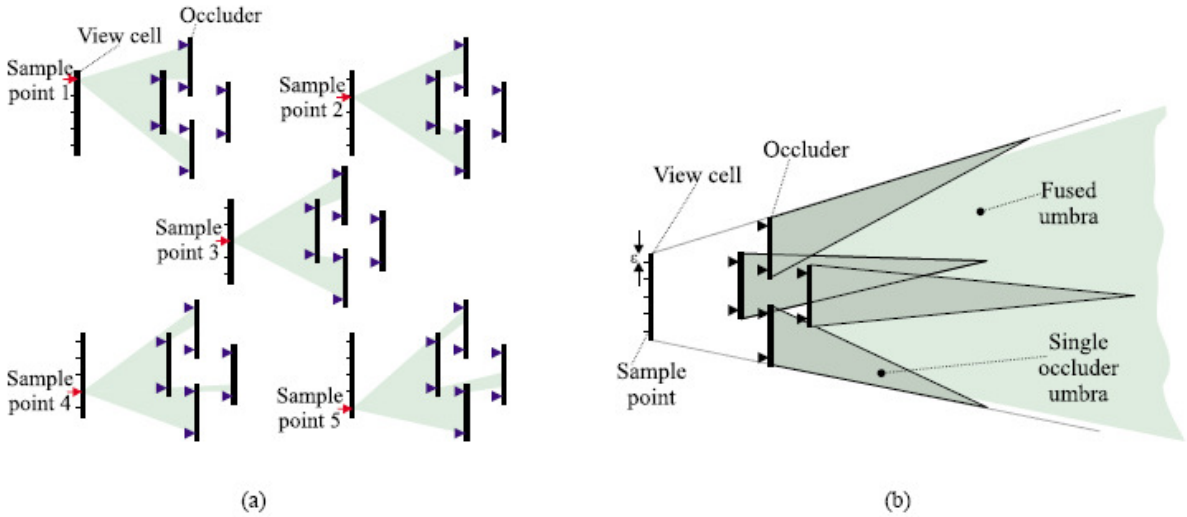


Figure 18: (a) Sampling of the occlusion from five sampling points. (b) The fused umbra from the five points is the intersection of the individual umbrae. It is larger than the union of umbrae of the original view cell. Courtesy of Peter Wonka, Vienna University of Technology.

Occlusion Culling Agressivo (3)

- Baseado em visibility cube
  - 2 viewpoints mais ou menos próximos tendem a ver imagens parecidas
    - Qualquer ponto de vista entre esses 2 viewpoints tenderão a ver a mesma coisa
  - Cria-se um conjunto de pontos de amostra dentro de uma região
  - Para cada ponto, gera-se o cubo de visibilidade nas 6 direções (similar ao CubeMap – apresentação Daniel)
  - O conjunto de polígonos que foram mapeados em pelo menos 1 ponto de uma das 6 faces constitui o PVS daquele ponto

S. Nirenstein, E. Blake. “Hardware Accelerated Visibility Preprocessing using Adaptive Sampling”. *15th Eurographics Workshop on Rendering*, pp. 207-216. 2004.

Occlusion Culling Agressivo (3)

- Baseado em visibility cube
  - S. Nirenstein, E. Blake. “Hardware Accelerated Visibility Preprocessing using Adaptive Sampling”. *15th Eurographics Workshop on Rendering*, pp. 207-216. 2004.

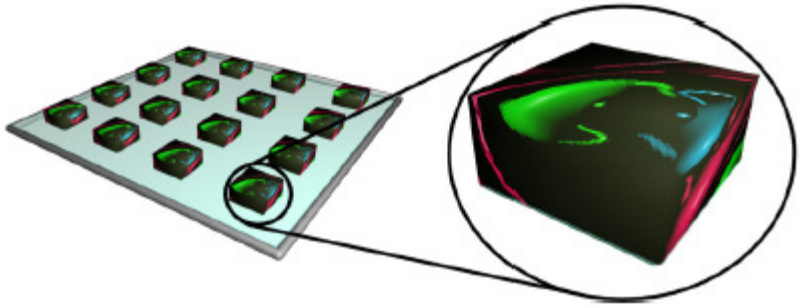


Figure 1: The Visibility Cube. *A sample of several visibility cubes over a surface. The visible geometry (of several teapots) has been projected onto the cubes.*

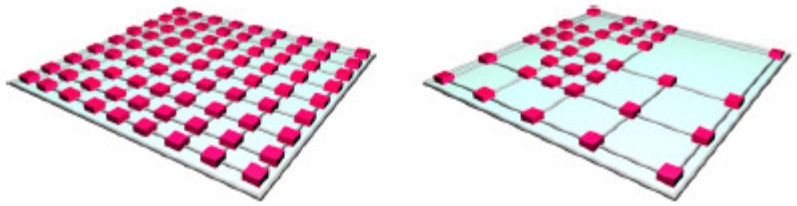


Figure 2: Uniform vs. Adaptive Sampling. *(a) A uniform distribution of visibility cubes on a 2D surface. (b) A non-uniform distribution of visibility cubes generated by an adaptive subdivision. The adaptive sub-division attempts to minimise both error, and the number of samples required. A quad-tree structure is effectively built on the surface.*

Tecgraf PUC-RIO VR GROUP

Occlusion Culling Agressivo (4)

- P. Wonka, M. Wimmer, K. Zhou, S. Maierhofer, G. Hesina, A. Reshetov. “Guided Visibility Sample”. SIGGRAPH 2006 – *ACM Transactions on Graphics*, 25(3): 494-502, 2006.

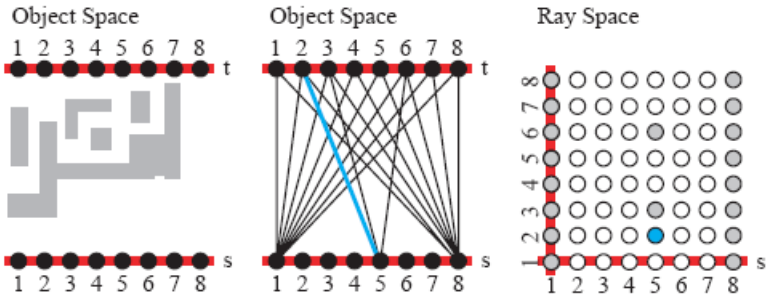


Figure 2: Sampling in object and ray space. Left: a scene with a set of objects. A view cell is shown as a line segment parameterized with s. We are interested in all rays that intersect the view cell and a second line segment parameterized with t. Middle: Shows a subset of the possible rays. One ray is highlighted in blue. Right: A depiction of the discrete ray space. Any ray in the middle figure corresponds to a point in ray space. The blue point corresponds to the blue ray in the middle figure.

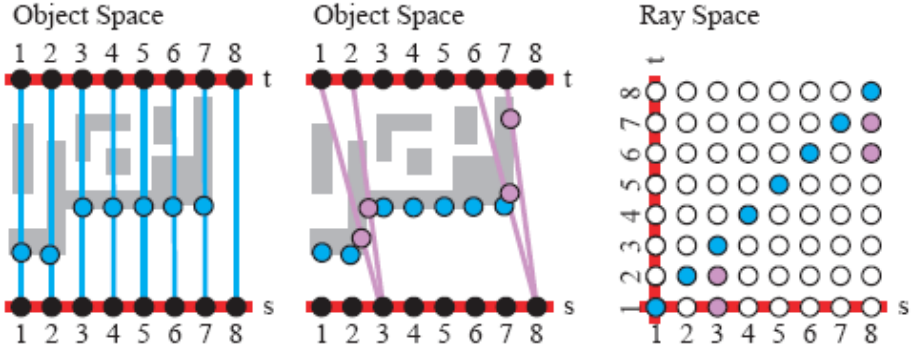


Figure 3: Left: The scene sampled orthogonally. Middle: Additional samples to capture oblique surfaces. Right: The rays used to sample the scene are shown in corresponding colors.

Tecgraf PUC-RIO VR GROUP

Estratégias de Subdivisão do Espaço

- A maioria dos algoritmos “from region” assume que as células são previamente conhecidas, ou que são subdivisões simples do espaço
- Há poucos trabalhos sobre como computar essa subdivisão
  - Exemplo: O. Mattausch, J. Bittner, P. Wonka, M. Wimmer. “Optimized Subdivisions for Preprocessed Visibility”. *Graphics Interface 2007*, pp. 335-342. May 2007

Balanço: PVS e Memória

- Células menores
  - Aumentam qualidade do PVS
  - Aumentam número de células a serem pré-computadas (maior tempo de pré-processamento)
  - Maior número de células → maior necessidade de memória
- Há trabalhos propondo técnicas para comprimir PVS, para computar PVS dinamicamente durante o Walkthrough (em thread paralela), dentre outros

Conclusões

- Aplicações com modelos massivos devem escolher as melhores técnicas de visibilidade para seu domínio
- From point techniques
  - Mais robustas, requerem menos pré-processamento e menos armazenamento em memória → mais facilmente integradas em aplicações
- From region techniques
  - Atendem situações específicas (por ex., são usados em games)