IA725 – Computação Gráfica I

Visibilidade

29/04/2008

Durand, capítulos 7 a 11
(http://people.csail.mit.edu/fredo/THESE/vo.pdf)
Möller, Real-time rendering, 2ª ed., A.K. Peters, 2002

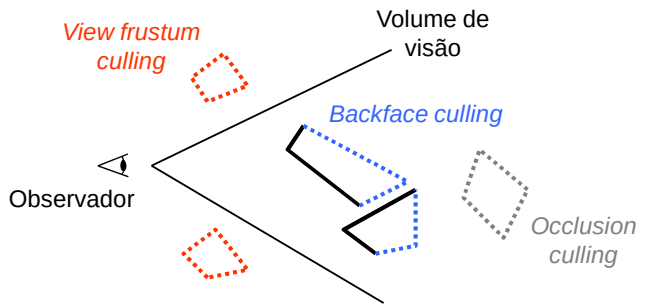
Visão geral

- Definição do problema.
- Coerência espacial e temporal.
- Estruturas de dados espaciais.
  - Hierarquia de volumes limitantes.
  - *Quadtree*, *octree*, *k-d tree*.
  - Árvore BSP.
- Algoritmos de *visibility culling*.
  - *Backface culling*.
  - *View frustum culling*.
  - *Occlusion culling* e algoritmos baseados em portais.

Introdução

- Considerando uma cena na qual podem ser desprezadas as interreflexões de luz especular entre os objetos, são 3 os motivos principais pelos quais as primitivas podem ser descartadas do processo de *rendering*:
  - Elas não estão “encarando” a câmera. Válido apenas para primitivas que compõem objetos fechados, tais como sólidos.
  - Elas estão fora do campo de visão.
  - Elas estão sendo escondidas por primitivas mais próximas do plano de projeção.

Introdução



- *Z-buffering* em *hardware* é geralmente utilizado sobre os resultados de algoritmos de *culling* para tratar as relações de visibilidade não resolvidas.

Introdução

- Quanto mais cedo ocorrer esse descarte no fluxo de *rendering*, melhor.
  - A complexidade deve ser "sensível à saída," *i.e.*, proporcional ao número de primitivas visíveis.
  - *Z-buffer* não é um algoritmo de *culling.* A complexidade de tempo é linear no número de primitivas da cena.
    - Não é eficiente para tratar, por exemplo, o modelo de um Boeing-777 com 500.000.000 polígonos.
  - Em algoritmos de *visibility culling*, a complexidade de tempo deve ser sempre sub-linear.
    - Válido para cenas "densamente oclusas."

Introdução

- Definição dos resultados obtidos por algoritmos de *visibility culling*, classificados segundo a acurácia dos resultados:
  - EVS (*Exact Visible Set*): Conjunto de todos os modelos que contribuem para a formação da imagem, e apenas estes.
  - PVS (*Potentially Visible Set*): Superestimativa do EVS. Contém o EVS e também modelos que não contribuem para a formação da imagem.
  - Aproximado (*Contribution Culling*): Subestimativa do EVS. Alguns modelos do EVS podem não ser incluídos.

Introdução

- Classificação dos algoritmos de *visibility culling* pelo local de validade dos resultados computados.
  - *Visibility culling* pontual.
    - Resultados são computados com relação a um dado ponto de vista (possivelmente com campo de visão de 360 graus).
  - *Visibility culling* regional.
    - Resultados são computados com relação a regiões da cena. Consideram, assim, todos os possíveis pontos de vista em cada região.

Introdução

- Algoritmos de *visibility culling* exploram as coerências espacial e temporal presentes na cena.
- Coerência espacial.
  - Se um objeto não é visível para um dado ponto de vista, todos os objetos contidos dentro dele também não serão visíveis para o mesmo ponto de vista.
- Coerência temporal.
  - Os resultados de visibilidade do quadro de exibição atual geralmente são parecidos com os resultados do quadro anterior.

Estruturas de dados espaciais

- Estruturas de dados que organizam a geometria em um espaço n-dimensional, de acordo com a coerência espacial ou temporal.
- Tais estruturas podem ser utilizadas para acelerar consultas sobre a sobreposição de entidades geométricas.
  - Aceleração de testes de interseção.
  - Traçado de raios.
  - Detecção de colisão.
  - **Algoritmos de culling.**

Estruturas de dados espaciais

- São geralmente organizadas de forma hierárquica.
  - Estrutura aninhada e de natureza recursiva.
  - Já vimos o exemplo da árvore BSP!
- Diferentes tipos de consultas podem ser realizadas de forma muito mais eficiente. Geralmente de $O(n)$ para $O(\log n)$.
- Construção das estruturas é computacionalmente custosa.
  - Feita como pré-processamento.
  - Entretanto, são possíveis atualizações incrementais em tempo real.

Estruturas de dados espaciais

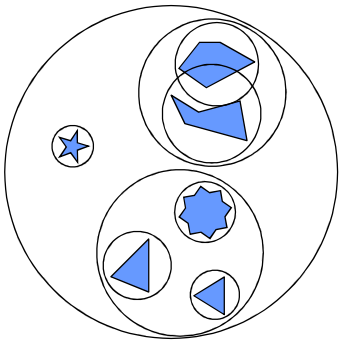
- Estruturas de hierarquia de volumes limitantes.
  - Esferas limitantes.
  - AABBs (*Axis-aligned Bounding Boxes*).
  - OBBs (*Oriented Bounding Boxes*).
  - k-DOPs.
- Estruturas de subdivisão espacial.
  - Subdividem o espaço da cena e a codificam em uma estrutura de dados.
  - Árvore BSP.
  - *Quadtree*, *octree*.

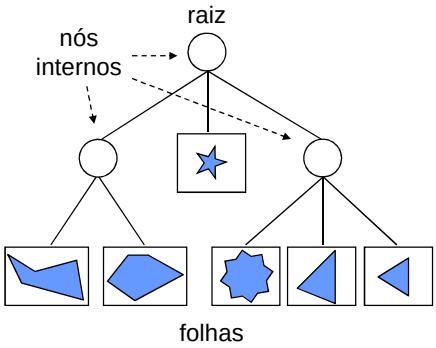
Volumes limitantes

- Um volume limitante (BV – *Bounding Volume*) é um volume que contém um conjunto de objetos (geometria ou BVs).
  - Não contribui visualmente para a geração das imagens.
  - Geometria mais simples que a geometria contida nele.
- Hierarquia de volumes limitantes.
  - Organizada em uma árvore.
  - O nó raiz corresponde ao BV que inclui toda a cena.
  - Cada nó interno possui um BV que delimita a geometria (e BVs) de todos seus descendentes.
  - As folhas possuem um BV que delimita a geometria da cena, mas não outros BVs.

Volumes limitantes

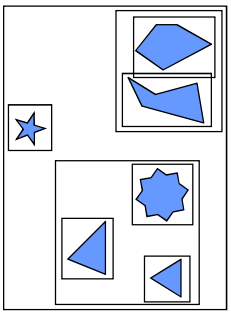
- Ilustração (em 2D) de uma hierarquia de esferas limitantes.

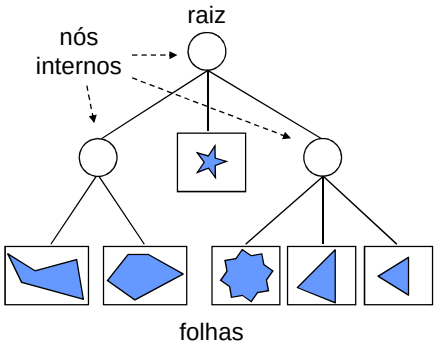




Volumes limitantes

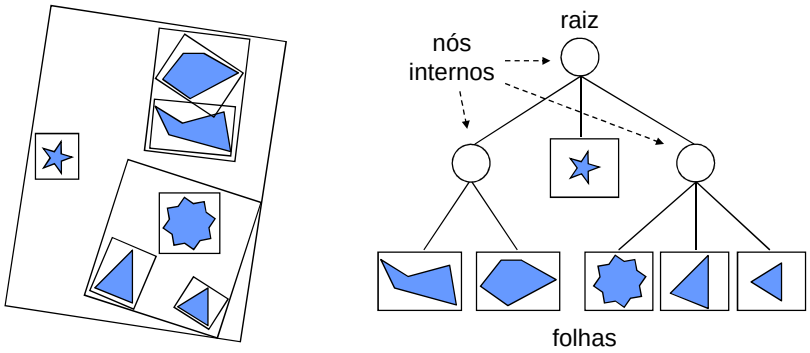
- Ilustração (em 2D) de uma hierarquia de AABBs.





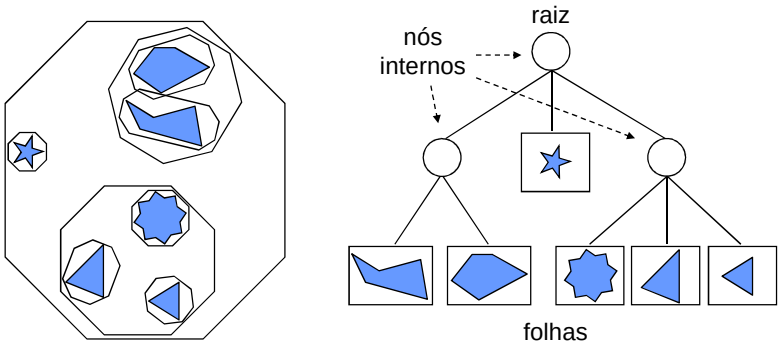
Volumes limitantes

- Ilustração (em 2D) de uma hierarquia de OBBs.



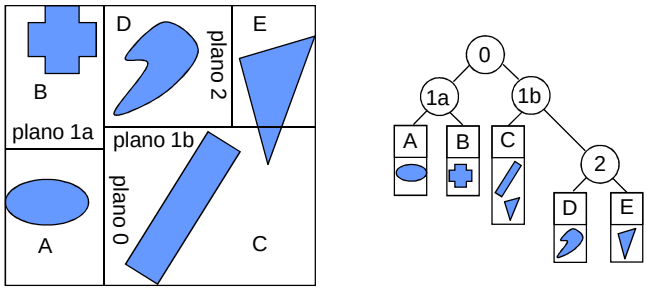
Volumes limitantes

- Ilustração (em 2D) de uma hierarquia de 8-DOPs.



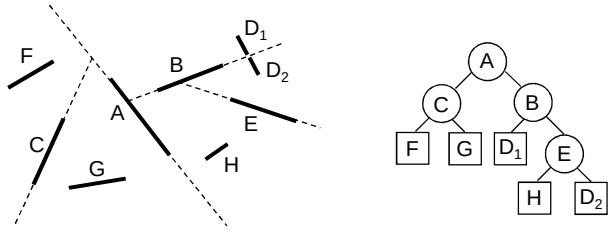
Subdivisões espaciais

- Árvores BSP (*Binary Space Partitioning*)
  - Podem ser definidas por planos de partição alinhados aos eixos (*axis-aligned*) ou alinhados aos planos das primitivas (*polygon-aligned*).
- *Axis-aligned*:



Subdivisões espaciais

- Árvores BSP (*Binary Space Partitioning*)
  - Podem ser definidas por planos de partição alinhados aos eixos (*axis-aligned*) ou alinhados aos planos das primitivas (*polygon-aligned*).
- *Polygon-aligned*:

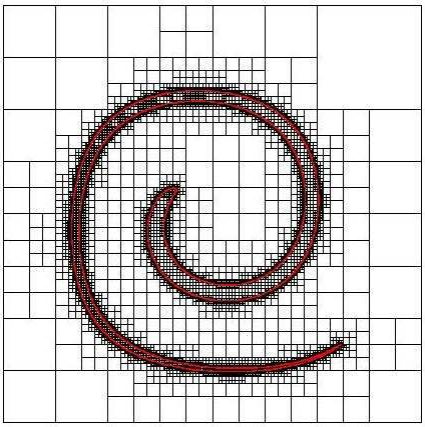


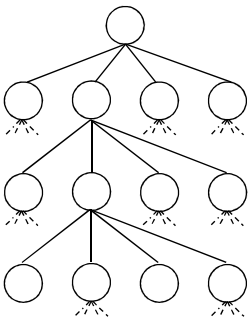
Subdivisões espaciais

- *Quadtree, octree*:
  - Similar à árvore BSP com planos de partição alinhados aos eixos, porém não produz uma árvore binária.
    - Caso 2D ou 2D½: *Quadtree*, 4 filhos por nó.
    - Caso 3D: *Octree*, 8 filhos por nó.
  - Particionamento regular
    - Consultas podem se tornar mais eficientes.

Subdivisões espaciais

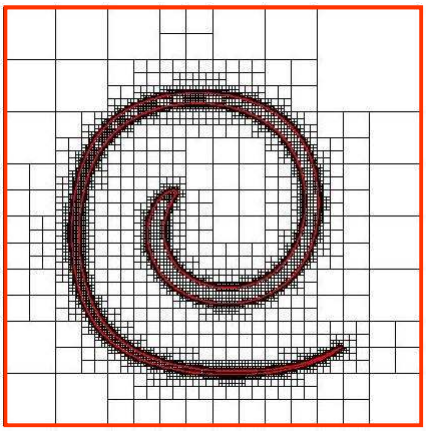
- Exemplo de uma *quadtree:*

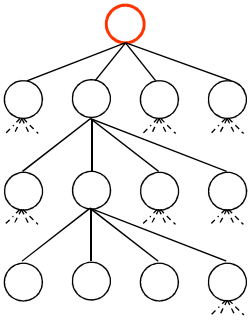




Subdivisões espaciais

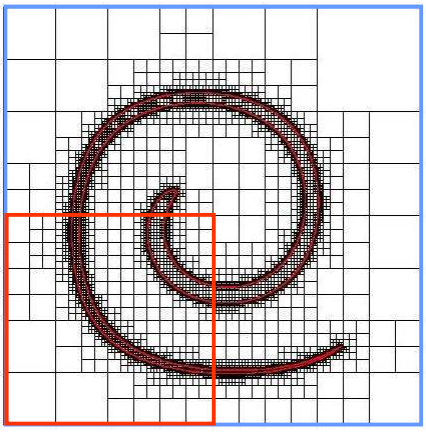
- Exemplo de uma *quadtree:*

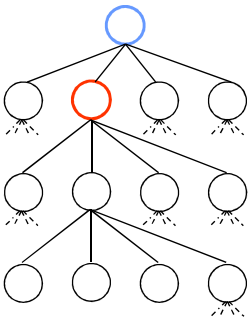




Subdivisões espaciais

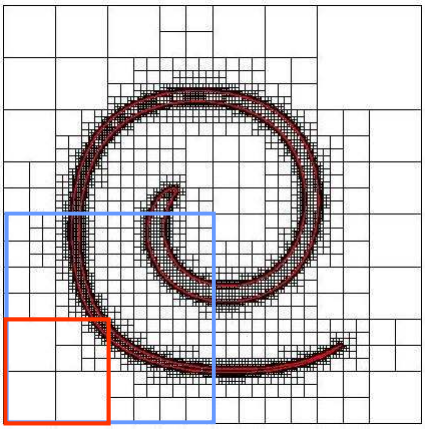
- Exemplo de uma *quadtree:*

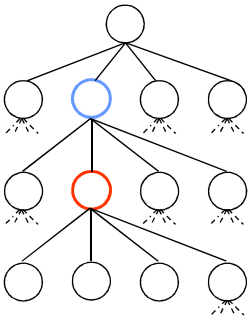




Subdivisões espaciais

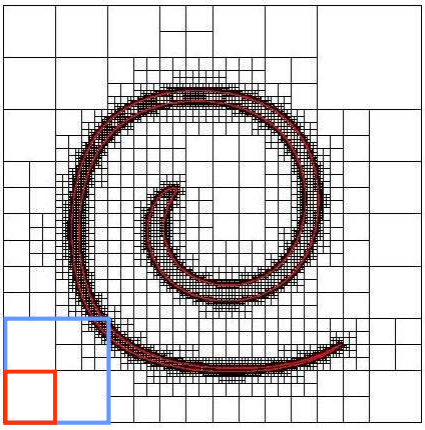
- Exemplo de uma *quadtree:*

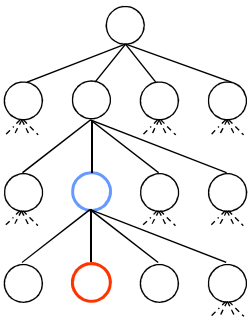




Subdivisões espaciais

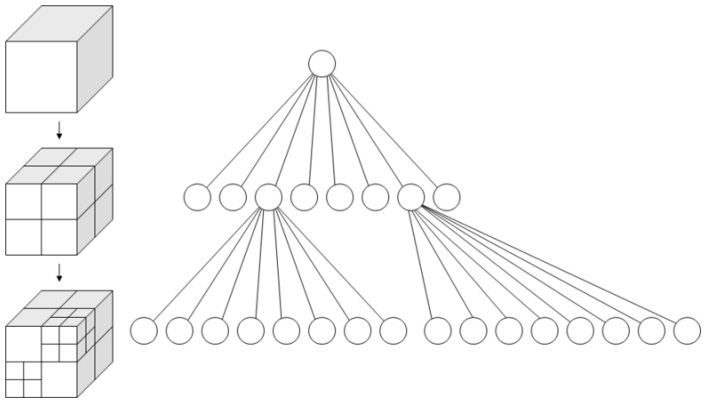
- Exemplo de uma *quadtree:*





Subdivisões espaciais

- Exemplo de uma *octree*:

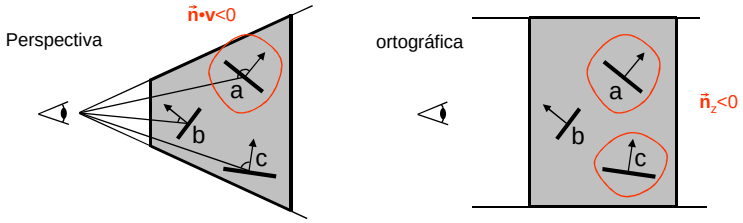


Backface culling

- Em objetos sólidos, os lados de cada faceta podem ser classificados em lado de dentro e lado de fora.
- Os lados são definidos pela direção do vetor normal segundo uma convenção adotada pela aplicação.
  - Em geral, o lado de fora é o lado para onde o vetor normal está apontando.
  - Os vetores normais de todas as faces devem seguir a mesma convenção em todo o modelo. Para isso, a ordem de definição dos vértices de cada triângulo deve ser ou no sentido horário (*clockwise*) ou no sentido anti-horário (*counterclockwise*).

Backface culling

- *Backface culling* é baseado na observação de que, em sólidos, faces cujo ângulo entre seu vetor normal ($\vec{n}$) e o vetor entre o observador até um ponto da face ($\vec{v}$) é maior que π/2 radianos, estão escondidas por faces da frente.
  - Em resumo, testar se $\vec{n} \bullet \vec{v} < 0$.
  - Em projeções ortográficas, basta verificar se $\vec{n}_z < 0$.

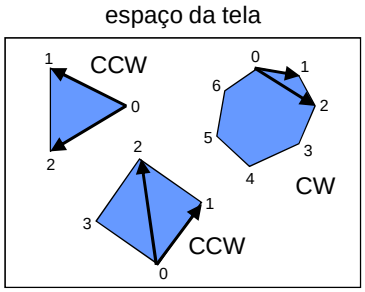


Backface culling

- As faces escondidas não contribuem para a formação da imagem final e podem ser removidas da cena para o atual ponto de vista.
  - Em geral, cerca de 50% das faces são removidas.
- Com exceção de sólidos convexos, não se sabe se as faces restantes estão ou não visíveis.
  - Em objetos convexos, as faces restantes com certeza estão visíveis se não houver oclusão por outros objetos.

Backface culling

- Atualmente é implementado em todas as placas de vídeo.
  - Teste é realizado no espaço da tela.
- Exemplo de estados do OpenGL relacionados a *backface culling*:
  - glEnable (GL_CULL_FACE);
  - glCullFace (GL_BACK);
  - glFrontFace (GL_CCW);

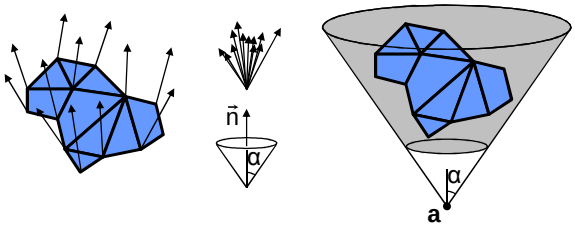


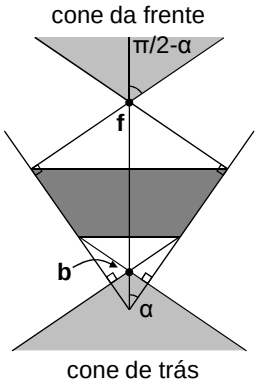
Clustered backface culling

- *Backface culling* é um algoritmo $O(n)$.
  - É possível melhorar esse desempenho?
- Em *clustered backface culling*, podemos decidir se um conjunto de polígonos pode ser classificado como escondido ou não em um único tese.
  - Complexidade sub-linear no número de faces.
- Baseado no conceito de "cone de normais" (*normal cone*).
  - Shirman, Leon A., and Salim S. Abi-Ezzi, "The Cone of Normals Technique for Fast Processing of Curved Patches," *Proceedings of Eurographics '93*, vol. 12, no. 3, pp. 261-272, 1993.

Clustered backface culling

- Exemplo usando cone de normais:



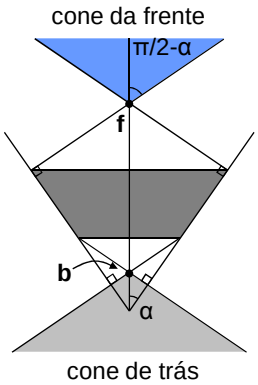


Clustered backface culling

- Dado um observador **e**, então **e** está no cone da frente se:

$$\vec{n} \bullet (\mathbf{e}-\mathbf{f}/||\mathbf{e}-\mathbf{f}||) \geq \cos(\pi/2-\alpha)=\sin(\alpha).$$

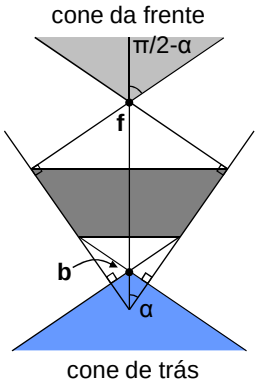
- Todas as faces entre **b** e **f** formam um ângulo menor que 90 graus com o vetor entre o observador e cada face.



Clustered backface culling

- Dado um observador **e**, então **e** está no cone de trás se:

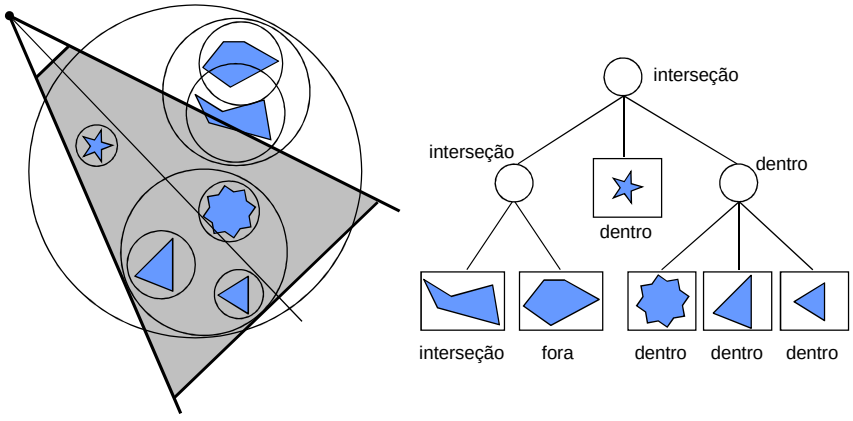
$$-\vec{n} \bullet (\mathbf{e}-\mathbf{b}/||\mathbf{e}-\mathbf{b}||) \geq \cos(\pi/2-\alpha)=\sin(\alpha).$$



- Todas as faces entre **b** e **f** formam um ângulo maior que 90 graus com o vetor entre o observador e cada face.

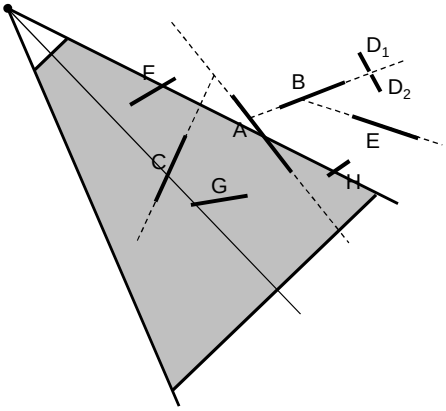
View frustum culling

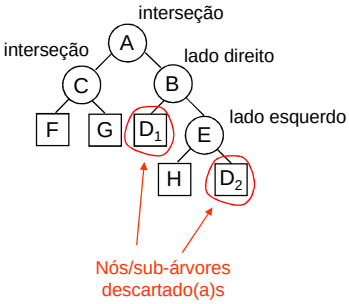
- Somente primitivas totalmente ou parcialmente contidas no volume de visão precisam ser enviadas à placa de vídeo.



View frustum culling

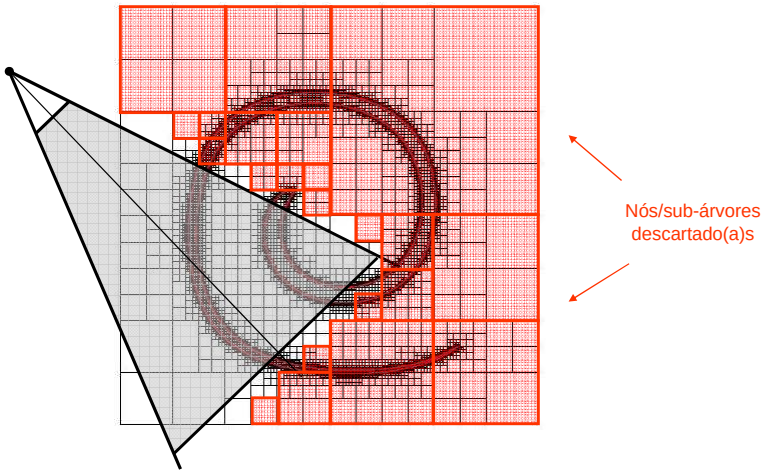
- *View frustum culling* usando uma árvore BSP:





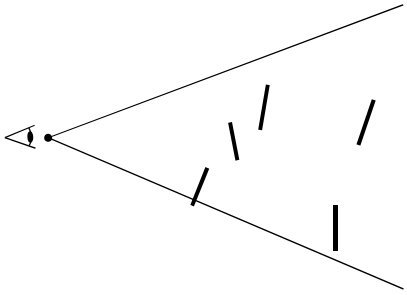
View frustum culling

- *View frustum culling* usando uma *quadtree*:



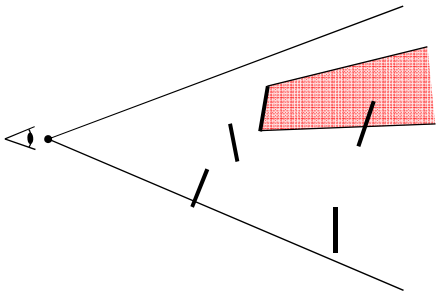
Occlusion culling

- Dado um ponto de vista, cada objeto visível produz um volume de oclusão, equivalente a um volume de sombra se o observador é considerado uma fonte de luz.
- Descartam-se os objetos totalmente contidos nos volumes de oclusão.



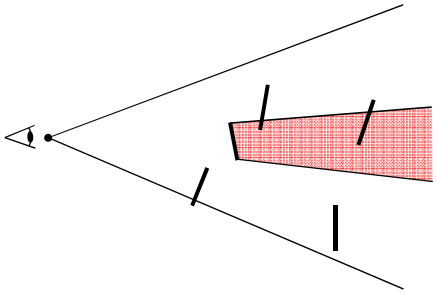
Occlusion culling

- Dado um ponto de vista, cada objeto visível produz um volume de oclusão, equivalente a um volume de sombra se o observador é considerado uma fonte de luz.
- Descartam-se os objetos totalmente contidos nos volumes de oclusão.



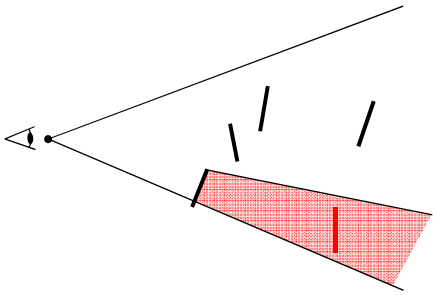
Occlusion culling

- Dado um ponto de vista, cada objeto visível produz um volume de oclusão, equivalente a um volume de sombra se o observador é considerado uma fonte de luz.
- Descartam-se os objetos totalmente contidos nos volumes de oclusão.



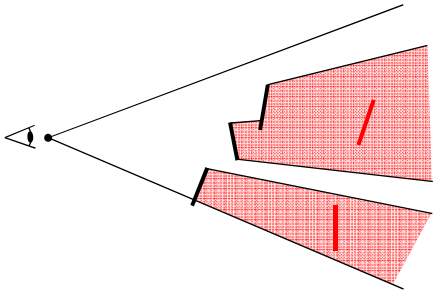
Occlusion culling

- Dado um ponto de vista, cada objeto visível produz um volume de oclusão, equivalente a um volume de sombra se o observador é considerado uma fonte de luz.
- Descartam-se os objetos totalmente contidos nos volumes de oclusão.



Occlusion culling

- Dado um ponto de vista, cada objeto visível produz um volume de oclusão, equivalente a um volume de sombra se o observador é considerado uma fonte de luz.
- Descartam-se os objetos totalmente contidos nos volumes de oclusão, ou totalmente contidos na união desses volumes.



Algoritmos de portais

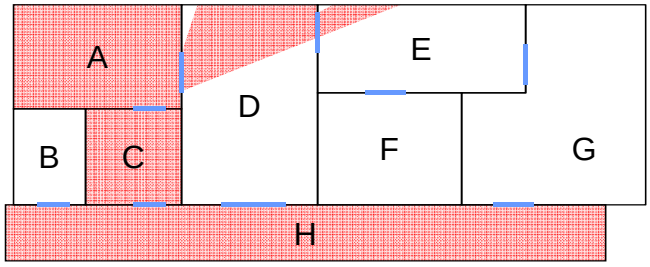
- *Portal culling* é um método de *occlusion culling* para modelos arquitetônicos.
  - Teller, Seth J., and Carlo H. Séquin, “Visibility Preprocessing For Interactive Walkthroughs,” *Computer Graphics (SIGGRAPH 94 Proceedings)*, pp. 443-450, July 1994.
- Em ambiente interiores, as paredes funcionam como grandes oclusores da cena.
- Cena é dividida em células e portais.
  - Células são as salas.
  - Portais são portas e janelas que conectam as salas.

Algoritmos de portais

- Divisão da cena em células e portais é geralmente feita à mão, por quem projetou o cenário.
- Pré-processamento:
  - Cada objeto de uma célula, incluindo as próprias paredes, são armazenados em uma estrutura de dados associada à célula.
  - Informações de células adjacentes e seus portais são armazenadas em um grafo de adjacência.
  - Pode-se pré-computar um PVS para cada célula.

Algoritmos de portais

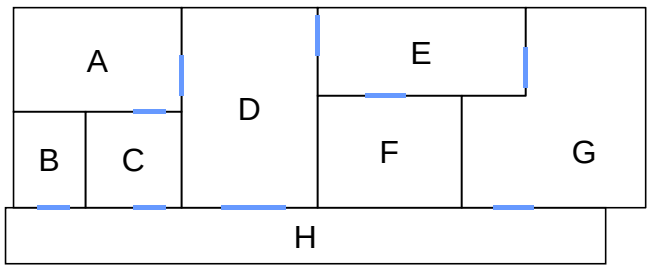
- Exemplo:
  - PVS da célula C contém todos os objetos que intersectam a região vermelha.

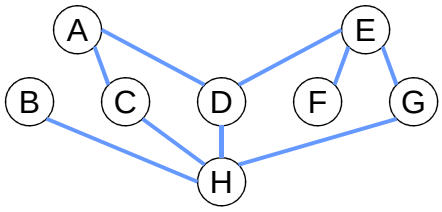


Algoritmos de portais

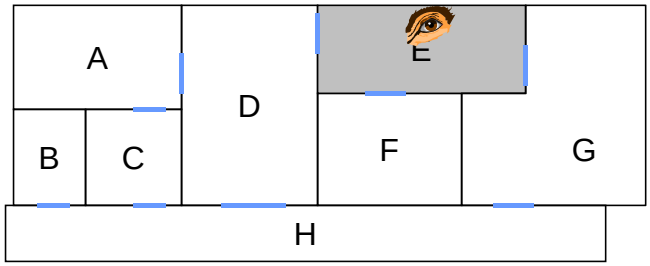
- Em tempo de execução (uma estratégia):
  1. Determina-se qual célula contém o observador.
  2. O PVS para tal célula é enviado à placa de vídeo.
- Em tempo de execução (outra estratégia):
  1. Determina-se qual célula contém o observador.
  2. O conteúdo da célula é enviado à placa de vídeo.
  3. Para cada célula adjacente no grafo de adjacências:
     - Existe uma linha de visão entre a célula atual e a célula adjacente? Se sim, volte para 2 e aplique o procedimento de forma recursiva para a célula adjacente.

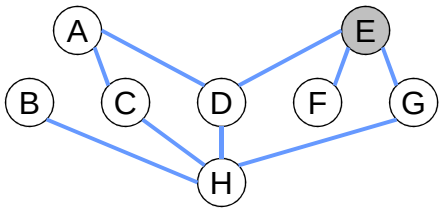
Algoritmos de portais



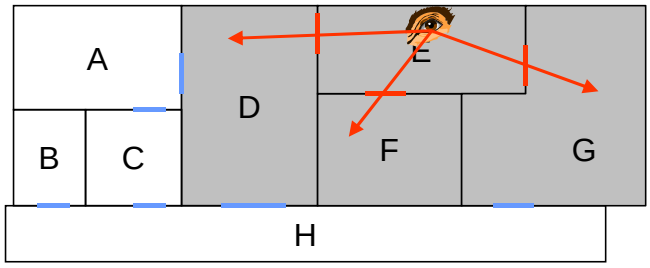


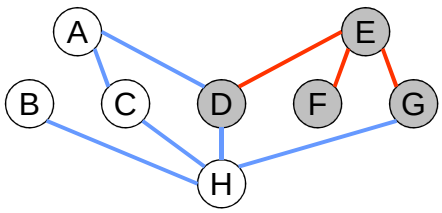
Algoritmos de portais



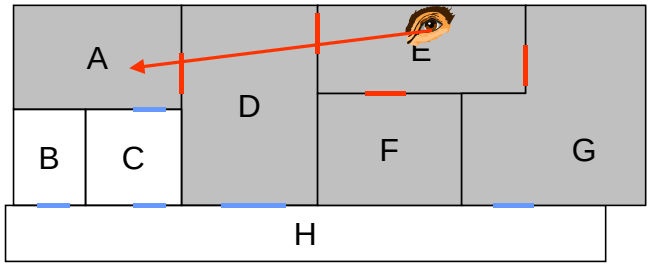


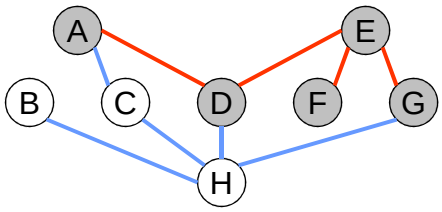
Algoritmos de portais



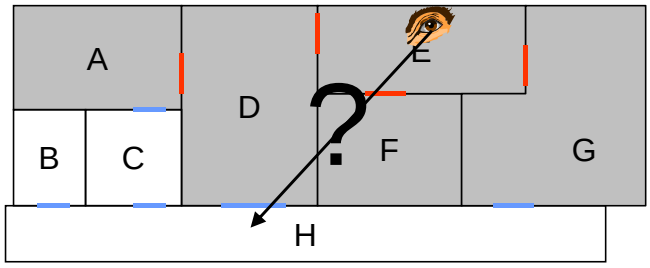


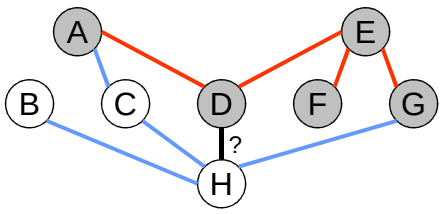
Algoritmos de portais



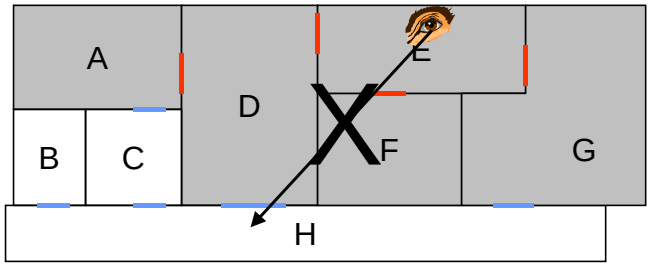


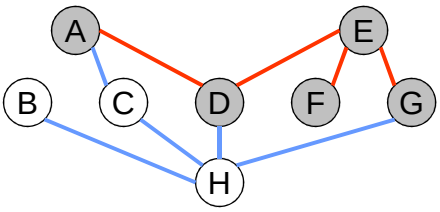
Algoritmos de portais





Algoritmos de portais





Algoritmos de portais

- PVS pode ser computado em tempo real.
  - Luebke, David P., and Chris Georges, “Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets,” *Proceedings 1995 Symposium on Interactive 3D Graphics*, pp. 105-106, April 1995.
- Algoritmo:
  1. Localize a célula **V** onde está o observador.
  2. Inicialize uma região limitante **P** ao retângulo da tela.
  3. Desenhe a geometria da célula **V** usando *view frustum culling* para o volume definido pelo observador e o retângulo *P* (inicialmente a tela toda).

Algoritmos de portais

- Algoritmo (continuação):
  4. Para cada portal de **V**, projete o portal na tela e ache o AABB correspondente da projeção. Calcule a interseção lógica de **P** e o AABB.
  5. Para cada interseção:
     - Se for vazia, descarte a célula conectada a **V** através desse portal.
     - Se não for vazia, então a célula conectada a **V** através desse portal pode estar visível. Desse modo, volte ao passo 3 com **P** sendo o resultado da interseção.