IA725 – Computação Gráfica I

Visibilidade

24/04/2008

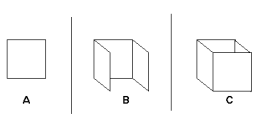
Shirley, capítulo 8

Durand, capítulos 7 a 11

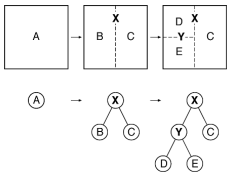
(http://people.csail.mit.edu/fredo/THESE/vo.pdf)

Visão geral

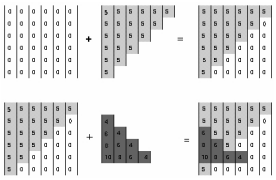
- Definição e aplicações de visibilidade em Computação Gráfica.
- Remoção de superfícies escondidas.
  - Algoritmo do pintor.
  - Árvore BSP (*Binary Space Partitioning*).
  - *Buffer* de profundidade (*Z-buffer*).



Algoritmo do pintor



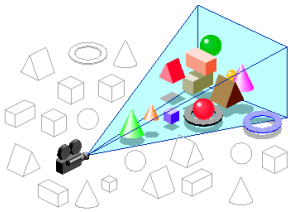
Árvore BSP

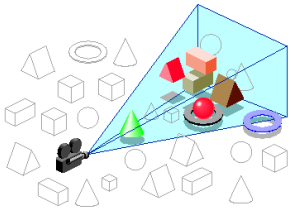


Z-buffer

Introdução

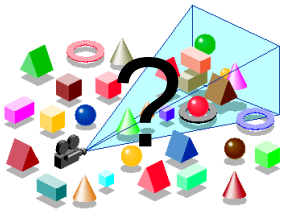
- Através de nosso sistema visual, percebemos que:
  - Processamos apenas as informações visuais dentro de nosso campo de visão.
  - Objetos opacos mais próximos do observador escondem os objetos mais distantes (relação de oclusão).





Introdução

- No processo de *rendering*, o computador não sabe:
  - O que precisa ser processado e o que pode ser descartado durante a geração da imagem.
  - Quais as relações de visibilidade na cena.
- Sem algoritmos de visibilidade, a cena inteira é projetada e preenchida, em ordem arbitrária.
- Problemas resultantes:
  - Imagem incorreta.
  - Baixa eficiência.

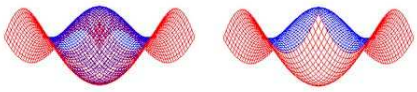


Introdução

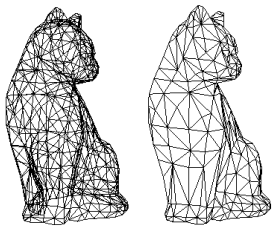
- Imagem incorreta
  - Objetos que estão mais próximos do plano de projeção devem esconder os objetos mais distantes.
  - Objetos fora do volume de visão não devem contribuir para a formação da imagem.
- Baixa eficiência
  - Em cenas grandes, apenas uma fração é visível para a maioria dos pontos de vista. Processar a parte não visível resulta em desperdício de processamento.
  - Utilizar placas gráficas mais eficientes não basta.

Introdução

- Aplicações de visibilidade em Computação Gráfica:
  - **Remoção de superfícies escondidas.**
  - Descarte de primitivas (*culling*).
  - Geração de sombras.
  - Iluminação global.



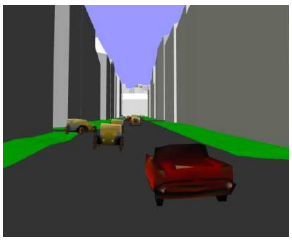
Sem remoção Com remoção



Sem remoção Com remoção

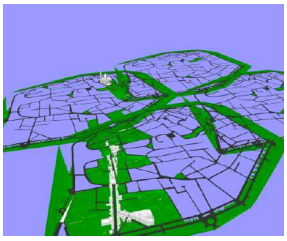
Introdução

- Aplicações de visibilidade em Computação Gráfica:
  - Remoção de superfícies escondidas.
  - **Descarte de primitivas (*culling*).**
  - Geração de sombras.
  - Iluminação global.



Cena vista pelo observador

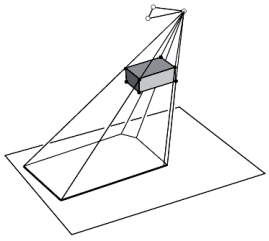
Cena completa (2,6M polígonos)



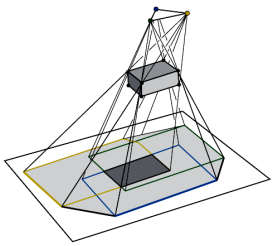
Após o descarte

Introdução

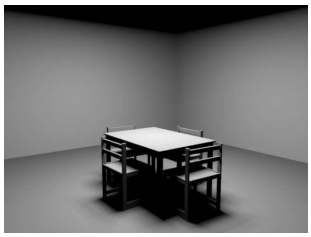
- Aplicações de visibilidade em Computação Gráfica:
  - Remoção de superfícies escondidas.
  - Descarte de primitivas (*culling*).
  - **Geração de sombras.**
  - Iluminação global.



Visibilidade com relação a um vértice da fonte de luz



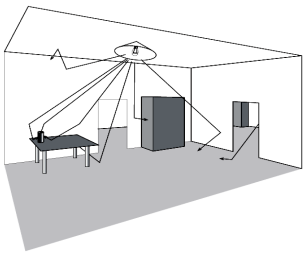
Umbra (cinza escuro) e penumbra (cinza claro)



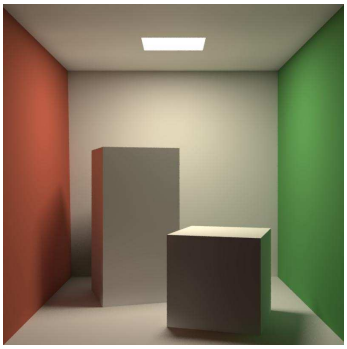
Exemplo

Introdução

- Aplicações de visibilidade em Computação Gráfica:
  - Remoção de superfícies escondidas.
  - Descarte de primitivas (*culling*).
  - Geração de sombras.
  - **Iluminação global.**



Iluminação direta e indireta



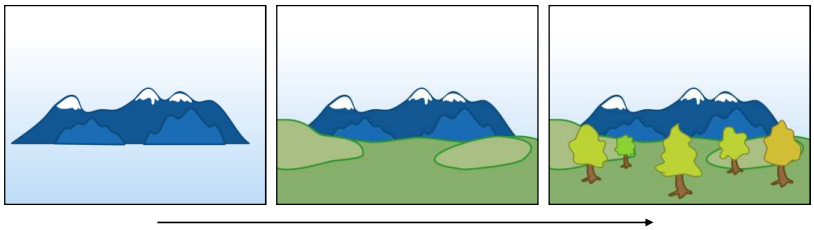
Exemplo (*Cornell Box*)

Introdução

- Neste curso, analisaremos os principais algoritmos de remoção de superfícies escondidas (hoje) e algoritmos de *culling* (próxima aula).
  - Espaço do objeto:
    - Algoritmo do pintor e árvore BSP.
  - Espaço da imagem:
    - *Buffer* de profundidade (*Z-buffer*).
  - *Back-face culling*, *view-frustum culling* e *occlusion culling*, portais.

Algoritmo do pintor

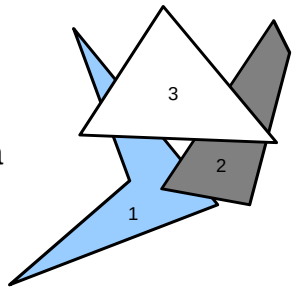
- Exibição de primitivas usando lista de prioridades.
- Objetos são desenhados na ordem em que um pintor de tela a óleo pintaria os objetos em um quadro.
- Objetos mais próximos do observador sobrepõem os mais distantes, preservando assim as relações de oclusão.



Ordem de desenho

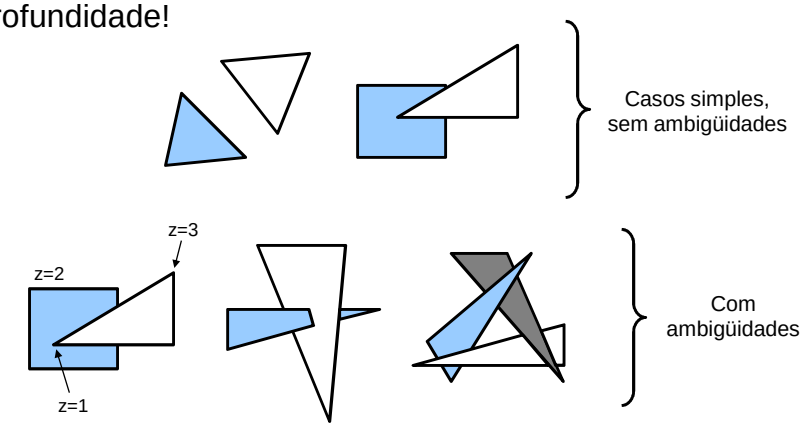
Algoritmo do pintor

- A cena é desenhada numa ordem de profundidade de trás para frente com relação ao plano de projeção.
- Primitivas podem ser ordenadas em relação à coordenada z no espaço da câmera.
- Desempenho Ω(n log n).
- Uma vez que a primitiva mais próxima é desenhada por último, ela estará no topo e, portanto, será visualizada corretamente.



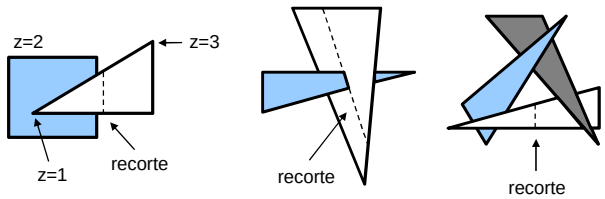
Algoritmo do pintor

- Podem surgir ambigüidades na ordenação por profundidade!



Algoritmo do pintor

- Ambigüidades podem ser resolvidas se as primitivas forem recortadas em partes menores.
  - Pode gerar até $O(n^2)$ primitivas.



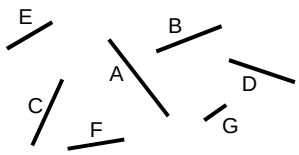
- M.E. Newell, R.G. Newell, and T.L. Sancha. A solution to the hidden surface problem. In *Proceedings of the ACM Annual Conference*, volume I, pages 443-450, Boston, Massachusetts, August 1972.

Árvore BSP

- Considere as seguintes observações:
  - Cada face divide o espaço em duas partições ou *half-spaces* ($ax+by+cz+d<0$ e $ax+by+cz+d>0$).
  - Intuitivamente, os objetos contidos na mesma partição que contém o observador estão "mais próximos" do observador que aqueles contidos na outra partição.
- Suponha que seja construída uma árvore dessas partições. Tal árvore de particionamento espacial é chamada de árvore BSP (*Binary Space Partitioning*).
- H. Fuchs, Z.M. Kedem, and B.F. Naylor. On visible surface generation by a priori tree structures. In *Computer Graphics (SIGGRAPH '80 Proceedings*), volume 14(3), pages 124-133, July 1980.

Árvore BSP - Geração

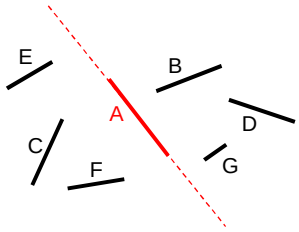
- Procedimento recursivo para gerar uma árvore BSP:
  - Selecione uma primitiva e determine seu hiperplano de partição.
  - Particione todas as outras primitivas com relação aos lados do hiperplano de partição em que elas se encontram.
  - Repita o procedimento para o conteúdo de cada partição.



Exemplo em 2D
(hiperplanos = linhas)

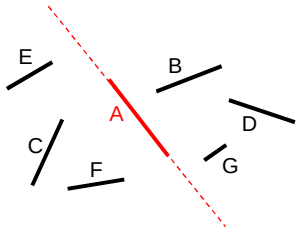
Árvore BSP - Geração

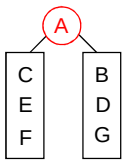
- Procedimento recursivo para gerar uma árvore BSP:
  - Selecione uma primitiva e determine seu hiperplano de partição.
  - Particione todas as outras primitivas com relação aos lados do hiperplano de partição em que elas se encontram.
  - Repita o procedimento para o conteúdo de cada partição.



Árvore BSP - Geração

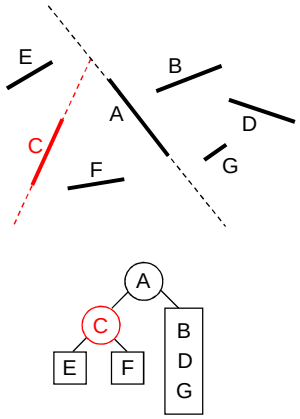
- Procedimento recursivo para gerar uma árvore BSP:
  - Selecione uma primitiva e determine seu hiperplano de partição.
  - Particione todas as outras primitivas com relação aos lados do hiperplano de partição em que elas se encontram.
  - Repita o procedimento para o conteúdo de cada partição.





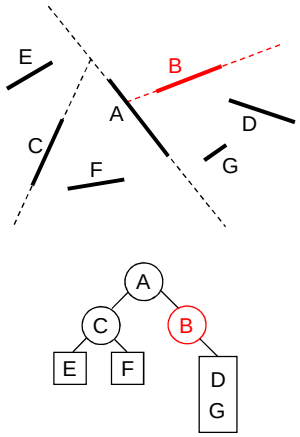
Árvore BSP - Geração

- Procedimento recursivo para gerar uma árvore BSP:
  - Selecione uma primitiva e determine seu hiperplano de partição.
  - Particione todas as outras primitivas com relação aos lados do hiperplano de partição em que elas se encontram.
  - Repita o procedimento para o conteúdo de cada partição.



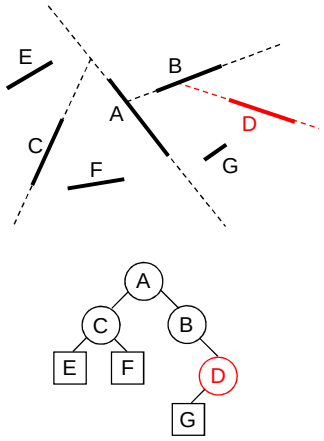
Árvore BSP - Geração

- Procedimento recursivo para gerar uma árvore BSP:
  - Selecione uma primitiva e determine seu hiperplano de partição.
  - Particione todas as outras primitivas com relação aos lados do hiperplano de partição em que elas se encontram.
  - Repita o procedimento para o conteúdo de cada partição.



Árvore BSP - Geração

- Procedimento recursivo para gerar uma árvore BSP:
  - Selecione uma primitiva e determine seu hiperplano de partição.
  - Particione todas as outras primitivas com relação aos lados do hiperplano de partição em que elas se encontram.
  - Repita o procedimento para o conteúdo de cada partição.



Árvore BSP - Geração

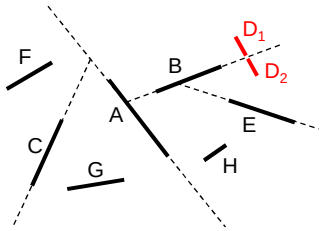
- O particionamento exige que todas as primitivas do conjunto ativo sejam testadas para saber em que lado da partição elas estão. Consideramos três possibilidades.
  - No lado positivo.
  - No lado negativo.
  - Cruzando o hiperplano de partição.
    - Neste caso, a primitiva em questão deve ser recortada pelo hiperplano de partição de modo a obter duas metades: a que está no lado positivo e a que está no lado negativo.

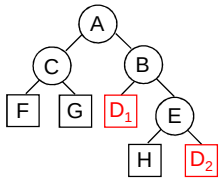
Árvore BSP - Geração

```
BSPtree makeBSP( L: list of polygons )
{
  if L is empty {
      return the empty tree
  }
  Choose a polygon P from L to serve as root
  Split all polygons in L according to P
  return new TreeNode(
      P,
      makeBSP( polygons on negative side of P ),
      makeBSP( polygons on positive side of P ))
  }
}
```

Árvore BSP - Geração

- Exemplo: Primitiva D cruzando o plano de partição B.





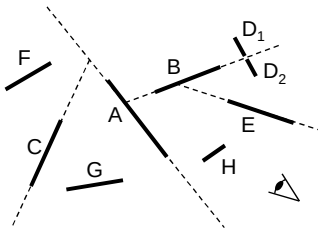
- Recortes podem gerar até $O(n^3)$ novas primitivas.
- Heurísticas são geralmente utilizadas para minimizar número de recortes.

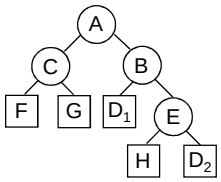
Árvore BSP - Percurso

- A árvore BSP contém implicitamente a informação de profundidade relativa entre as primitivas.
- Através de uma busca em profundidade (*depth-first search*), é possível determinar, para qualquer ponto de vista, a ordem correta de exibição das primitivas com o algoritmo do pintor.
  - Não há ambigüidades de sobreposição de desenho das primitivas, pois todas já foram resolvidas durante a construção da árvore.

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  1. Classifique a posição do observador com relação ao lado do hiperplano de partição.
  2. Chame esse procedimento para o nó contido na partição oposta.
  3. Desenhe a primitiva atual desse hiperplano de partição.
  4. Chame esse procedimento para o nó contido no mesmo lado.



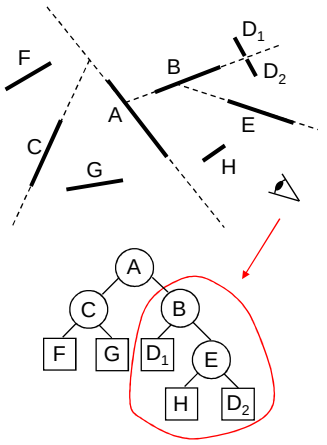


Árvore BSP - Percurso

```
showBSP( v: Viewer, T: BSPtree )
{
  if T is empty then return
  P := root of T
  if viewer is in front of P
  {
      showBSP( back subtree of T )
      draw P
      showBSP( front subtree of T )
  } else {
      showBSP( front subtree of T )
      draw P
      showBSP( back subtree of T )
  }
}
```

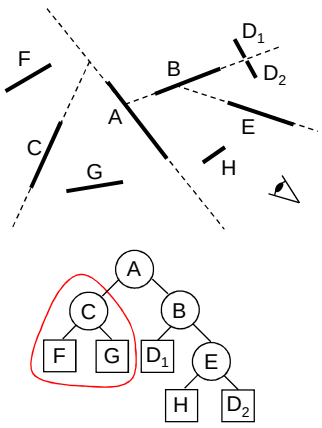
Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.



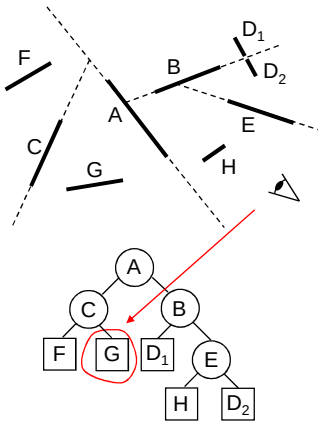
Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.



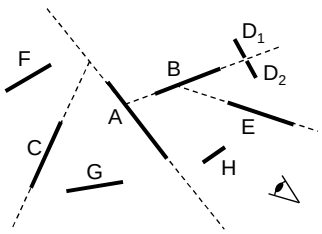
Árvore BSP - Percurso

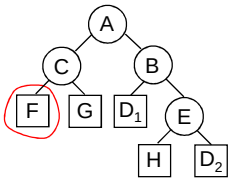
- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.



Árvore BSP - Percurso

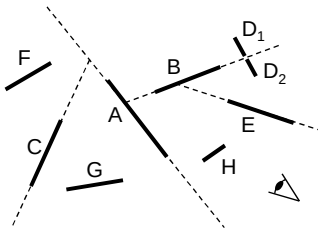
- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

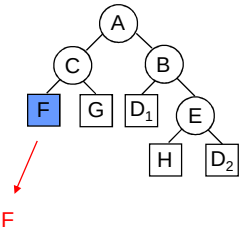




Árvore BSP - Percurso

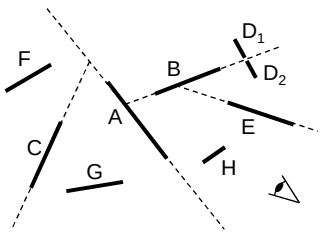
- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

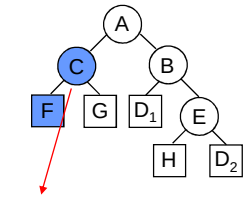




Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

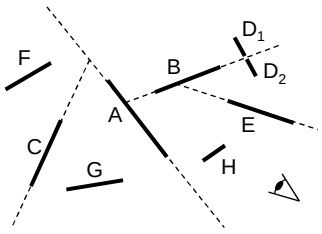


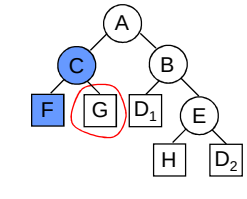


F, C

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

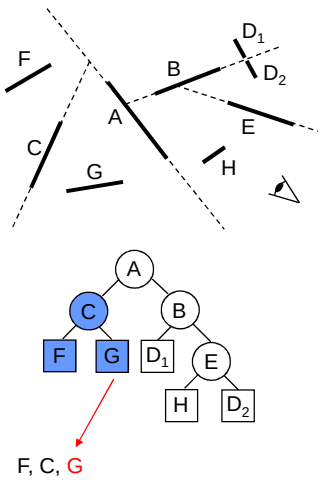




F, C

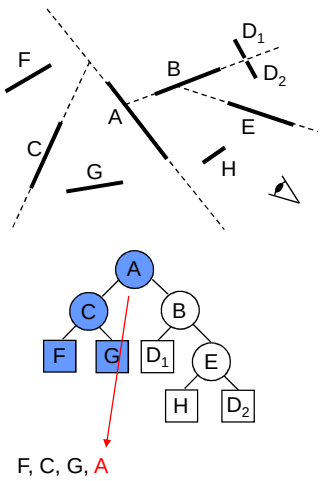
Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.



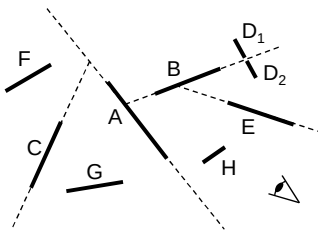
Árvore BSP - Percurso

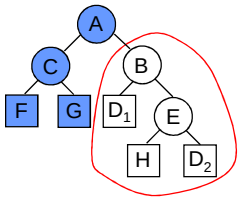
- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.



Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

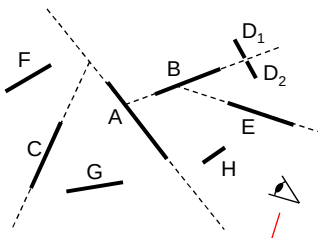


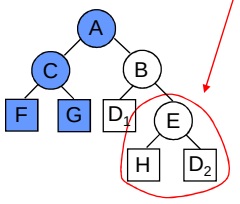


F, C, G, A

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

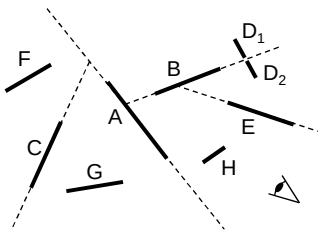


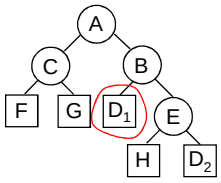


F, C, G, A

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

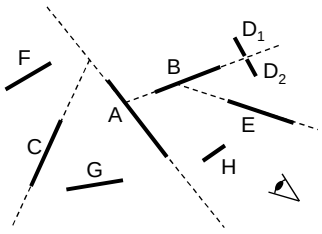


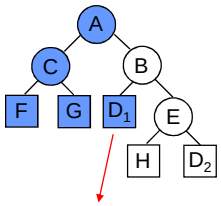


F, C, G, A

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

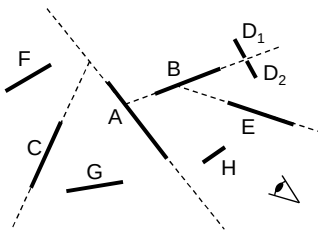


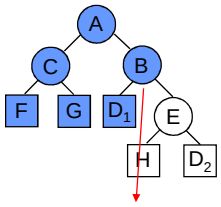


F, C, G, A, D_1

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

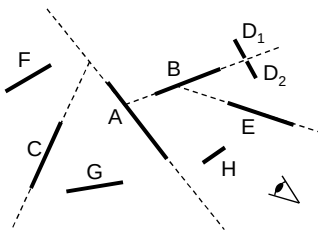


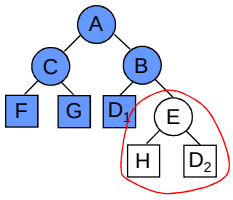


F, C, G, A, D_1, B

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

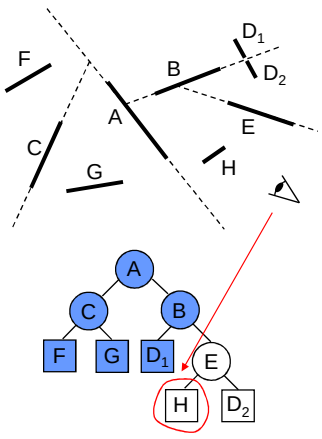




F, C, G, A, D_1, B

Árvore BSP - Percurso

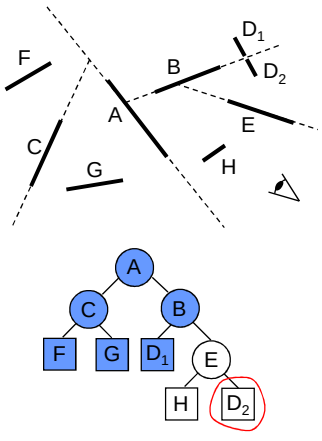
- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.



F, C, G, A, D_1, B

Árvore BSP - Percurso

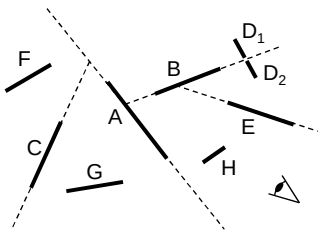
- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

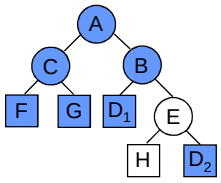


F, C, G, A, D_1, B

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

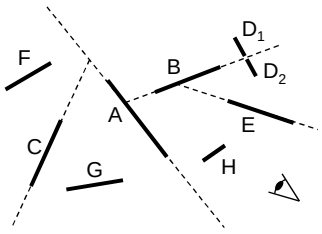


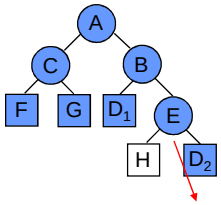


F, C, G, A, D_1, B, D_2

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

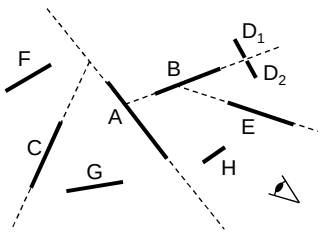


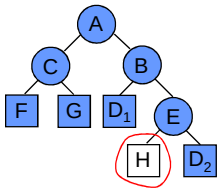


F, C, G, A, D_1, B, D_2, E

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

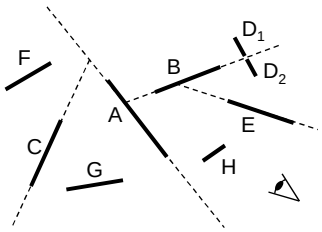


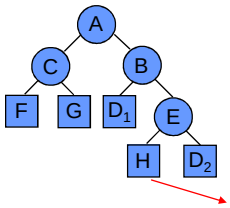


F, C, G, A, D_1, B, D_2, E

Árvore BSP - Percurso

- Percorre-se a árvore BSP pelo seguinte procedimento recursivo, a partir do nó raiz :
  - Classifique a posição do observador com relação ao lado do hiperplano de partição.
  - Chame esse procedimento para o nó contido na partição oposta.
  - Desenhe a primitiva atual desse hiperplano de partição.
  - Chame esse procedimento para o nó contido no mesmo lado.

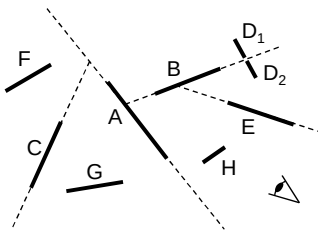


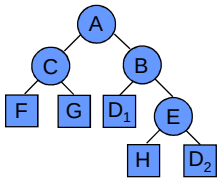


F, C, G, A, D_1, B, D_2, E, H

Árvore BSP - Percurso

- Em cada partição, primeiro desenhamos as primitivas mais distantes do hiperplano atual, então a primitiva do hiperplano atual, e então as primitivas mais próximas.
- Uma vez que cada partição é uma região convexa, o resultado é uma ordenação de trás para frente das primitivas em tempo *O*(n).





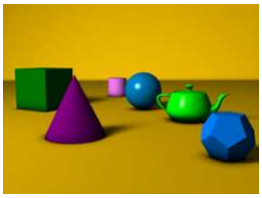
F, C, G, A, D_1, B, D_2, E, H

BSP – Considerações finais

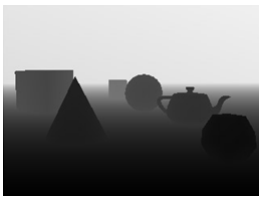
- Prós:
  - Percurso simples e eficiente.
  - Visibilidade é resolvida no espaço do objeto, portanto independe da resolução da imagem.
- Contras:
  - Requer um pré-processamento considerável.
  - Primitivas devem ser fáceis de serem subdivididas pelos hiperplanos.
  - Subdivisão pode gerar até *O*(n^3) novas primitivas.

Z-buffer

- Desenvolvido para resolver visibilidade de superfícies curvas.
  - Edwin E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.D. thesis, University of Utah, December 1974.
- Um valor de profundidade (também chamado de valor z) é armazenado para cada *pixel* da imagem em um *buffer* de profundidade (*Z-buffer*).



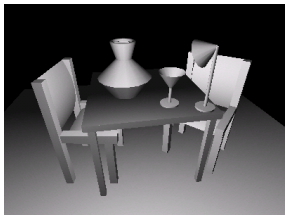
Buffer de cor



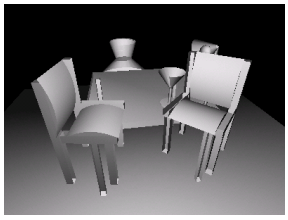
Buffer de profundidade

Z-buffer

- Assim que cada primitiva é rasterizada, a profundidade de cada *pixel* que o cobre na imagem é calculada e comparada com o valor de profundidade atual.
- O *pixel* é desenhado no *buffer* de cor somente se o valor de profundidade da primitiva está mais próximo do observador.



Cena visualizada com *Z-buffering*

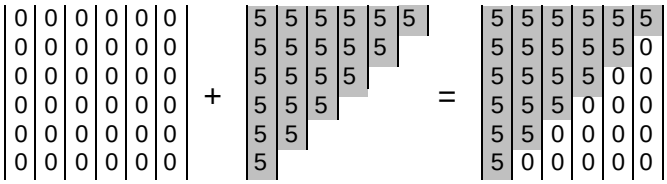


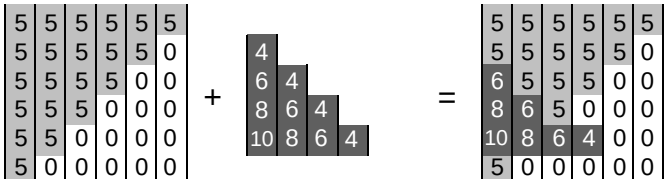
Cena visualizada sem *Z-buffering*

Z-buffer - Algoritmo

```
for each pixel pi
{
   Z-buffer[ pi ] = FAR
   color-buffer[ pi ] = BACKGROUND_COLOR
}
for each polygon P
{
   for each pixel pi in the projection of P
   {
        Compute depth z and shade s of P at pi
        if z < Z-buffer[ pi ]
        {
               Z-buffer[ pi ] = z
               color-buffer[ pi ] = s
        }
   }
}
```

Z-buffer - Exemplo





Z-buffer

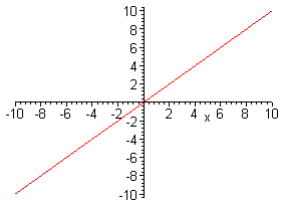
- Quanto maior o número de *bits* por *pixel*, maior a precisão.
  - Em geral, 24, 16 ou 32 *bits* em *hardware*.
- Após a divisão pela coordenada homogênea (z/w), a precisão da profundidade deixa de ser uniforme ao longo do volume de visão:
  - Pontos próximos têm maior precisão que pontos distantes.
  - Problema para cenas grandes, pois erros de quantização (*Z-fighting*) podem ocorrer.
- Uma possível solução é armazenar o valor w. Tal técnica é chamada de *W-buffering*.

Z-buffer e OpenGL

- OpenGL provê suporte a *Z-buffering*.
- No GLUT, deve ser criado com glutInitDisplayMode (GLUT_RGB | **GLUT_DEPTH | …**);
- Testes e escritas no *Z-buffer* são habilitados de forma independente do acesso ao *color buffer*, pelos seguintes estados:
  - glEnable(GL_DEPTH_TEST);
  - glDepthFunc(GL_LESS_OR_EQUAL);
  - glDepthMask(GL_TRUE);
  - glClear(GL_DEPTH_BUFFER_BIT);
  - glClearDepth (1.0f);

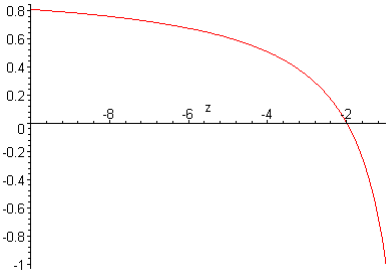
Z-buffer e OpenGL

- Assuma w=1 e glFrustum(-1.0, 1.0, -1.0, 1.0, 1.0, 200.0).
- Coordenada x no espaço da câmera versus coordenada x após projeção e divisão pela coordenada homogênea:



plot((2.0*1.0/(1.0-(-1.0))*x)/(1.0), x=-10..10);

- Coordenada z no espaço da câmera versus coordenada z após projeção e divisão pela coordenada homogênea:



plot((((-200-1)/(200-1))*z+(-2.0*200*1.0/(200-1)))/(-z), z=-1..-10);

Z-buffer - Considerações finais

- Prós:
  - Fácil de implementar.
  - Algoritmo incremental.
  - Implementado em todas as placas gráficas atuais.
- Contras:
  - Requer mais memória do *frame buffer*.
  - Válido somente para a imagem gerada.
  - Erros de quantização (distribuição dos valores Z não é linear ao longo do volume de visão).