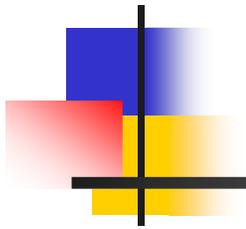
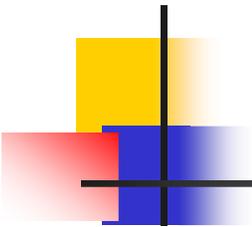


Eliminação de Superfícies Escondidas



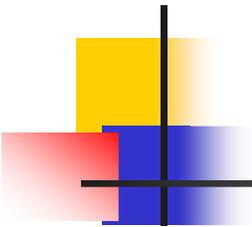
**Thanks to
Isabel Harb Manssour
Marcelo Cohen**

2012



Roteiro

1. Introdução
2. Remoção de Faces Traseiras
3. Algoritmo do Pintor
4. Algoritmo *Z-Buffer*
5. Árvores BSP

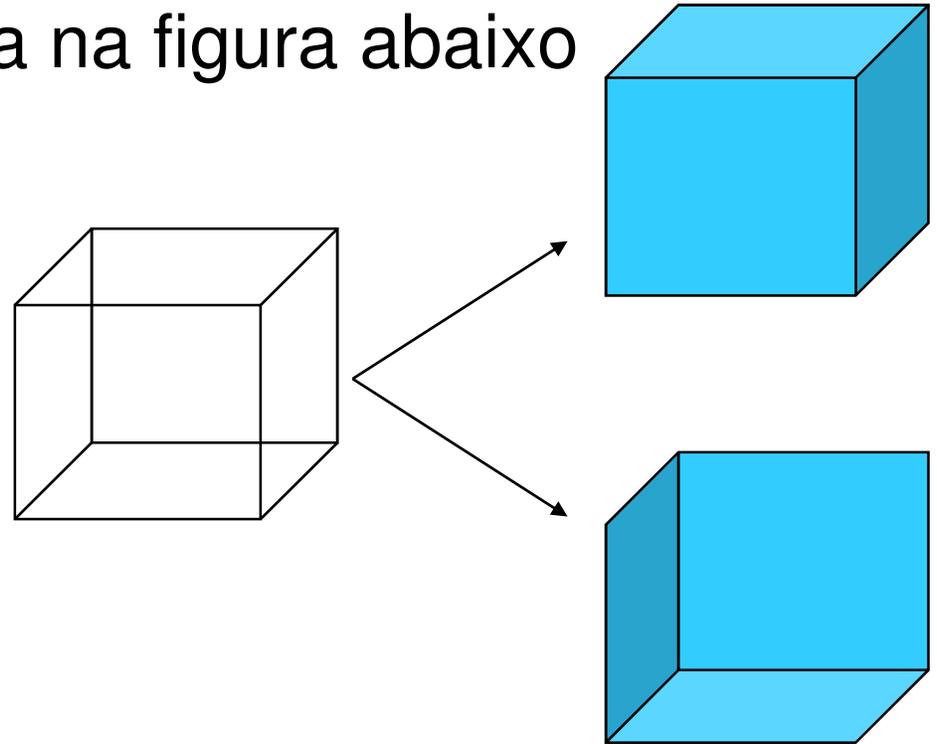


Introdução

- Eliminação de superfícies escondidas
 - Também conhecido por Remoção de Elementos Ocultos
 - Já foi um dos problemas mais difíceis da Computação Gráfica
 - Algoritmos
 - Usados para determinar as linhas, arestas, superfícies ou volumes que são visíveis ou não para um observador localizado em um ponto específico no espaço
 - Atualmente, a maior parte é implementada nas placas gráficas

Introdução

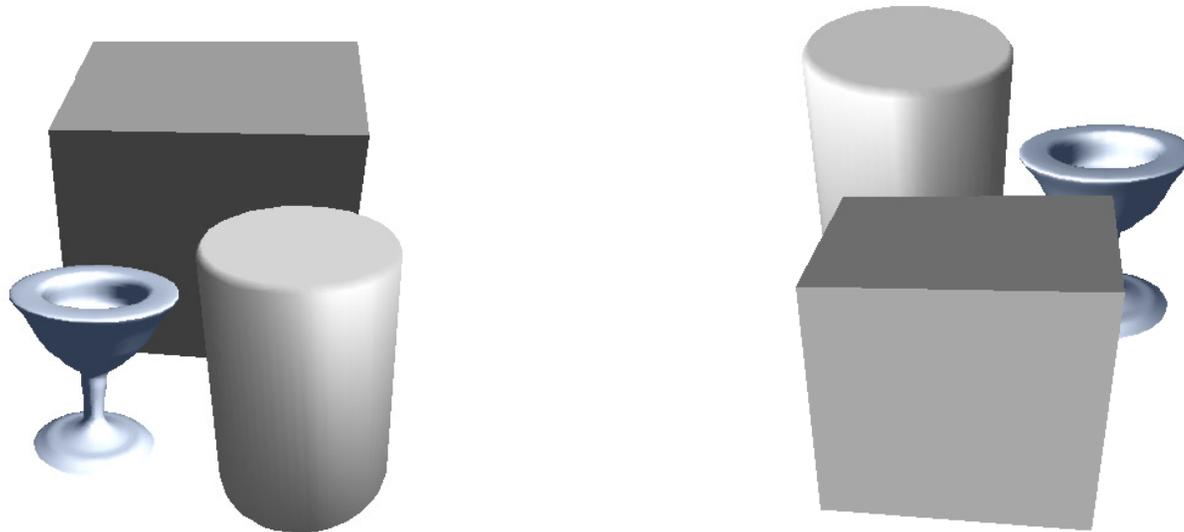
- Uma das necessidades de eliminar superfícies escondidas está ilustrada na figura abaixo



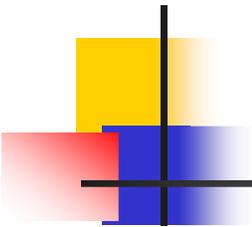
- O cubo pode ser interpretado tanto como uma vista superior/esquerda ou inferior/direita
- Esta ambigüidade pode ser eliminada removendo-se as linhas ou superfícies que são invisíveis a partir das duas visões

Introdução

- Alguns objetos também podem ser ocultos por outros objetos, como mostram as figuras abaixo

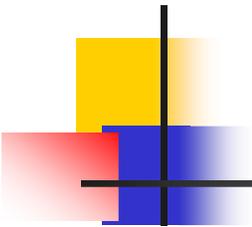


- Portanto, as faces/linhas são ocultas:
 - Pelo próprio objeto
 - Por outros objetos



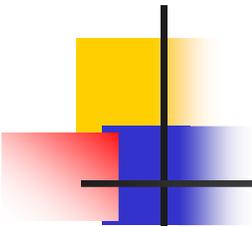
Introdução

- A complexidade do problema das superfícies escondidas resultou em um grande número de soluções
 - Não se pode afirmar que uma técnica é melhor do que a outra
 - Depende da aplicação (complexidade da cena, tipos de objetos, equipamento disponível, entre outros)
 - Alguns algoritmos fornecem soluções mais rápidas
 - Outros necessitam de muita memória
 - Outros são mais lentos, mas fornecem soluções realísticas detalhadas (incluem sombras, transparência, etc)



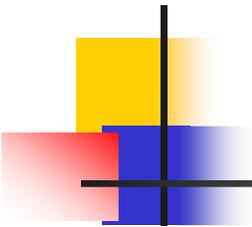
Introdução

- Alguns algoritmos podem envolver uma ordenação
 - Distância do observador ao volume, superfície ou aresta
 - Parte do pressuposto que quanto mais longe um objeto está do observador, mais chances ele tem de estar totalmente ou parcialmente encoberto por um objeto mais próximo do observador
 - Neste caso, a eficiência do algoritmo de eliminação de superfícies escondidas depende da eficiência do processo de ordenação



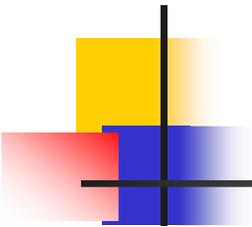
Introdução

- De acordo com a abordagem adotada, os algoritmos de eliminação de superfícies escondidas podem ser classificados em:
 - Métodos que trabalham no **espaço de objeto**
 - Métodos que trabalham no **espaço de imagem**



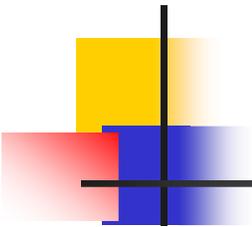
Introdução

- Métodos que trabalham no espaço de objeto
 - Implementados no sistema de coordenadas no qual o objeto é descrito
 - Determinam as porções visíveis dos objetos pela comparação entre eles
 - Para cada objeto:
 - Determinam as porções do objeto que não estão ocultas por quaisquer outros presentes na cena
 - Exibem as porções visíveis dos objetos



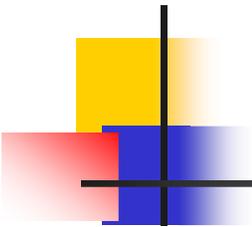
Introdução

- Métodos que trabalham no espaço de imagem
 - Implementados no sistema de coordenadas de tela (SRT) no qual os objetos são visualizados
 - Para cada **pixel** da tela
 - Determinam qual dentre n objetos está visível
- Obs.: Alguns algoritmos combinam as duas abordagens.



Roteiro

- ✓ Introdução
 - ✓ Pense numa solução...



Roteiro

✓ Introdução

✓ Pense numa solução...

2. Remoção de Faces Traseiras

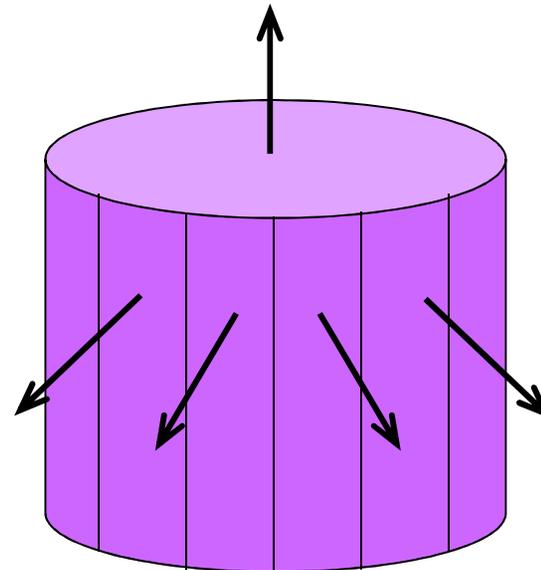
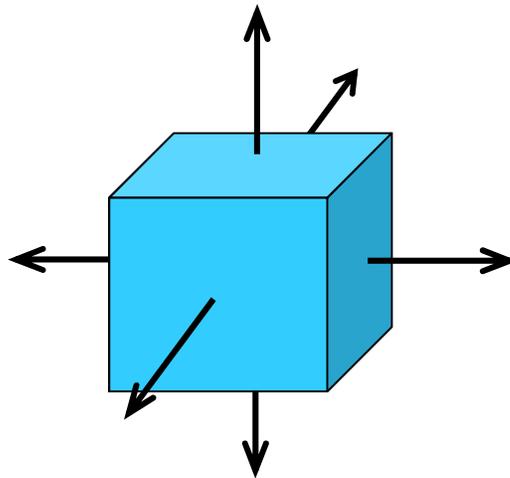
3. Algoritmo do Pintor

4. Algoritmo *Z-Buffer*

5. Árvores BSP

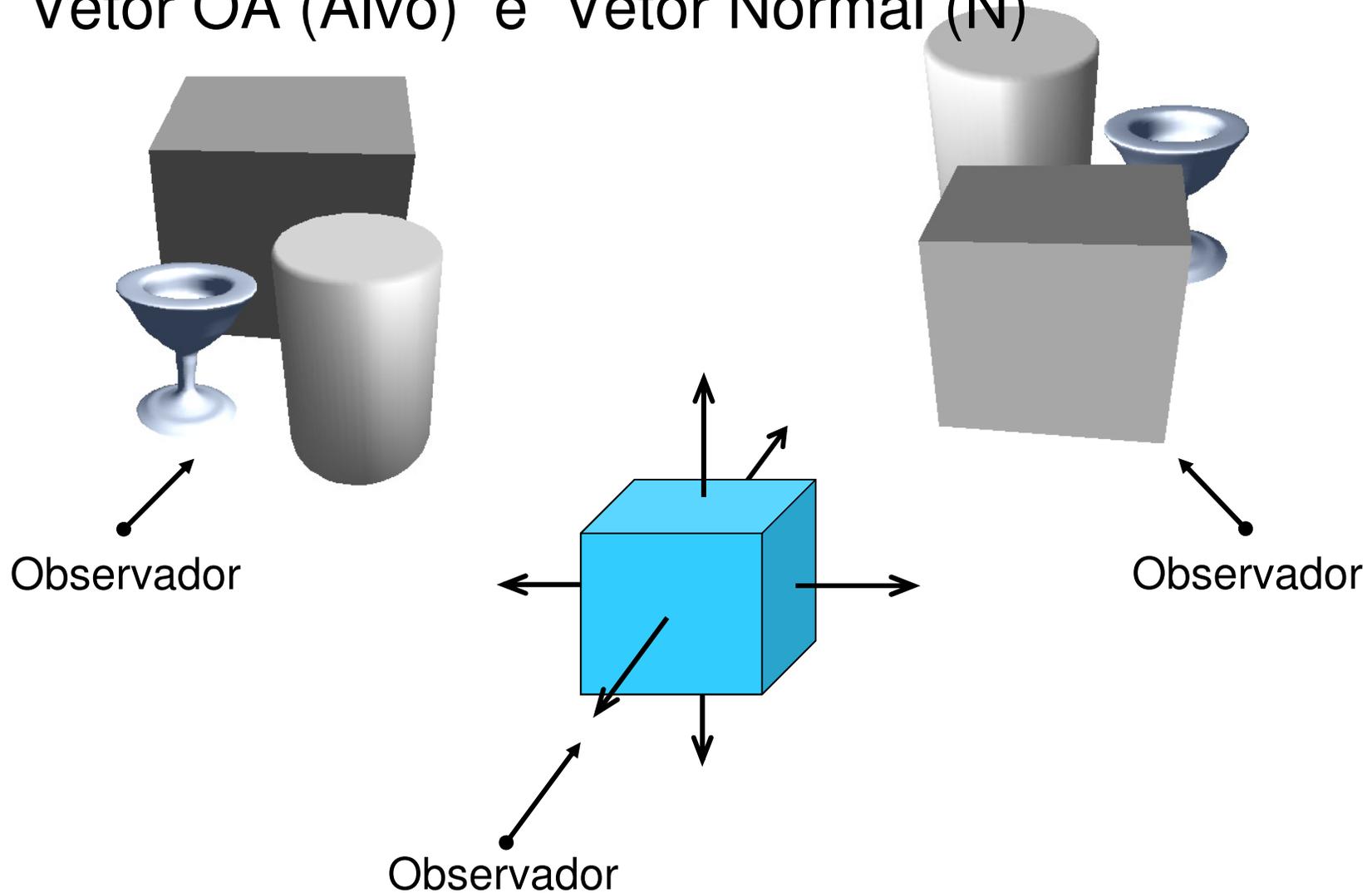
Remoção de Faces Traseiras

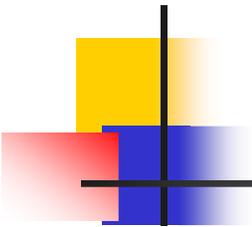
- Objeto aproximado por um poliedro sólido
 - Faces poligonais "cercam" completamente seu volume
 - Assume-se que todos os polígonos são definidos de tal forma que as normais às suas superfícies apontam "para fora" do poliedro



Remoção de Faces Traseiras

- Vetor \vec{OA} (Alvo) e Vetor Normal (N)



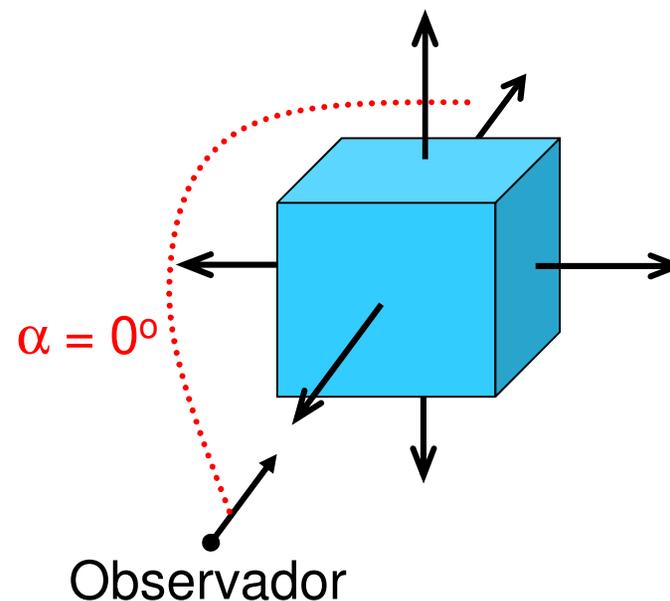
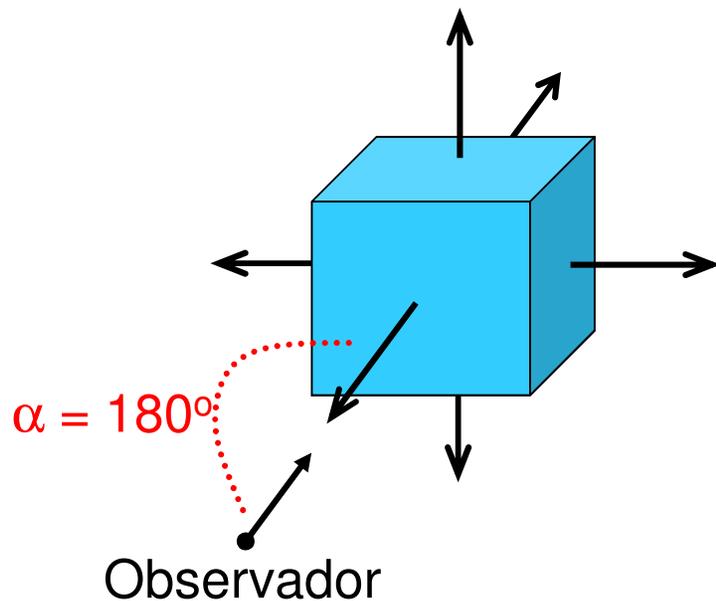


Remoção de Faces Traseiras

- Resumindo, esta estratégia simples:
 - É chamada de **remoção de faces traseiras**
 - Funciona para **objetos sólidos convexos**
 - Consiste em
 - Determinar as faces traseiras (que estão voltadas para o lado oposto do observador)
 - Eliminar estas faces do processo de desenho (*back-face culling*)

Remoção de Faces Traseiras

- Polígono com face traseira
 - No SRC (Sistema de Referência da Câmera)
 - Pode ser identificado pelo ângulo entre o vetor direção de observação (\vec{OA}) e o vetor normal de cada face

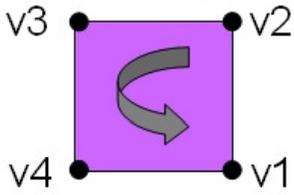
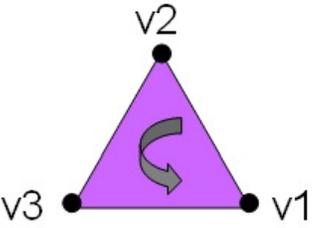
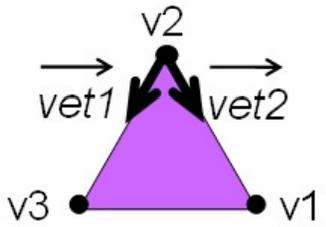
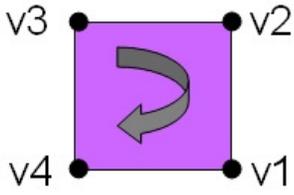
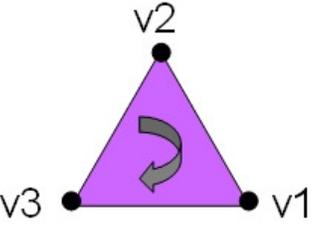


Remoção de Faces Traseiras

- Cálculo do vetor normal à face

1) Faces com os vértices ordenados no sentido anti-horário (ou horário)

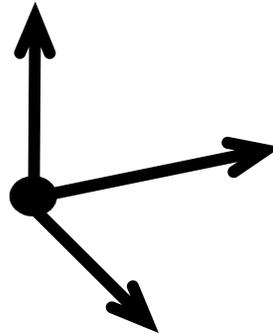
2) Processamento de dois vetores

 <p>Face composta pelos vértices v1, v2, v3, v4</p> <p>(a)</p>	 <p>Face composta pelos vértices v1, v2, v3</p>	 $vet1 = \frac{v3 - v2}{ v3 - v2 }$ $vet2 = \frac{v1 - v2}{ v1 - v2 }$ <p>(c)</p>
 <p>Face composta pelos vértices v4, v3, v2, v1</p> <p>(b)</p>	 <p>Face composta pelos vértices v3, v2, v1</p>	

Remoção de Faces Traseiras

- Cálculo do vetor normal à face

3) Normal = produto vetorial entre os dois vetores (o sentido depende da escolha dos vetores *vet1* e *vet2*)



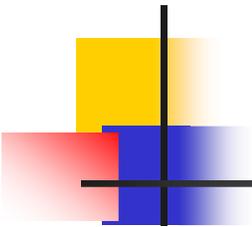
$$\mathbf{N} = \mathbf{vet1} \times \mathbf{vet2}$$

$$N.x = vet1.y * vet2.z - vet1.z * vet2.y;$$

$$N.y = vet1.z * vet2.x - vet1.x * vet2.z;$$

$$N.z = vet1.x * vet2.y - vet1.y * vet2.x;$$

$$\mathbf{vet1} \times \mathbf{vet2} \neq \mathbf{vet2} \times \mathbf{vet1}!!!$$



Remoção de Faces Traseiras

- Cálculo do ângulo através do produto escalar

- Expressão cartesiana

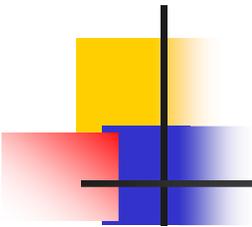
$$\vec{OA} \cdot \vec{N} = |\vec{OA}| * |\vec{N}| * \cos(\hat{\text{ângulo}})$$

- Expressão analítica

$$\vec{OA} \cdot \vec{N} = OA.x * N.x + OA.y * N.y + OA.z * N.z$$

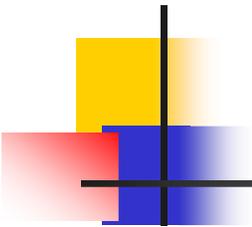
- O produto escalar $\vec{OA} \cdot \vec{N}$ é

- **Positivo** para um polígono de face traseira (vetores com ângulo menor que 90° , face virada para trás, não visível)
- **Negativo** para um polígono de face frontal (vetores com ângulo maior que 90° , face virada para a frente)
- **Igual a zero** para um polígono de face “lateral” (vetores perpendiculares, face não visível)



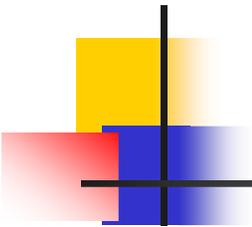
Remoção de Faces Traseiras

- Em OpenGL
 - Controle
 - *glEnable (GL_CULL_FACE)*
 - *glDisable (GL_CULL_FACE)*
 - Depende da ordem em que os vértices foram definidos na modelagem: *glFrontFace(GL_CW ou GL_CCW)*
 - Para indicar que as faces traseiras serão removidas
 - *void glCullFace (GLenum mode)*
 - O parâmetro *mode* pode receber os valores GL_FRONT (remove as faces frontais), GL_BACK (remove as faces traseiras) ou GL_FRONT_AND_BACK (remove as faces frontais e traseiras); nesse último caso, nenhuma face será desenhada, mas outras primitivas como pontos e linhas ainda serão.



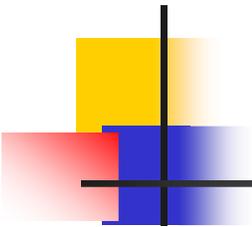
Roteiro

- ✓ Introdução
- ✓ Remoção de Faces Traseiras
- 3. Algoritmo do Pintor**
- 4. Algoritmo *Z-Buffer*
- 5. Árvores BSP



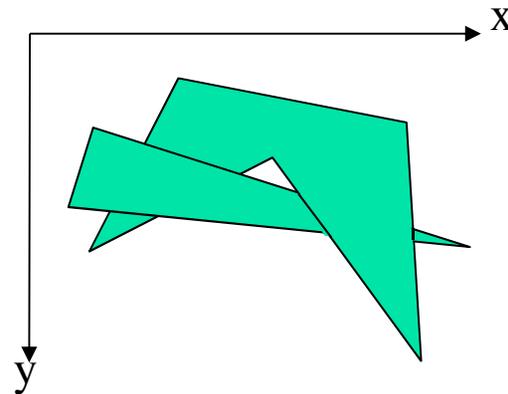
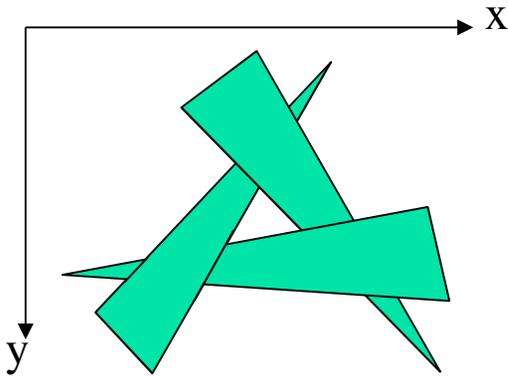
Algoritmo do Pintor *(Depth-Sorting Method)*

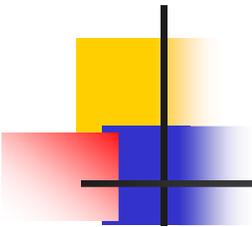
- Usa operações no espaço de imagem e no espaço de objeto
- Possui uma abordagem direta:
 1. Ordena todos os polígonos (faces) de acordo com a distância do observador (mais distante, maior coordenada Z)
 2. Resolve problemas de ambigüidade que podem ocorrer quando a extensão Z dos polígonos se sobrepõe (objetos se interseccionam)
 3. Pinta os polígonos na tela na ordem decrescente (mais distantes primeiro)
- Problema:
 - Ordenação não é trivial!



Algoritmo do Pintor

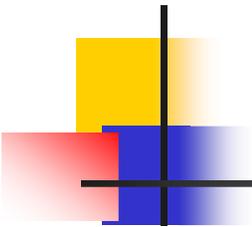
- Idéia básica
 - Ordenar os polígonos pelas suas distâncias do observador, colocá-los em um *buffer* em ordem decrescente de distância e pintá-los de trás para frente
- Problema de ambiguidade quando há interseção entre os objetos da cena





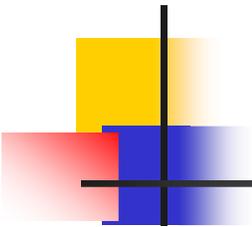
Algoritmo do Pintor

- Solução para o problema da ambigüidade
 - Chama-se o polígono do final da lista ordenada de P (*face mais distante*)
 - Antes deste polígono ser colocado no *buffer*, ele deve ser testado com cada face Q cujo Z se sobrepõe ao Z de P (*para algum vértice*)



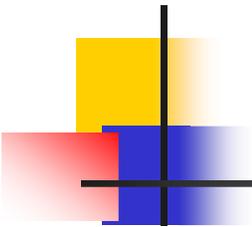
Algoritmo do Pintor

- Solução para o problema da ambiguidade
 - O teste é uma sequência de cinco passos (ver se P se sobrepõe a Q)
 1. A extensão X dos polígonos não se sobrepõem, então os polígonos não se sobrepõem
 2. A extensão Y dos polígonos não se sobrepõem, então os polígonos não se sobrepõem
 3. P está totalmente atrás de Q , então os polígonos não se sobrepõem
 4. Q está totalmente atrás de P , então os polígonos não se sobrepõem
 5. As projeções dos polígonos no plano XY (tela) não se sobrepõem, então os polígonos não se sobrepõem
 - Se um dos testes falha, assume-se que P sobrepõe Q , e trocam-se suas posições na lista



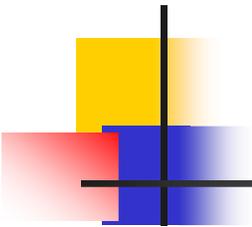
Algoritmo do Pintor

- Solução para o problema da ambiguidade
 - Nos casos de ambiguidade apresentados na figura anterior, mais cedo ou mais tarde Q será trocado novamente e o algoritmo entrará em *loop*
 - Para evitar o *loop*, considera-se a restrição que uma vez que o polígono é passado para o final da lista (e marcado) ele não pode ser movido novamente
 - Neste caso, os polígonos P ou Q são divididos um pelo plano do outro
 - O polígono original é descartado, suas duas partes são adicionadas na lista ordenada
 - Este procedimento é realizado em uma etapa de pré-processamento



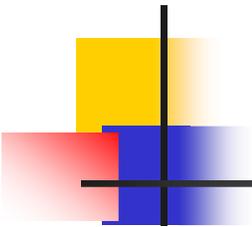
Roteiro

- ✓ Introdução
- ✓ Remoção de Faces Traseiras
- ✓ Algoritmo do Pintor
- 4. Algoritmo *Z-Buffer***
- 5. Árvores BSP**



Algoritmo *Z-Buffer*

- Operações são realizadas no espaço de imagem
- Baseado no procedimento de preenchimento de polígonos tipo *scan-line*
- Polígonos
 - Já foram projetados (estão no SRP)
 - Já foram mapeados para o espaço de tela (SRT)
 - Porém mantém-se a coordenada Z (SRP) de cada vértice, de modo que seja possível recuperar a informação de profundidade

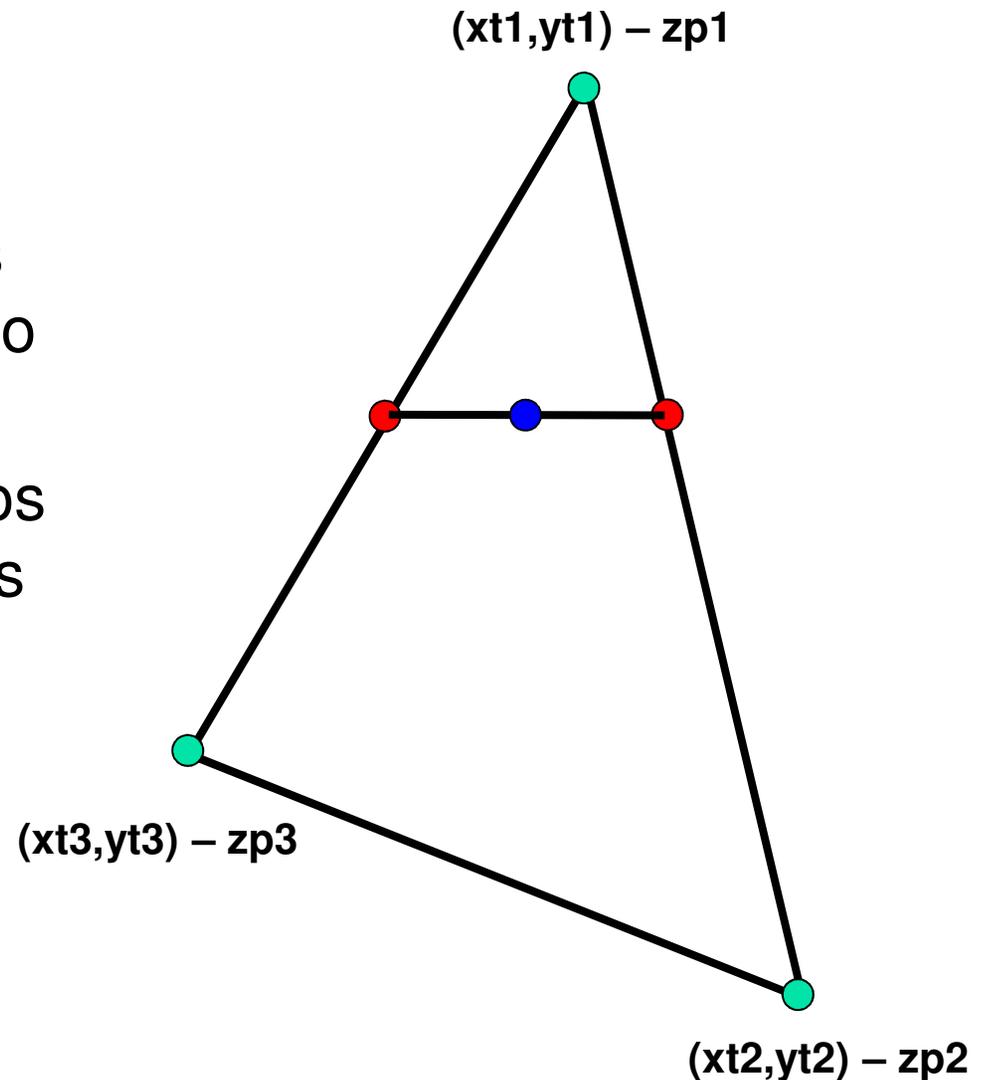


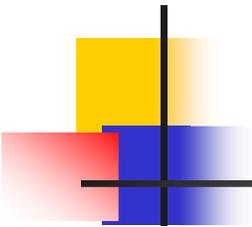
Algoritmo *Z-Buffer*

- Requer dois *buffers* (tamanho da tela)
 - *Buffer de Cor (Color Buffer)*
 - Armazena os valores de cor
 - Inicializado com a cor de fundo
 - *Buffer de Profundidade (Z-Buffer)*
 - Armazena os valores de Z (para cada pixel)
 - Inicializado com o maior valor possível para Z (depende da quantidade de bits por pixel no *z-buffer*)

Algoritmo Z-Buffer

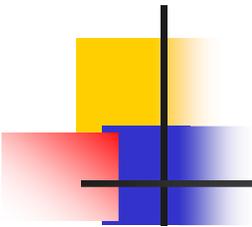
- Para cada polígono a ser exibido
 - Calcula os coeficientes da equação do plano do polígono
 - Coeficientes são usados para calcular os valores de Z no interior do polígono, através de interpolação





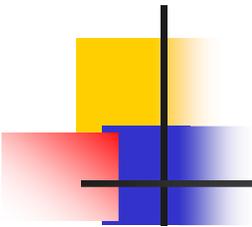
Algoritmo *Z-Buffer*

- Funcionamento
 - Inicialização dos *buffers*
 - Para cada pixel em cada polígono, compara o valor de profundidade (z) com o valor já armazenado no *Z-Buffer*, para determinar a visibilidade
 - Se $Z(x,y)$ é menor que o valor do *Z-Buffer* em (x,y) , então:
 - a) Coloca $Z(x,y)$ no *Z-Buffer* na posição (x,y) e
 - b) Coloca o valor do pixel no *buffer* de cor em (x,y)
 - Caso contrário, nem o *Z-buffer* nem o *buffer* de cor são alterados



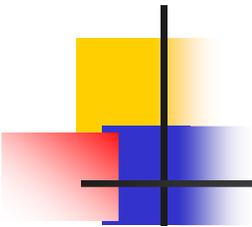
Algoritmo *Z-Buffer*

- Desvantagem
 - Precisa de uma grande quantidade de memória para o *Z-Buffer*
- Vantagem
 - Simples de implementar
- O desempenho do algoritmo tende a ser constante
 - Em média, o número de pixels que pertence a cada polígono decresce conforme o número de polígonos no volume de visualização aumenta



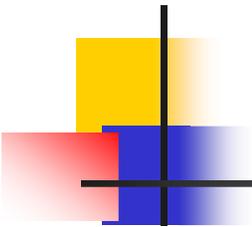
Roteiro

- ✓ Introdução
- ✓ Remoção de Faces Traseiras
- ✓ Algoritmo do Pintor
- ✓ Algoritmo *Z-Buffer*
- 5. Árvores BSP**



Árvores BSP

- BSP = *Binary Space Partitioning*
- Estrutura de dados utilizada para organizar objetos dentro de um espaço
- Tem aplicações na remoção de superfícies escondidas, em *ray tracing* e... em jogos!



Árvores BSP

■ Árvore BSP

- Subdivisão recursiva do espaço que trata cada segmento de linha (ou polígono, em 3D) como um plano de corte, o qual é usado para classificar todos os objetos restantes no espaço como estando na "frente" ou "atrás" desse plano
- Quando uma partição é inserida na árvore, esta é primeiro classificada em relação ao nodo raiz e então recursivamente em relação a cada filho apropriado

■ Pinta de trás para frente, como no algoritmo do Pintor

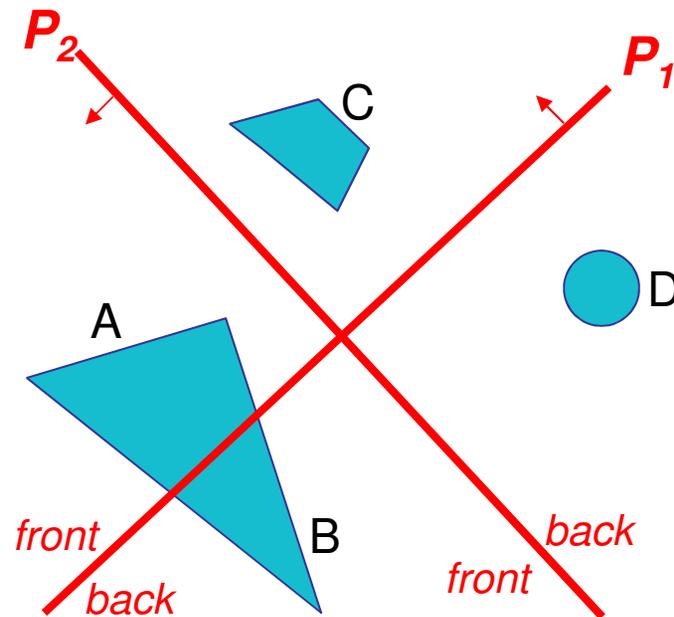
■ Vantagem

- Se o observador se move, e os objetos da cena estão em posições fixas, não é preciso reordenar os polígonos

http://symbolcraft.com/graphics/bsp/bsptreedemo_portugese.html

Árvores BSP

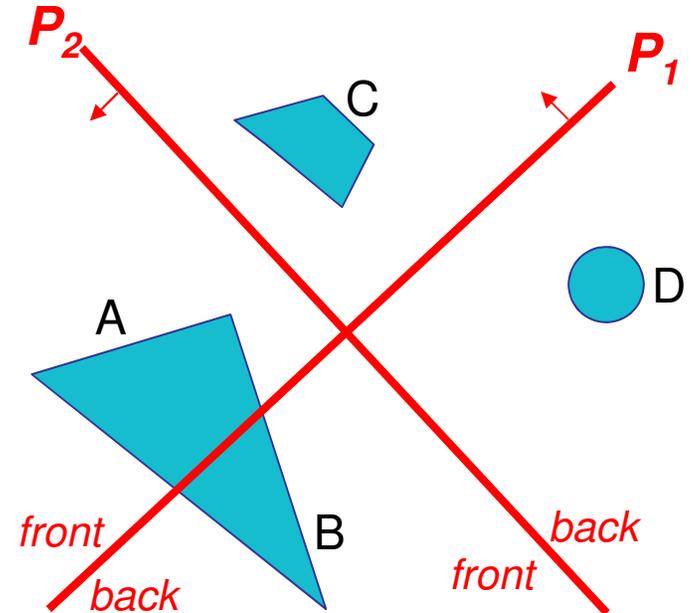
- A aplicação da árvore BSP para testes de visibilidade
 - Identificação das superfícies que estão “na frente” ou “atrás” do plano de partição em cada passo de subdivisão do espaço, de acordo com a direção de visualização



Árvores BSP

■ Funcionamento (exemplo)

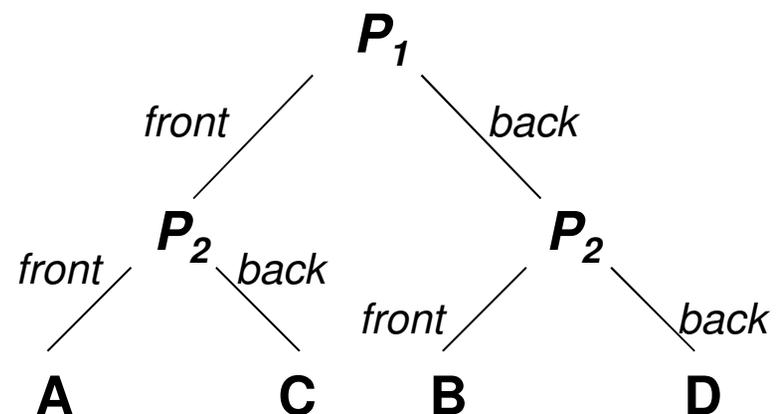
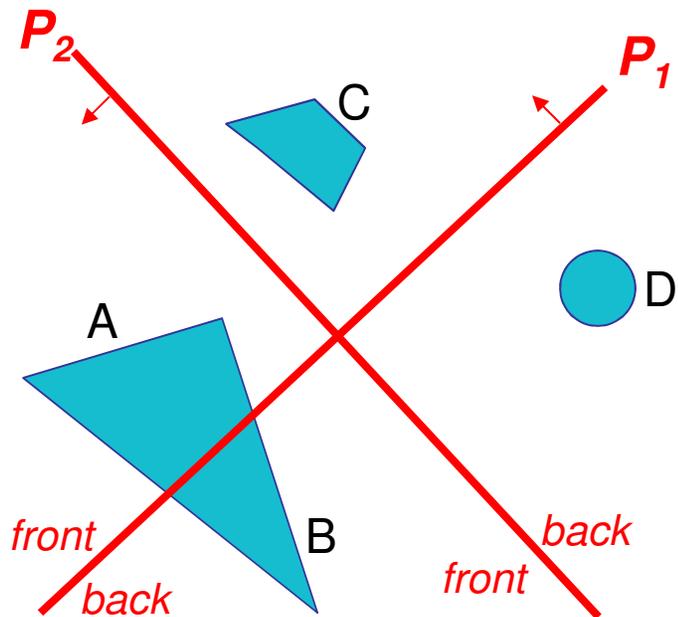
- Divisão do espaço em dois conjuntos de objetos com o plano P_1
 - Um conjunto está atrás do plano, de acordo com a direção de visualização
 - Outro conjunto está na frente do plano, também considerando a direção de visualização
 - Como um objeto é interseccionado pelo plano, este é dividido em dois objetos (A e B)



Árvores BSP

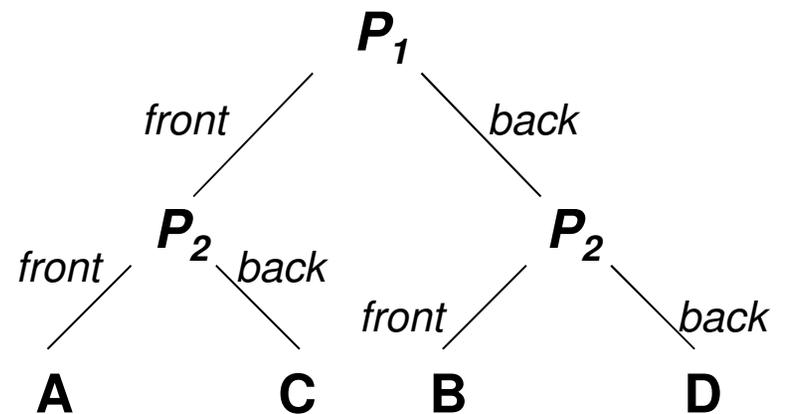
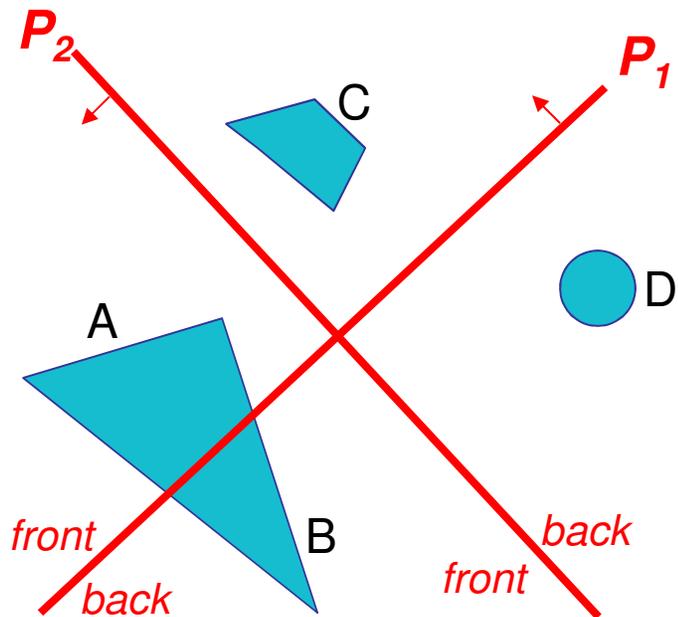
■ Funcionamento (exemplo)

- Então, os objetos A e C estão na frente de P_1 e os objetos B e D estão atrás
- É feita uma nova partição do espaço com o plano P_2 e construída a representação de árvore binária apresentada abaixo



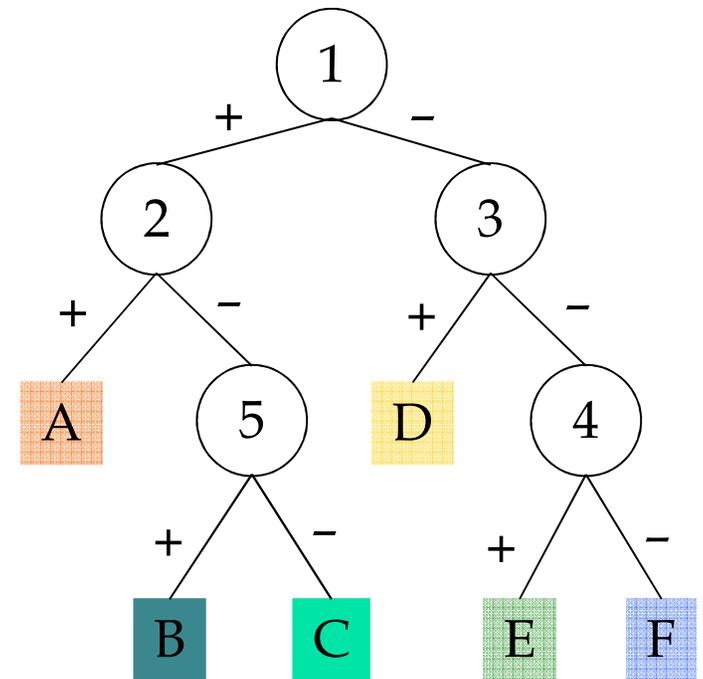
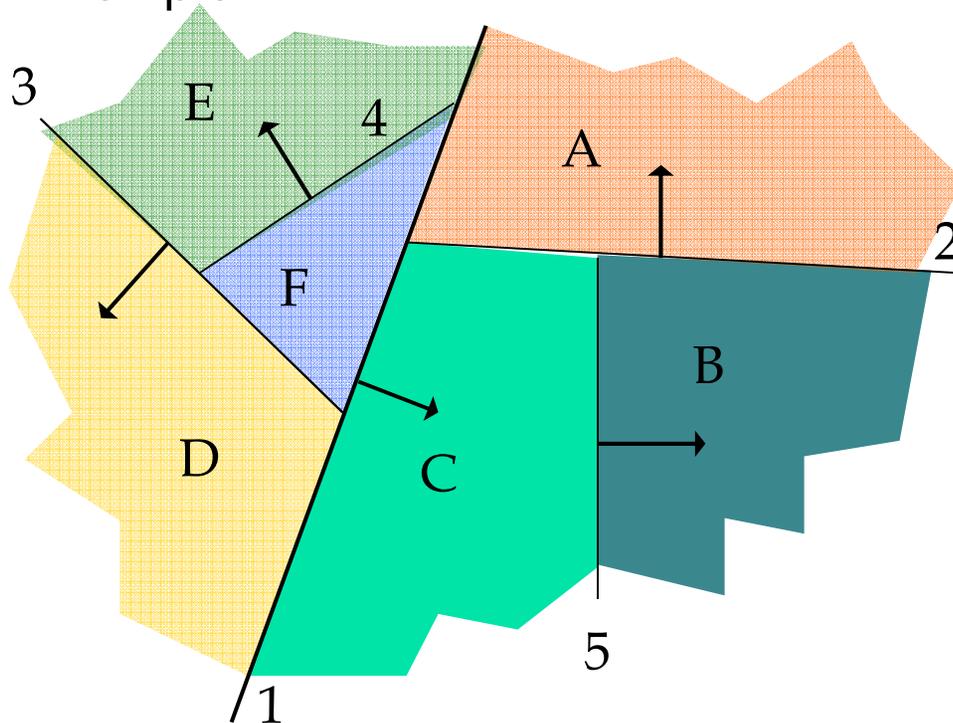
Árvores BSP

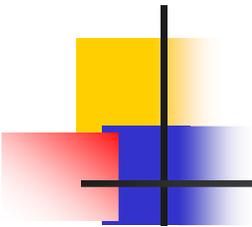
- Funcionamento (exemplo)
 - Nesta árvore, os objetos são representados como nodos terminais
 - Objetos frontais estão nas sub-árvores esquerda
 - Objetos que estão na parte de trás estão nas sub-árvores direita



Árvores BSP

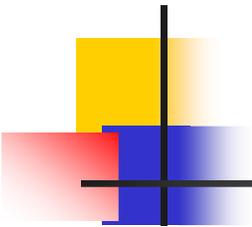
- Para objetos descritos através de uma malha (conjunto de faces), os planos de partição são escolhidos de tal maneira que coincidam com as faces
 - A árvore é construída com um plano de partição para cada face
 - Exemplo:





Árvores BSP

- Equações são usadas para identificar os polígonos (faces) que estão na sua frente ou atrás
- Qualquer polígono interseccionado por um plano de partição é dividido em duas partes
- Quando a árvore BSP estiver completa, ela é processada através da seleção das superfícies que devem ser exibidas na ordem *back to front*
 - As faces que estiverem mais na frente irão sobrepor as que estiverem mais atrás
- Implementações por hardware para construção e processamento de árvores BSP são usadas em alguns sistemas



Referências

- FOLEY, James D., et al. **Computer Graphics: Principles and Practice**. 2nd Ed., New York, Addison Wesley, 1990.
- HEARN, Donald; BAKER, M. Pauline. **Computer Graphics - C Version**. 2nd Ed. Upper Saddle River, New Jersey: Prentice Hall, 1997, 652 p.
- WATT, Alan. **3D Computer graphics**. 3th Ed. Harlow: Addison-Wesley, 2000. 570 p. il.