Eliminação de Superfícies Escondidas

Isabel Harb Manssour

Porto Alegre, Maio de 2003

Roteiro

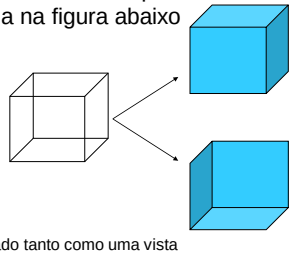
1. Introdução
2. Remoção de Faces Traseiras
3. Algoritmo do Pintor
4. Algoritmo *Z-Buffer*
5. Árvores BSP

Introdução

- Eliminação de superfícies escondidas
  - Também conhecido por Remoção de Elementos Ocultos
  - É um dos problemas mais difíceis da Computação Gráfica
  - Os algoritmos são usados para determinar as linhas, arestas, superfícies ou volumes que são visíveis ou não para um observador localizado em um ponto específico no espaço

Introdução

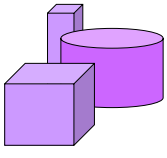
- Uma das necessidades de eliminar superfícies escondidas está ilustrada na figura abaixo



  - 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 mostra a figura 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 ainda, 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-objeto**
  - Métodos que trabalham no **espaço-imagem**

Introdução

- Métodos que trabalham no espaço-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

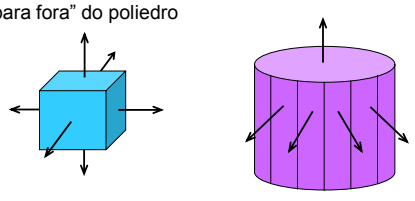
Introdução

- Métodos que trabalham no espaço-imagem
  - Maioria dos algoritmos
  - Implementados no sistema de coordenadas de tela (SRT) no qual os objetos são visualizados
  - Determinam qual dentre n objetos é visível em cada pixel da cena
  - Para cada pixel:
    - Determinam qual objeto está mais próximo do observador, de acordo com a projetante que passa pelo pixel
    - Pintam o pixel com a cor adequada
- *Obs.*: Alguns algoritmos combinam as duas abordagens.

Roteiro

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

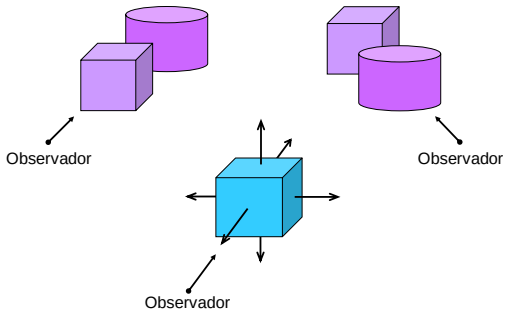


Remoção de Faces Traseiras

- Objeto aproximado por um poliedro sólido
  - Se nenhuma parte do poliedro é atingida pelo plano de corte, então os polígonos cuja a normal à superfície aponta para o lado oposto do observador, estão em uma parte do poliedro onde a visibilidade é totalmente bloqueada por outros polígonos

Remoção de Faces Traseiras

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

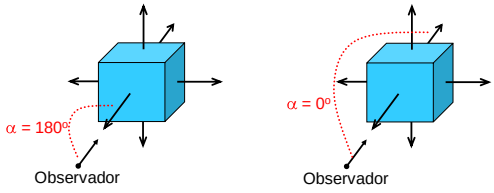


Remoção de Faces Traseiras

- 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 desenho (*back-face culling*)

Remoção de Faces Traseiras

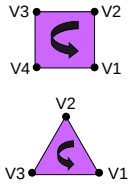
- No SRC (Sistema de Referência da Câmera), um polígono com face traseira pode ser identificado pelo produto escalar que a normal à sua superfície forma com o vetor da direção de observação ($\overrightarrow{OA}$)
  - Objetivo é determinar o ângulo entre a direção de observação e o vetor normal de cada face



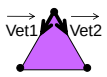
Remoção de Faces Traseiras

- Cálculo do vetor normal à face

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



2) Processamento de dois vetores



$$Vet1 = \frac{V3 - V2}{|\ V3\text{-}V2\ |}$$

$$Vet2 = \frac{V1 - V2}{|\ V1\text{-}V2\ |}$$

3) Normal = produto vetorial entre os dois vetores (o sentido depende da escolha dos vetores Vet1 e Vet2)

N = Vet1 x 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;

Vet1 x Vet2 ≠ Vet2 x Vet1!!!

Remoção de Faces Traseiras

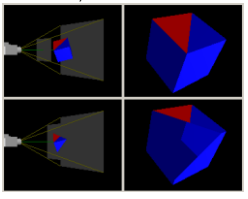
- Cálculo do produto escalar
  - Expressão cartesiana

    $\overrightarrow{OA} \bullet \vec{N} = |\overrightarrow{OA}| * |\vec{N}| * \cos(\text{ângulo})$
  - Expressão analítica

    $\overrightarrow{OA} \bullet \vec{N} = OA.x * N.x + OA.y * N.y + OA.z * N.z$
- O produto escalar $\overrightarrow{OA} \bullet \vec{N}$ é
  - Positivo para um polígono de face traseira (vetores com ângulo menor que 90º, face virada para trás)
  - Negativo para um polígono de face frontal (vetores com ângulo maior que 90º, face virada para a frente)
  - Igual a zero para um polígono de face "lateral" (vetores perpendiculares, face não-visível)

Remoção de Faces Traseiras

- Se o poliedro teve suas faces frontais recortadas, então os polígonos de faces traseiras podem (devem) receber um tratamento especial
  - Neste caso, se a remoção não é desejada, uma abordagem simples resume-se em tratar um polígono de face traseira como se fosse de face frontal, alterando a sua normal para a direção oposta



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

Remoção de Faces Traseiras

- A idéia é simples
- Dependendo da solução adotada a implementação requer uma certa capacidade computacional
  - Memória
  - Velocidade de processamento

Roteiro

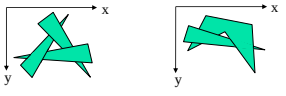
- ✓ Introdução
- ✓ Remoção de Faces Traseiras
- **3. Algoritmo do Pintor**
- 4. Algoritmo *Z-Buffer*
- 5. Árvores BSP

Algoritmo do Pintor *(Depth-Sorting Method)*

- Usa operações no espaço-imagem e no espaço-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 sobrepõem (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 ambigüidade 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*
  - Antes deste polígono ser colocado no *buffer*, ele deve ser testado com cada face *Q* cuja extensão de Z sobrepõe a extensão de Z de *P*

Algoritmo do Pintor

- Solução para o problema da ambigüidade
  - O teste é uma seqüência de cinco passos
    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 os cinco testes falharem, assume-se que P sobrepõe Q, e trocam-se suas posições na lista, marcando que Q foi movido para esta nova posição no fim da lista

Algoritmo do Pintor

- Solução para o problema da ambigüidade
  - Nos casos de ambigüidade 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-imagem
- Baseado no procedimento de preenchimento de polígonos tipo *scan-line*
- Os polígonos já estão transformados projetivamente, já estão mapeados para o espaço de tela, sendo mantida, entretanto, a coordenada Z de cada vértice de modo a poder ser recuperada a informação de profundidade

Algoritmo *Z-Buffer*

- Requer dois *buffers* (tamanho da tela)
  - *Refresh Buffer*
    - Armazena os valores de intensidade
    - Inicializado com a intensidade do pixel da cor de fundo
  - *Z-Buffer*
    - Armazena os valores de Z (para cada pixel)
    - Inicializado com o maior valor de Z

Algoritmo *Z-Buffer*

- Para cada polígono a ser exibido, calcula os coeficientes da equação do plano do polígono
  - Usados para calcular o valor de Z
- Para cada pixel a ser ligado durante o preenchimento do polígono, verifica se Z é menor do que o valor do *Z-Buffer*

Algoritmo *Z-Buffer*

- Funcionamento
  - Inicialização dos *buffers*
  - Para cada posição em cada polígono, compara os valores de profundidade com os valores armazenados no Z-Buffer para determinar a visibilidade
    - Cálculo da profundidade Z para cada posição (x,y) do polígono
    - 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 do polígono em Z(x,y) no *Refresh Buffer* em (x,y).

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 porque, em média, o número de pixels que pertencem a cada polígono decresce conforme o número de polígonos no volume visualização aumenta

Algoritmo *Z-Buffer*

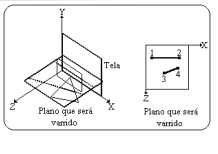
- Algoritmo implementado em *hardware* (OpenGL)
  - Inicialização:
    - ***glClear*** *(GL_COLOR_BUFFER_BIT |* ***GL_DEPTH_BUFFER_BIT****);*
  - Controle:
    - ***glEnable*** *(****GL_DEPTH_TEST****);*
    - ***glDisable*** *(****GL_DEPTH_TEST****);*

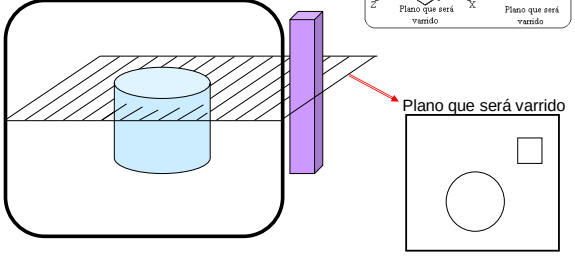
Algoritmo *Z-Buffer*

- *Scan-Line Z-Buffer*
  - Procedimento é semelhante, porém processa a cena varrendo linha por linha
  - Neste caso, o tamanho do *buffer* corresponde ao tamanho da linha
  - Tanto o *Z-Buffer* como o *scan-line Z-Buffer* operam com a noção de varredura do polígono a ser exibido, linha a linha
  - A intersecção de um plano que será varrido com a cena 3D define uma janela *scan-line* onde o problema da remoção de elementos ocultos é resolvido

Algoritmo *Z-Buffer*

- *Scan-Line Z-Buffer*
  - Exemplos





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 e em *ray tracing*

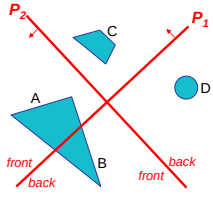
Árvores BSP

- 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://www.inf.pucrs.br/cg/Aulas/Applets/bsp/bsptreedemo.html

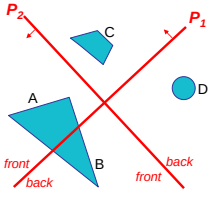
Árvores BSP

- 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



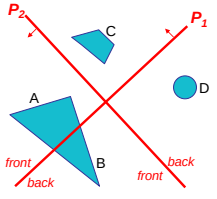
Árvores BSP

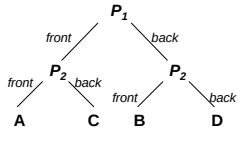
- Funcionamento (exemplo)
  - Divisão do espaço em dois conjuntos de objetos com o plano P_1
    - Um conjunto está atrás do plano, de acordo com a direção de visualização
    - Outro conjunto está na frente do plano, também considerando a direção de visualização
    - Como um objeto é interseccionado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



Árvores BSP

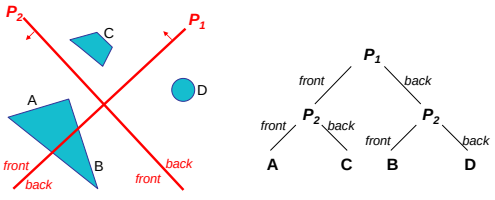
- Funcionamento (exemplo)
  - Então, os objetos A e C estão na frente de P_1 e os objetos B e D estão atrás
  - É feita uma nova partição do espaço com o plano P_2 e construída a representação de árvore binária apresentada abaixo





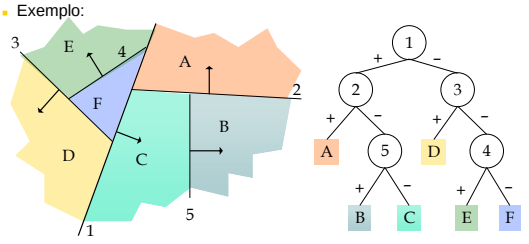
Árvores BSP

- 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.