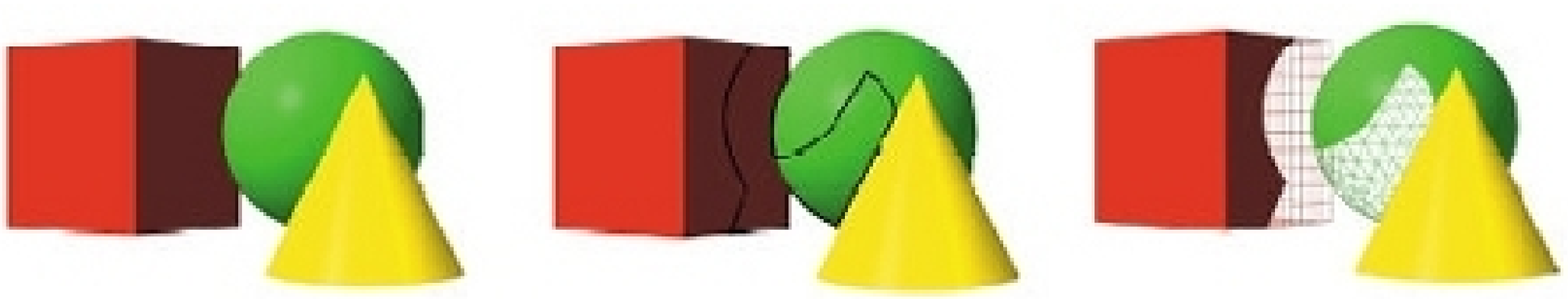
Computação Gráfica

Visibilidade

Moisés Henrique Ramos Pereira

Introdução

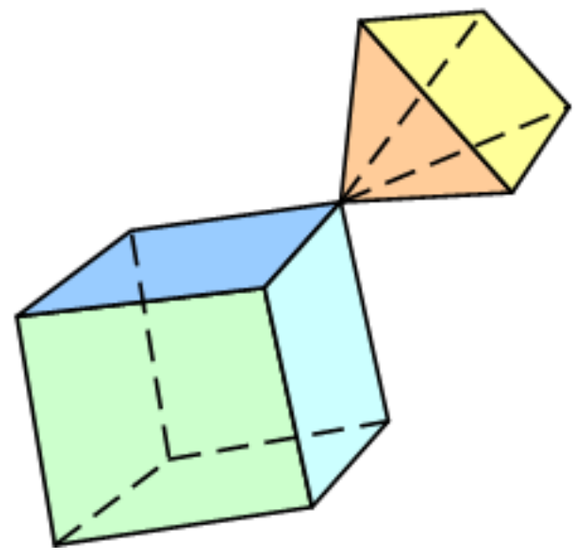
- Por que estudar algoritmos de visibilidade?
    - é necessário determinar o que é visível em um cena a partir de uma posição de visualização qualquer.
- No universo 3D, este algoritmos são também conhecidos como algoritmos de detecção de superfícies escondidas ou algoritmos de remoção de superfícies escondidas.



Introdução

- É interessante notar uma sutil diferença entre esta nomenclatura, detecção e remoção de superfícies.

    - a primeira é muito utilizada na representação wireframe de objetos tridimensionais onde a detecção de superfícies escondidas ajuda a dar a sensação de profundidade ao objeto (linhas tracejadas)



Introdução

- A escolha de qual, ou quais, dos métodos será utilizado depende de uma série de fatores que incluem:
    - a complexidade da cena, se a mesma apresenta muitos objetos, se os mesmos estão muito espaçados ou agrupados.
    - o tipo dos objetos na cena, polígonos simples ou objetos complexos descritos com milhões de superfícies.
    - o hardware disponível para a execução do método.

Introdução

- Algoritmos de visibilidade que serão abordados:
    - *back-face culling*
    - *buffer* de profundidade (*z-buffer*)
    - *a-buffer*
    - *scan-line*
    - ordem de profundidade
    - subdivisão do espaço (incluindo *octree*);

Classificação dos Algoritmos

- Os algoritmos de detecção de visibilidade são classificados de acordo com duas abordagens principais:
  - métodos baseados em objetos onde comparações entres objetos ou partes de objetos são realizadas para determinar quais são visíveis.
  - métodos baseados na imagem onde a visibilidade é calculada ponto-a-ponto para cada *pixel* da imagem formada no plano de projeção.
- Métodos baseados na imagem são mais comuns.

Classificação dos Algoritmos

- Algoritmicamente temos:
  - método baseado nos objetos - custo n^2 onde *n* é a quantidade de objetos:

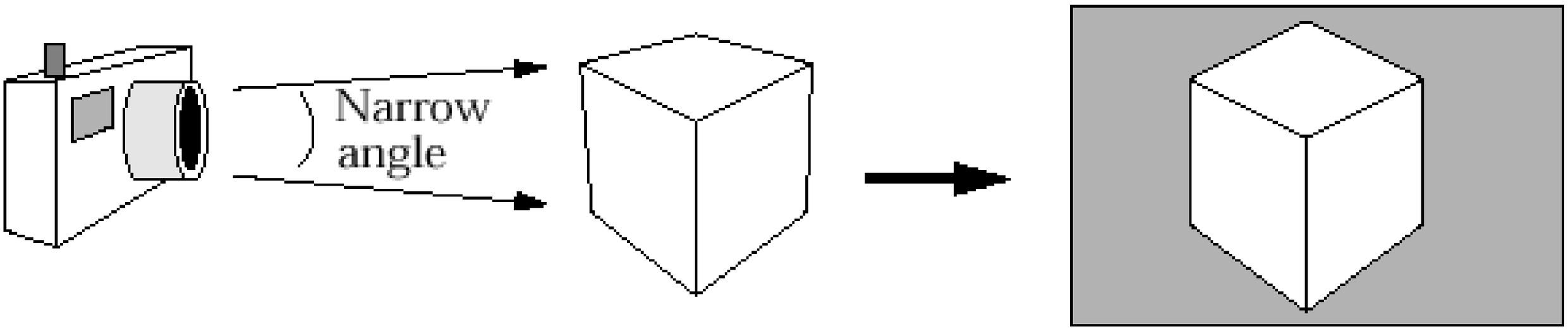
```
para cada objeto na cena faça
    determine e elimine as partes oclusas do objeto
    desenhe o restante
```

  - métodos baseados na imagem - custo *np* onde *n* é a quantidade de objetos na cena e *p* a quantidade de pixels no plano de projeção:

```
para cada pixel na imagem faça
    determine o objeto mais próximo do observador
        na direção do raio de projeção
    desenhe o pixel com cor deste objeto
```

Back-Face Culling

- Também conhecido como *back-face detection*, ou detecção de face oculta, este é o método mais simples de detecção de visibilidade.
  - consiste em descobrir as faces posteriores do poliedro e descartá-las.



Back-Face Culling

- É sabido que um ponto (x, y, x) está atras de uma superfície poligonal se

$$Ax + By + Cz + D < 0$$

  - onde A, B, C e D são os parâmetros do plano para a superfície.

- De forma geral, um polígono possuí duas faces: dentro e fora.

  - as faces de dentro, em um objeto fechado, nunca serão visíveis.

  - assim, se o a face de dentro de um polígono for a visível a partir do vetor de visualização da câmera, esta face não deve aparecer no resultado final.

Back-face Culling

- O método de back-face pode se tornar ainda mais simples se considerarmos o produto escalar entre vetor normal à face em análise (N) e o vetor de visualização (V).

  - se V . N > 0, os vetores estão na mesma direção e portanto a face em questão não será visível.

€

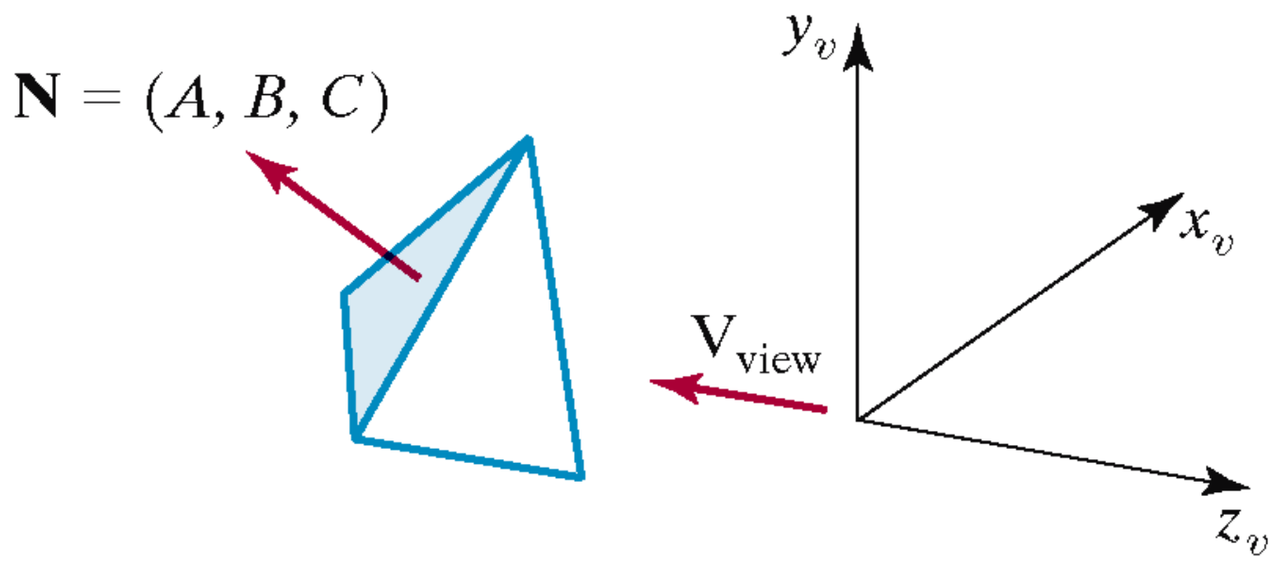
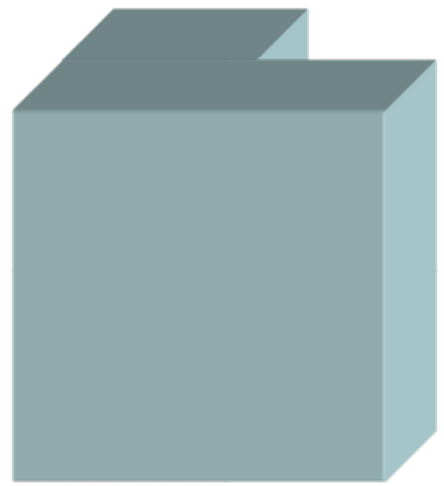


Figure 9-2

Back-face Culling

- Tipicamente, o método de *back-face culling* elimina aproximadamente metade das faces oclusas em um cena tridimensional.
    - por ser um método barato, é interessante utilizá-lo em uma etapa inicial do processo de eliminação das superfícies oclusas da cena.

- Para superfícies mais complexas, como a apresentada ao lado, o método não possui desempenho satisfatório e são necessárias técnicas melhores para lidar com tais situações:
    - as superfícies deste objeto se sobrepõem na linha de visada.



Buffer de Profundidade

- Também conhecido como *z-buffer*, uma vez que lida com a profundidade dos objetos (eixo z).

- Este método compara valores de profundidade das superfícies que compõem a cena para cada pixel do plano de projeção.

  - cada superfície da cena é processada separadamente, para cada pixel, um por vez.

- De forma geral, este método é utilizado em cenas contendo apenas polígonos.

- Uma vez que o cálculo de profundidade é simples, este método tende a ser muito rápido.

Buffer de Profundidade

- Tipicamente, no z-buffer, o cálculo de profundidade é realizado em coordenadas normalizadas, isto é, coordenadas com valores entre 0,0 e 1,0.

- E como o próprio nome indica, este método utiliza *buffers* para armazenar os resultados intermediários.

- Estes *buffers* possuem dimensões iguais às do plano de projeção e, para o *z-buffer*, dois *buffers* são empregados:

  - *buffer de profundidade* que armazena a menor profundidade corrente para cada *pixel* do plano de projeção.

  - *frame buffer* que armazena a cor corrente para cada *pixel* do plano de Projeção.

Buffer de Profundidade

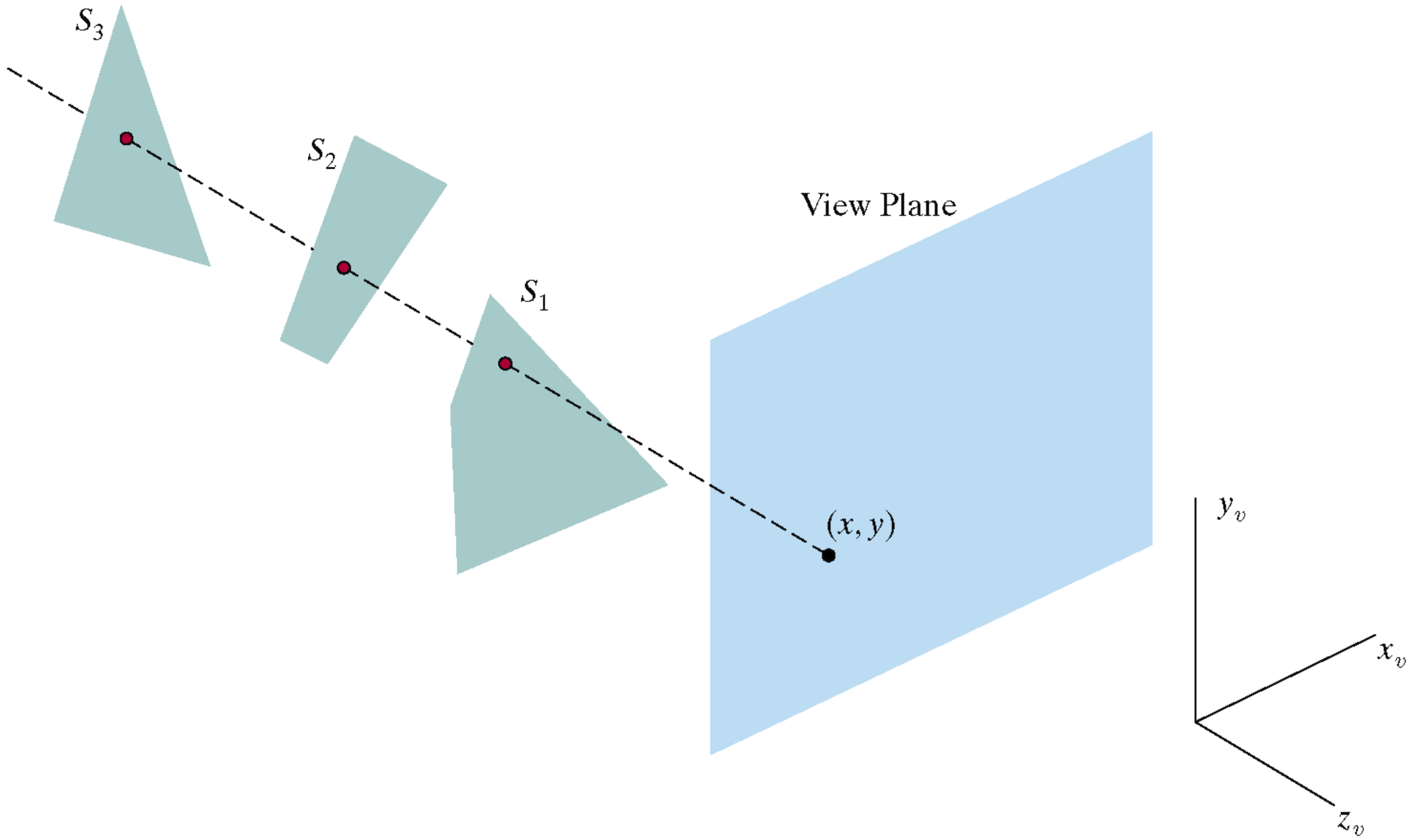


Figure 9-4

Buffer de Profundidade - Algoritmo

1. Inicie o buffer de profundidade e o frame buffer de tal forma que, para cada posição (x,y)

    bufferProdundidade(x, y) = 1.0;

    frameBuffer(x, y) = background color;

Buffer de Profundidade - Algoritmo

2. Processando cada polígono da cena, um por vez:

- para cada pixel (x,y) do polígono no plano de projeção, calcule a profundidade **z** (se já não tiver calculado)

- se z < bufferProfundidade(x,y), calcule a cor da superfície nesta posição e faça:

bufferProfundidade(x,y) = z;

frameBuffer(x, y) = corSuperficie(x,y)

Após todos os polígonos terem sido processados, o *buffer* de profundidade e o *frame buffer* irão conter os valores corretos para a renderização da cena.

Calculando a Profundidade

- Para qualquer ponto (x, y) da superfície, sua profundidade com relação ao plano de projeção é calculada a partir da equação do plano da seguinte forma:

$$z = \frac{-Ax - By - D}{C}$$

- Processando o polígono, linha-por-linha, em um método conhecido como *scan-line*, posições adjacentes no eixo x diferem por ± 1 (assim como valores adjacentes no eixo y).

- Assim, no cálculo de profundidade, o valor do *pixel* adjacente é dado por:

$$z' = \frac{-A(x+1) - By - D}{C} \qquad z' = z - \frac{A}{C}$$

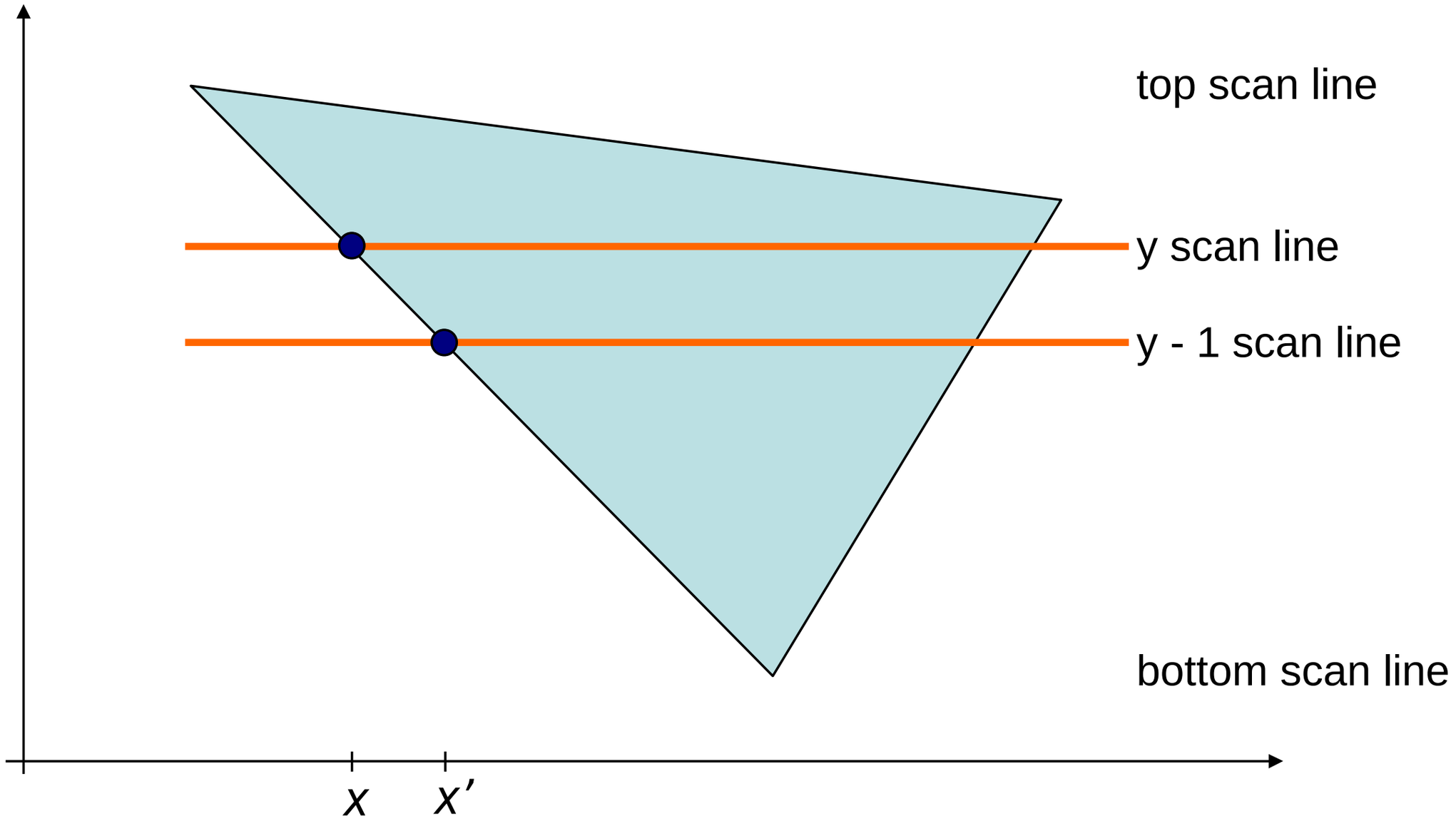
onde z é a profundidade do *pixel* anterior

Calculando a Profundidade

- O algoritmo de *buffer* de profundidade procede então, da seguinte forma:
  - comece pelo vértice no topo do polígono (aquele com menor x e maior y coordenada).
  - recursivamente calcule os valores de profundidade para cada ponto iniciando uma *Scan-line.*
  - calcule as profundidades ao longo da *scan-line* considerada usando

$$z' = z - \frac{A}{C}$$

Calculando a Profundidade

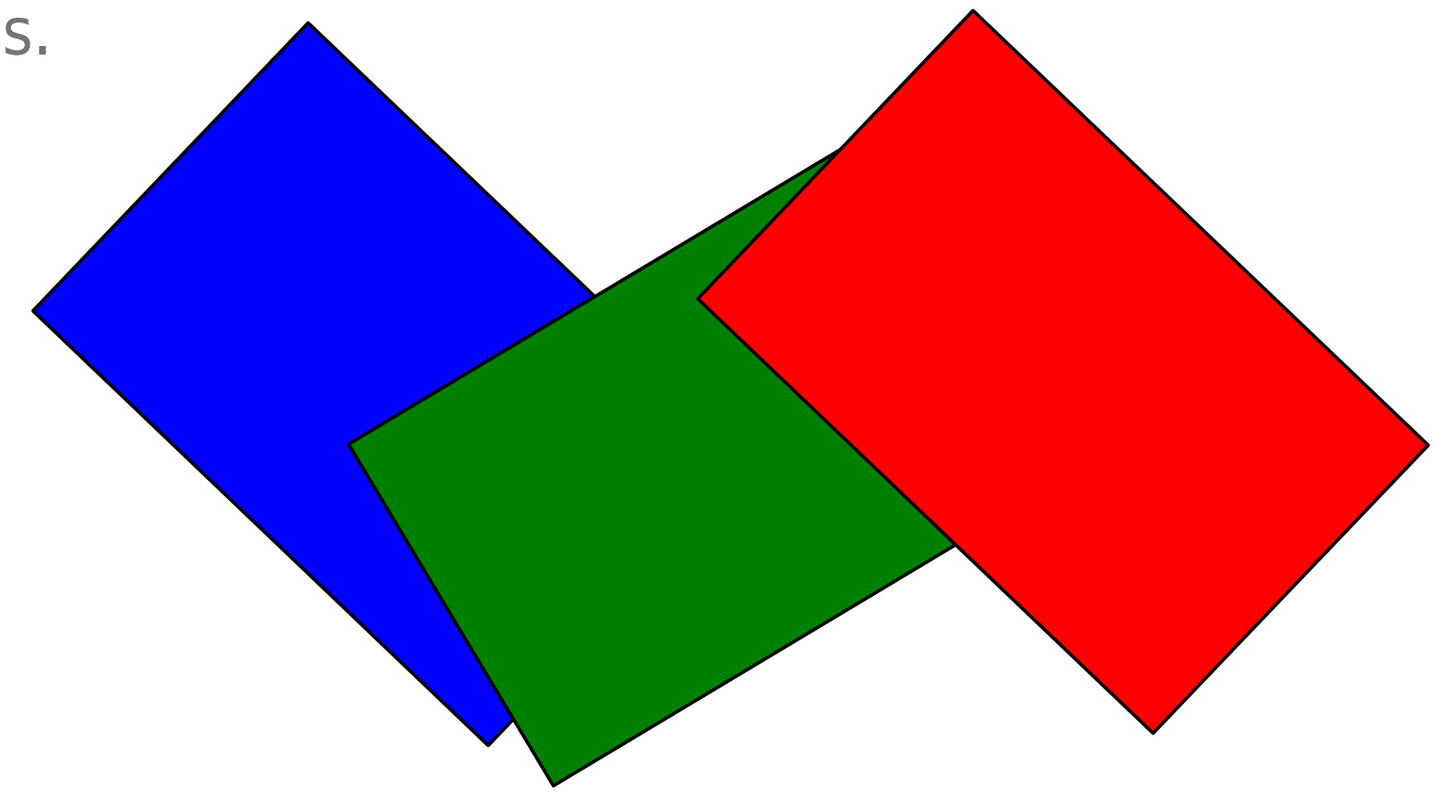


A-Buffer

- O método de visibilidade *A-buffer* é uma extensão do método de *buffer* de profundidade (*z-buffer*).
  - é um método de detecção de visibilidade desenvolvido pela Lucasfilm Ltd para seu sistema de renderização REYES
    - *Renders Everything You Ever Saw*

A-Buffer

- O método *A-buffer* expande o buffer de profundidade permitindo a criação de transparências na cena renderizada.
  - a estrutura de dados chave do *A-buffer* é o *buffer* de acumulação (dai o seu nome).
  - cada posição do *buffer* pode referenciar um lista encadeada de Superfícies.



A-Buffer

- Cada posição no A-buffer possui dois campos:
  - *Depth Field*: armazena um número real (positivo, negativo ou zero).
  - Surface Data Field: armazena dados da superfície ou um ponteiro.

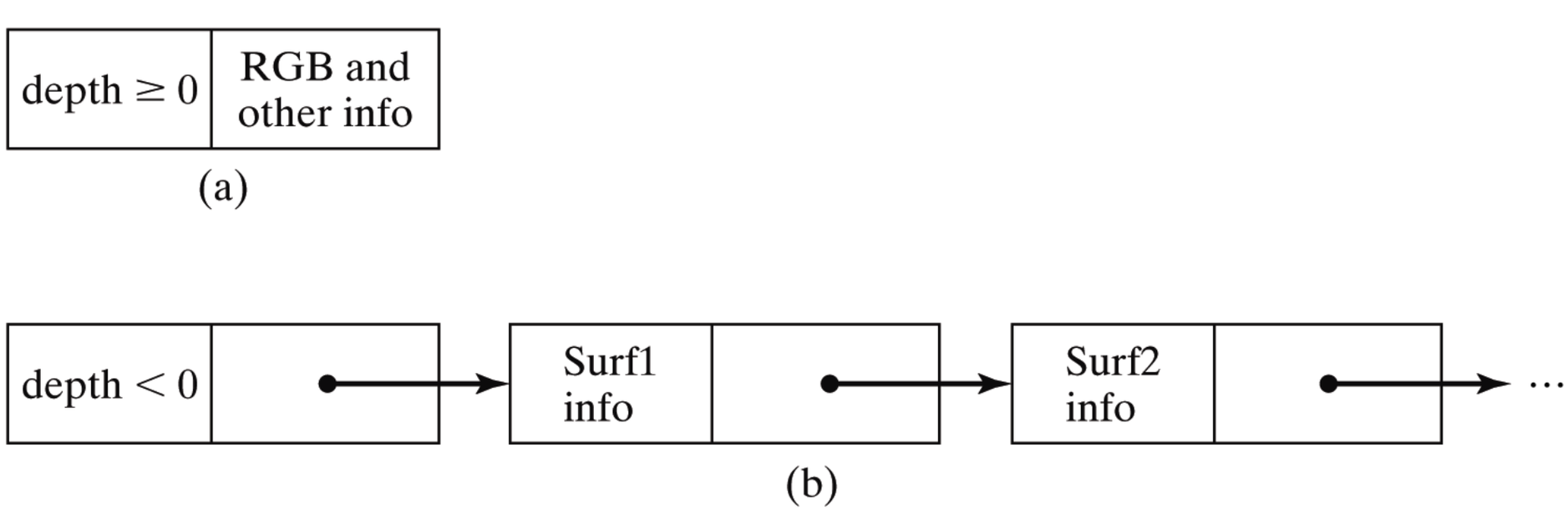


Figure 9-9

Two possible organizations for surface information in an A-buffer representation for a pixel position. When a single surface overlaps the pixel, the surface depth, color, and other information are stored as in (a). When more than one surface overlaps the pixel, a linked list of surface data is stored as in (b).

A-Buffer

- Se o valor do *depth field* for positivo (≥ 0), este número representa a profundidade da superfície que sobrepõem aquele pixel, como acontece no buffer de profundidade.

- Se o valor do *deth field* for negativo (< 0), isto indica contribuição de múltiplas superfícies no resultado final do pixel.

  - o *data field* contém então um ponteiro para uma lista encadeada de dados de superfícies (aquelas que irão contribuir para o resultado final).

A-Buffer

- Informações utilizadas no método incluem:
  - intensidade de cor (RGB)
  - opacidade (transparência)
  - profundidade
  - percentual de cobertura da área
  - identificador da superfície
  - outros parâmetros de renderização

A-Buffer - Algoritmo

- O algoritmo se comporta igual ao *z-buffer.*

- Valores de profundidade e opacidade são utilizados para determinar a cor final de um *pixel.*

Scan-line

- O scan-line é um método baseado na imagem para a identificação de superfícies visíveis.

- Em linhas gerais, o método calcula e compara valores de profundidade ao longo das várias *scan-lines* de uma cena.

- Durante seu processamento, o método conta com informações a respeito dos objetos que compõem a cena.

  - tais informações estão contidas em tabelas que armazenam os vértices, arestas, faces de dos polígonos que formam o objeto.

Scan-line

- Duas tabelas são de suma importância para o método de *scan-line*:
  - tabela de arestas
  - tabela de faces

- Na tabela de arestas armazena estão:
  - as coordenadas de cada linha presente na cena
  - o inverso da inclinação de cada uma destas linhas
  - ponteiros para a tabela de faces, conectando arestas às superfícies

Scan-line

- Na tabela de faces estão
  - os coeficientes (*A, B, C, D*) da equação do plano
  - propriedades do material da superfície (por exemplo, índice de reflexão e refração de luz).
  - possivelmente os ponteiros para a tabela de arestas.

Scan-line - Algoritmo

- Para facilitar a busca por superfícies que atravessam uma dada *scan-line,* uma lista de arestas ativas é mantida para cada *scan-line* à medida que estas vão sendo processadas.

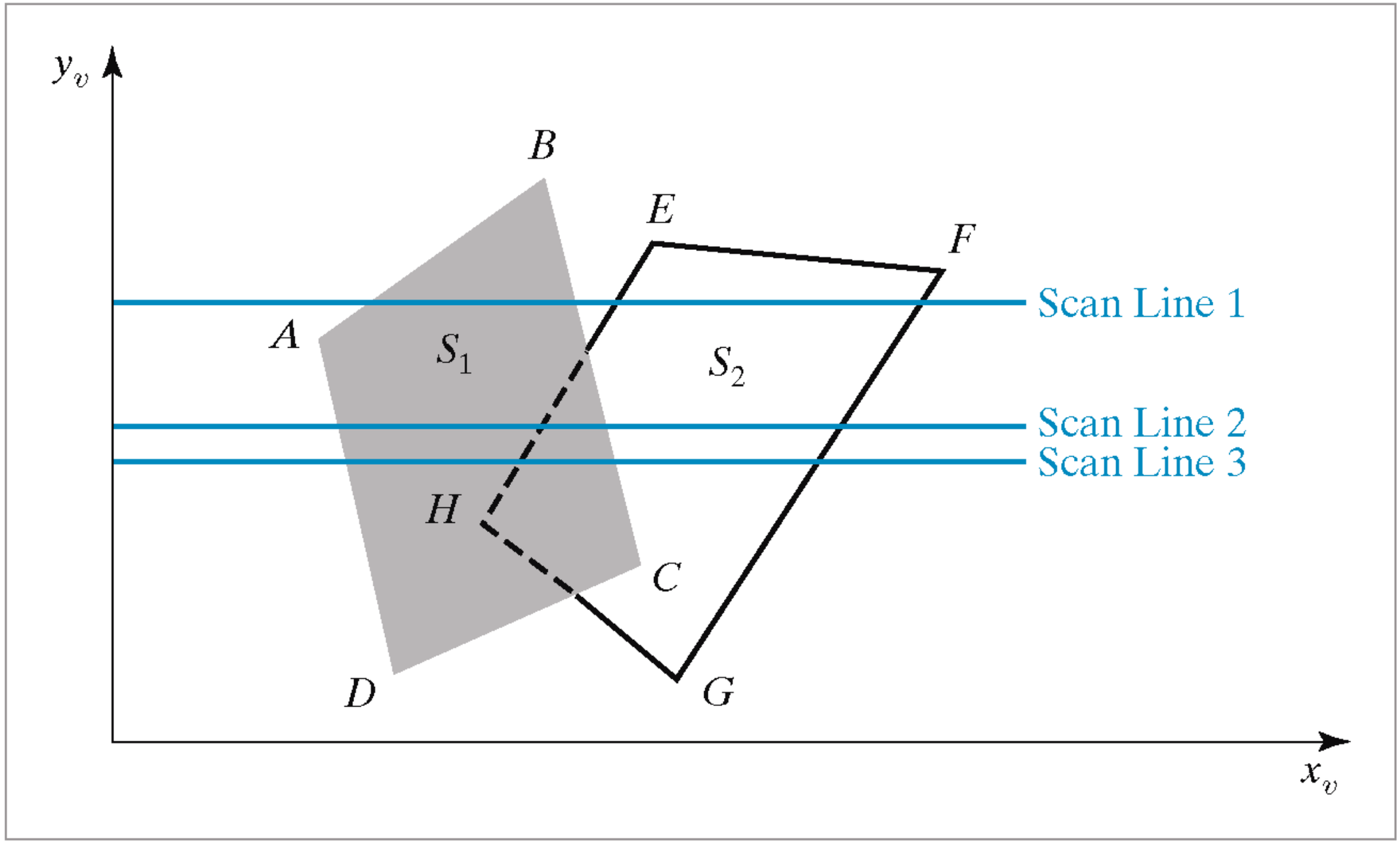
  - a lista contém apenas aquelas arestas que atravessam a *scan-line* e está em ordem crescente da coordenada *x.*

- Além desta lista, uma *flag* é ajustada para cada superfície indicando se uma dada posição (x, y) ao longo da *scan-line* está dentro ou fora da superfície.

Scan-line - algoritmo

- Os *pixels* ao longo de uma *scan-line* são processados partido da esquerda para a direita.
    - na interseção mais à esquerda com uma superfície, a *flag* da superfície é ajustada para 1 (*on / true*).
    - na interseção mais à direita com uma superfícies, a flag da superfície é ajustada para 0 (*off / false*).
- Cálculos de profundidade só são necessários quando mais de uma superfície tem sua *flag* de superfície ativa para uma posição (x,y) da *scan-line*.

Scan-line - exemplo



Scan-line - exemplo

- Para a *scan-line* 1 temos a seguinte lista de arestas ativas: AB, BC, EH, FG.

- Para posições ao longo da *scan-line* 1 que estejam entre as arestas AB e BC, apenas a *flag* da superfícies S_1 está ativa.

  - não há necessidade de cálculos de profundidade.

- De forma semelhante, nas posições entre EH e FG, apenas a flag da superfície S_2 esta ativa.

  - não há necessidade de cálculos de profundidade.

Scan-line - exemplo

- Para as *scan-lines* 2 e 3 temos a seguinte lista de arestas ativas: AD, EH, BC e FG.

- Ao longo da *scan-line* 2, entre as arestas AD e EH, somente a *flag* da superfície S1 está ativa.

- Mas entre EH e BC as *flags* de ambas superfícies estão ativas, sendo necessário calcular a profundidade das superfícies:
  - para este exemplo S_1 está mais próxima e portanto sua cor é utilizada.

- Após a aresta BC, somente a superfície S_2 tem sua *flag* ativa e não são necessários cálculos de profundidade.

Método de Ordem de Profundidade

- A ordem de profundidade é um método de detecção de superfícies visíveis que é baseado tanto em objetos quanto na imagem.

- Basicamente, as duas operações que seguem são executadas:

  - as superfícies são ordenadas em ordem crescente de profundidade.

  - as superfícies são renderizadas segundo uma ordem, tendo início pela superfície com maior profundidade.

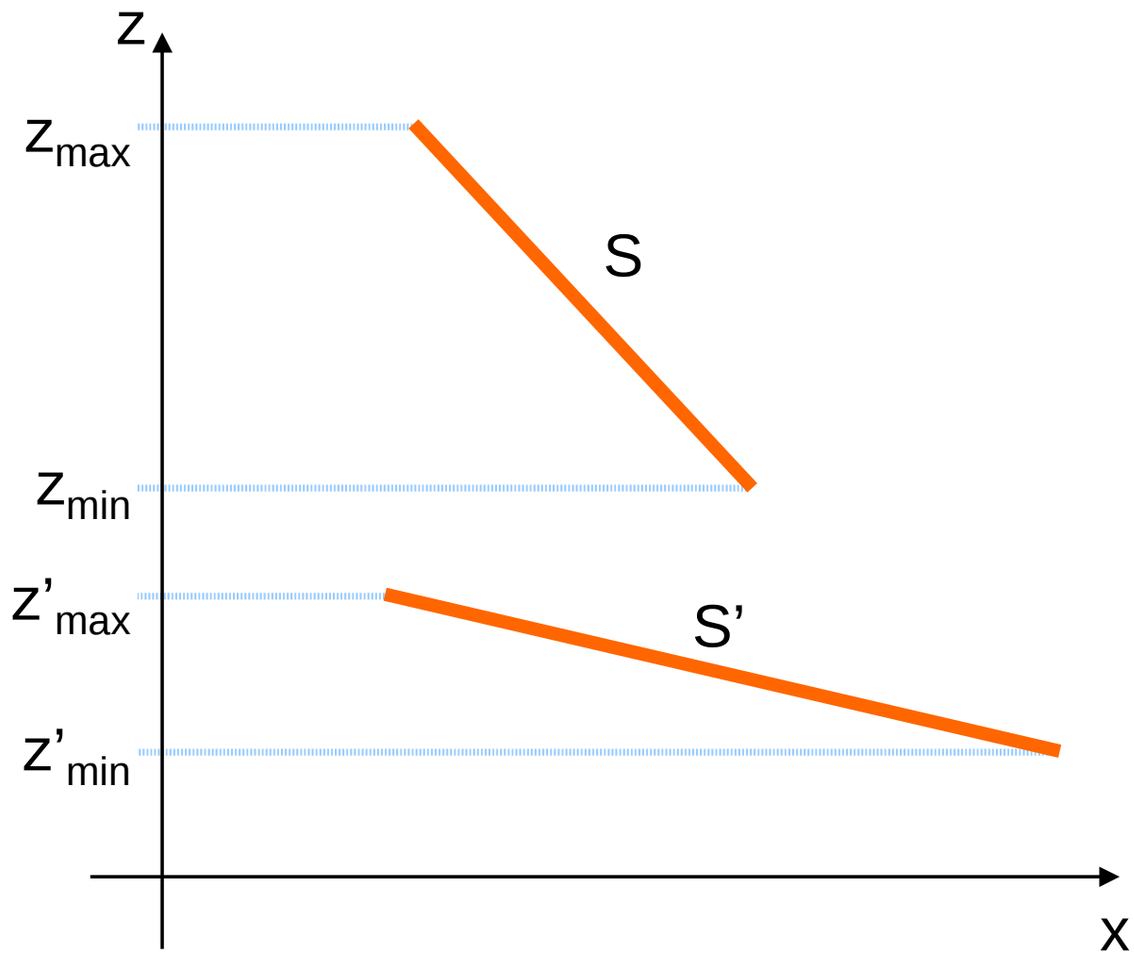
- O método de ordem de profundidade é também conhecido como o método do pintor.

Método de Ordem de Profundidade - algoritmo

- Primeiro, é assumido que estamos visualizando ao longo do eixo z.
- Todas as superfícies da cena são ordenadas de acordo com o menor valor de *z* em cada uma delas.
- A superfície S no final desta lista é então comparada a todas outras superfícies para ver se existe alguma sobreposição de profundidade (em z).
  - se nenhuma sobreposição em z é encontrada, a superfície é renderizada e o processo prossegue com a próxima superfície da lista.

Método de Ordem de Profundidade - algoritmo

- Superfícies sem sobreposição em z



Método de Ordem de Profundidade - algoritmo

- Quando existe alguma sobreposição de profundidade, os seguintes são realizados (em ordem de complexidade):

1. Os retângulos que encapsulam as duas superfícies não se sobrepõem;

2. A superfície S está completamente atrás da superfície de sobreposição com relação ao posição de visualização;

3. A superfície de sobreposição está completamente à frente de S com relação à posição de visualização;

4. A projeção, no plano de visualização, das bordas das duas superfícies não se sobrepõem.

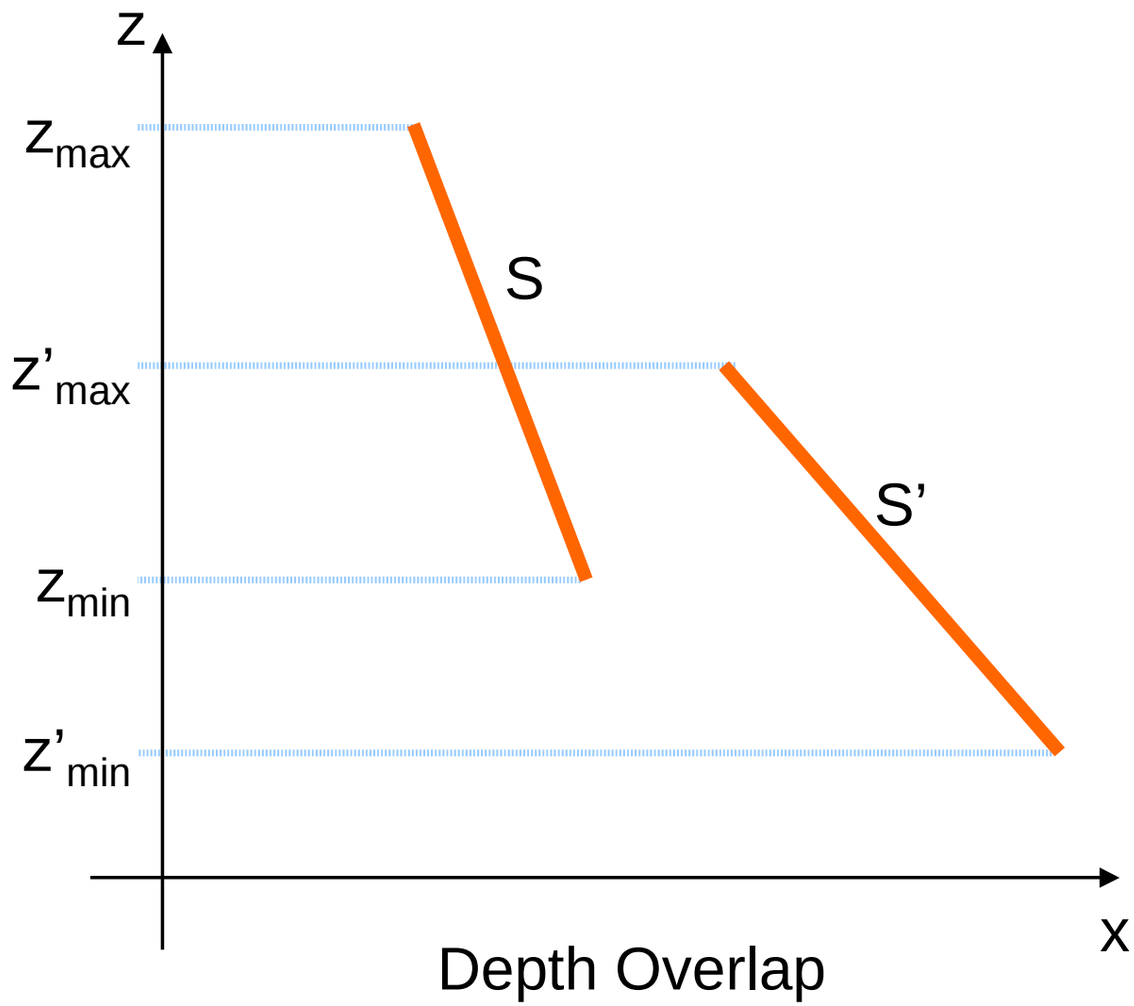
Método de Ordem de Profundidade - algoritmo

- Os testes são realizados na ordem apresentada e, assim que um destes testes passar, isto é, o resultado for verdadeiro, a próxima superfície é Processada.

- Se todos os testes falharem, as superfícies sendo comparadas são trocadas de posição na lista porque a ordem de profundidade entre elas está incorreta.

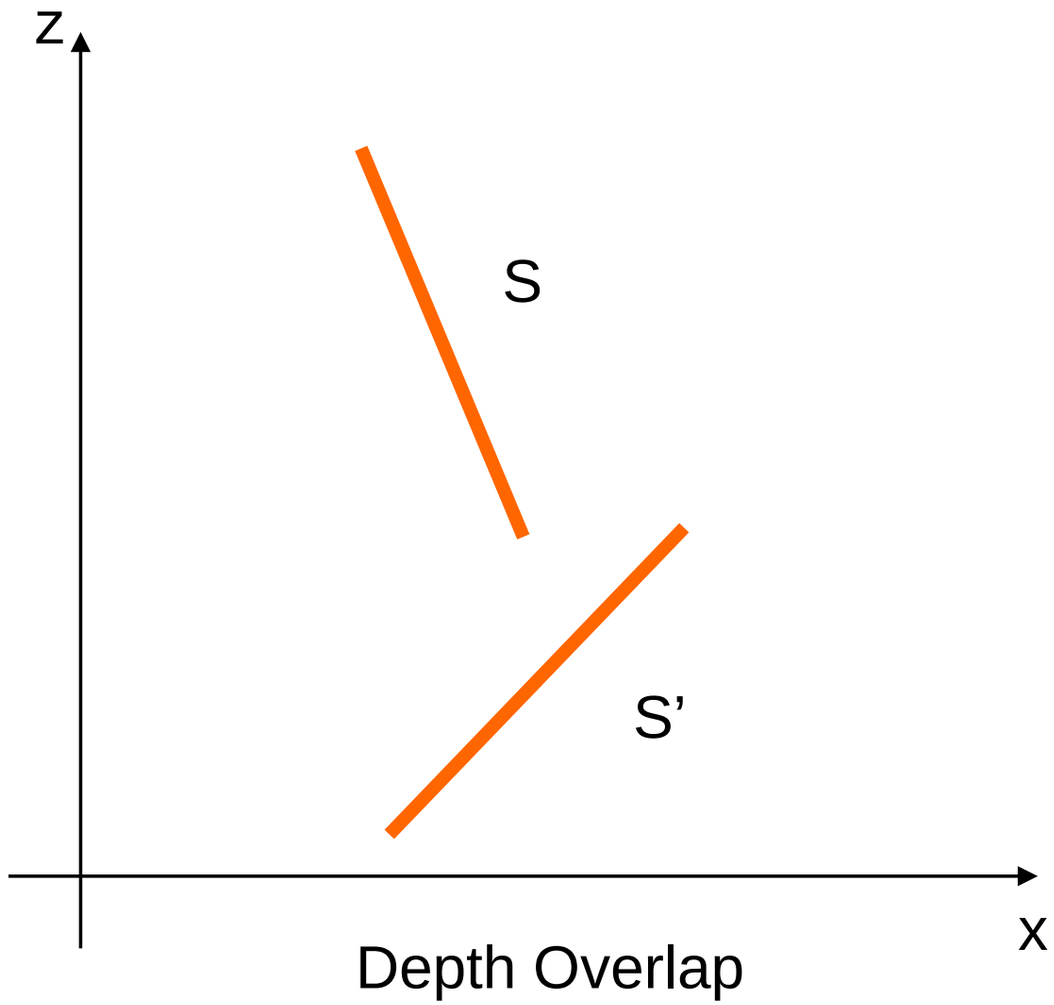
Método de Ordem de Profundidade - Testes

- Teste 1- exemplo
  - duas superfícies com sobreposição em *z*, mas sem sobreposição em *x*;



Método de Ordem de Profundidade - Testes

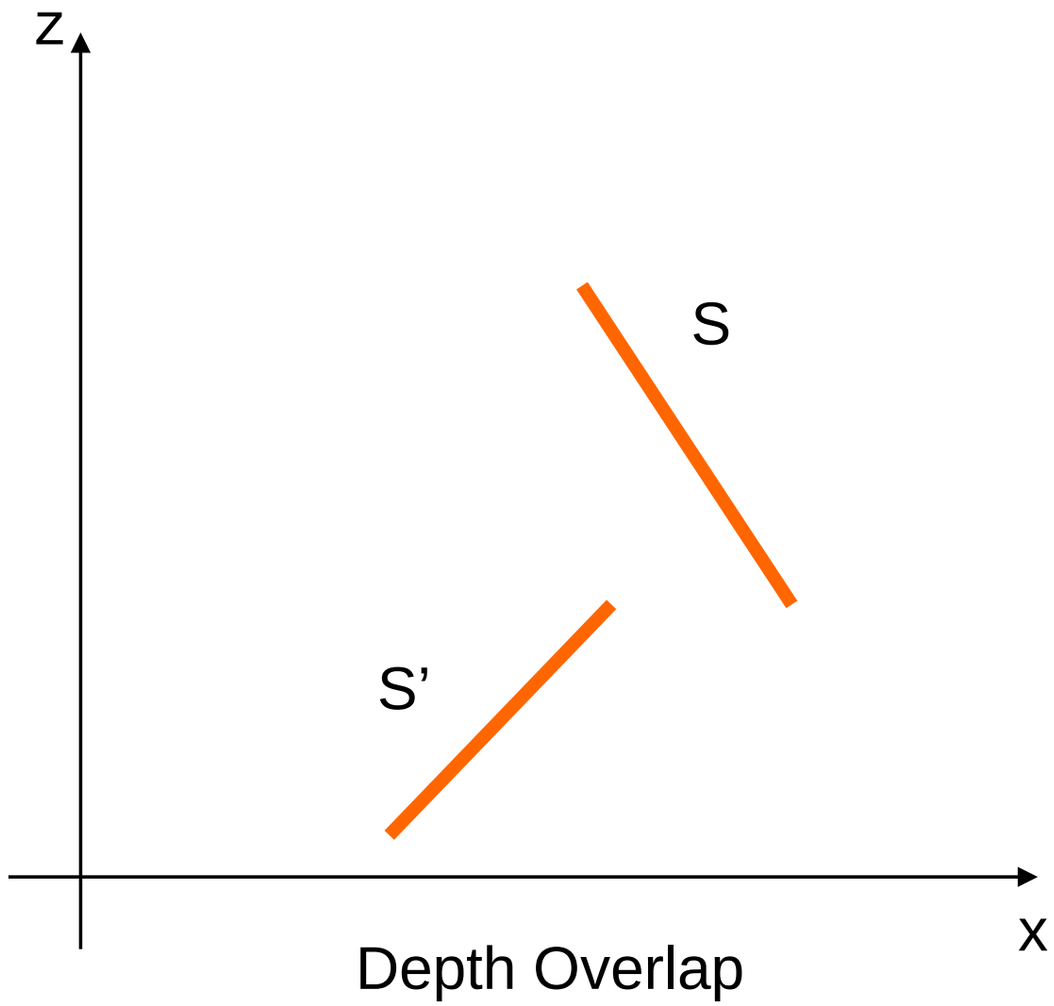
- Teste 2 - exemplo
  - superfície *S* está completamente atrás da superfície de sobreposição *S'*



Método de Ordem de Profundidade - Testes

- Teste 3 - exemplo

  - superfície de sobreposição *S'* está completamente na frente da superfície *S*, mas *S* não está completamente atrás de *S'*.

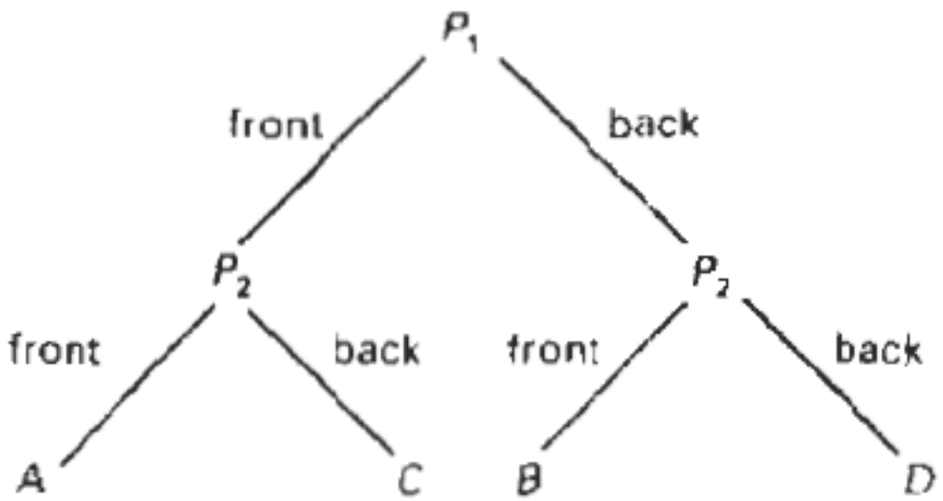


Árvores BSP

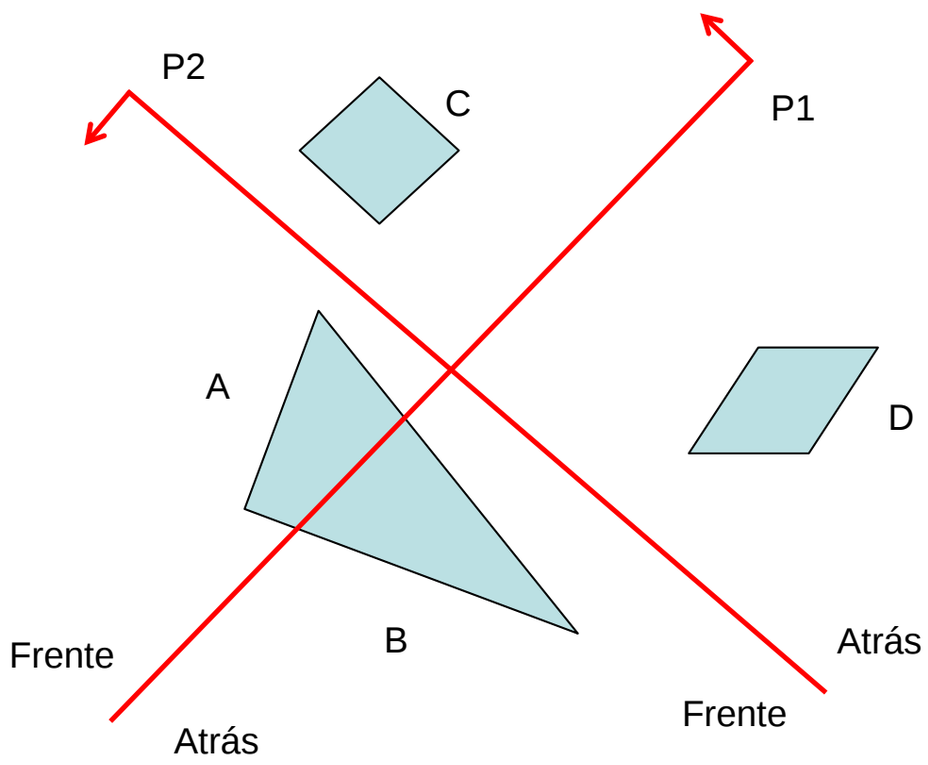
- As árvores BSP (binary space-partitioning) é um método eficiente para determinar a visibilidade de objetos desenhando os mesmos no frame buffer do fundo para a frente (igual ao método do pintor).
    - é particularmente interessante quando o ponto de referência de visualização muda mas, os objetos da cena permanecem fixos.
- O método consiste em subdividir o espaço utilizando planos de partição e identificando quais os objetos estão à frente ou atrás deste plano.
    - estes objetos são organizados em uma árvore onde os objetos mais afastados estão localizados mais à direita.
    - durante a renderização, a árvore é caminhada da direita para a esquerda.

Árvores BSP - exemplo

- Sejam os A, C e D e os planos de partição P1 e P2
  - com relação a P1, C está à frente e D está atrás; o polígono A, é dividido em dois uma vez que o plano P1 o corta ao meio.
  - com relação a P2, A e B estão à frente e C e D estão atrás.



Árvore BSP resultante



Comparação Geral entre Métodos

- Quando poucas superfícies estão presentes na cena, tanto o método de ordem de profundidade quanto as árvores BSP tendem a apresentar melhor desempenho.

- Métodos *scan-line* têm bom desempenho em situações com até alguns milhares de polígonos.

- O método de ordem de profundidade escala linearmente. Assim, para uma baixa quantidade de polígonos, seu desempenho é fraco,

    - mas é muito utilizado quando existem grande quantidades de polígonos (o que não quer dizer que existam muitos objetos).