COMPOSIÇÃO

Remoção de Superfícies Ocultas
Transparência

SUMÁRIO

Remoção de superfícies ocultas

- Algoritmo do Pintor
- Algoritmo Z-Buffer
- Árvores BSP
- Seleção do portal (*portal culling*)
- Seleção das faces traseiras (*back face culling*)

Transparência

- Mistura Alfa (*alpha blending*)
- Transparência "porta de rede" (*screen-door transparency*)
- Transparência Estocástica

REMOÇÃO DE SUPERFÍCIES OCULTAS

Sempre que uma imagem contém objetos ou superfícies opacas, aqueles que se encontram mais próximos do observador e diretamente na linha de vista de outros objetos, vão bloquear a visão destes últimos. As superfícies ocultas devem ser removidas de modo a que a imagem representada no ecrã seja realista.

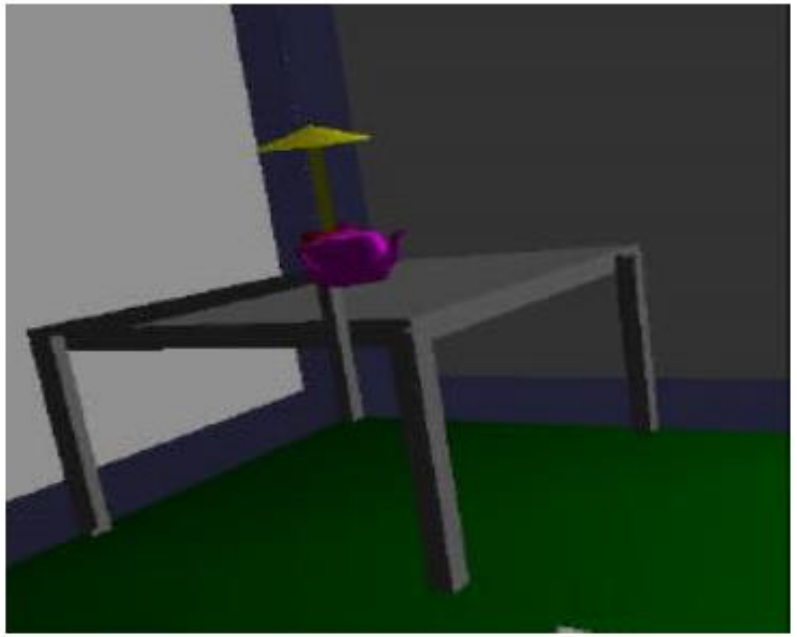
A identificação e a remoção destas superfícies constituem o problema da remoção das superfícies ocultas (RSO) também designado por cálculo de visibilidade.

A sua solução envolve a determinação da profundidade e da visibilidade para todas as superfícies que constituem a cena.

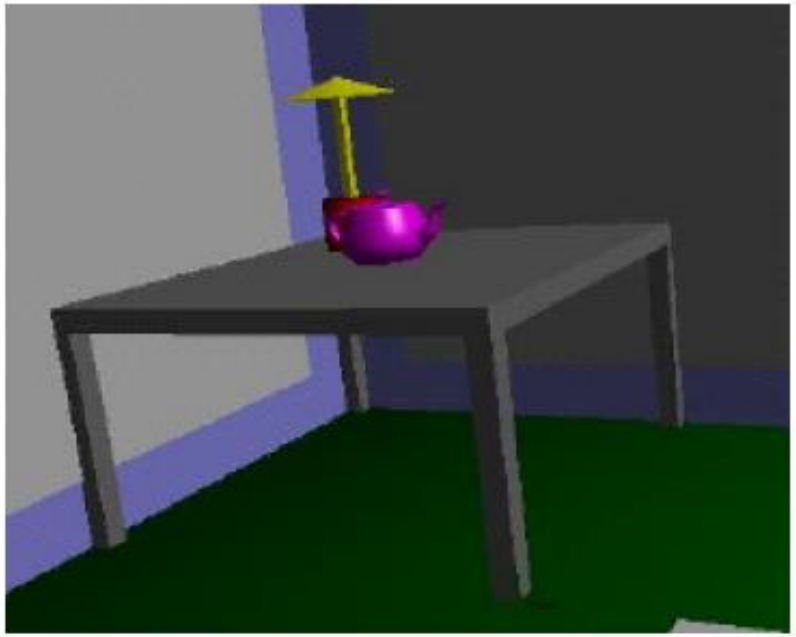
Existem vários algoritmos de Remoção de Superfícies Ocultas (RSO), sendo difícil dizer qual o melhor. Estes diferentes algoritmos têm a sua origem em determinados requisitos específicos.

REMOÇÃO DE SUPERFÍCIES OCULTAS

Visibilidade Incorreta



Visibilidade Correta



ESTRATÉGIAS ALGORÍTMICAS

São conhecidos vários algoritmos de visibilidade (***hidden surface*** ou ***hidden line algorithms***), a maioria dos quais data dos anos 70 e admitem objetos constituídos por faces planas. Ivan Sutherland e colegas apresentaram dez algoritmos de visibilidade para os quais estabeleceram a seguinte taxonomia:

- Algoritmos no espaço Objeto
- Algoritmos no espaço Imagem
- Algoritmos do tipo lista de prioridades

ESTRATÉGIAS ALGORÍTMICAS

Algoritmos no Espaço Objeto

Concentram-se na relação geométrica entre objetos no modelo de descrição da cena, com o fim de determinar que partes desses objetos são visíveis.

Os algoritmos no Espaço Objeto trabalham com uma precisão arbitrária, podendo a imagem final ser aumentada várias vezes sem perda de qualidade.

Inicialmente orientados para dispositivos de visualização vetoriais.

Algoritmos no Espaço Imagem

Determinam para cada quadrícula de imagem, qual o objeto que é visível.

Limitados a uma precisão bastante inferior, imposta pela resolução da superfície de visualização.

Orientados para dispositivos *raster* e, portanto mais suscetíveis ao fenómeno do *aliasing*.

ESTRATÉGIAS ALGORÍTMICAS

Algoritmos no Espaço Objeto

As operações elementares no caso de um algoritmo do tipo Espaço Objeto são complexas devido aos cálculos geométricos envolvidos e portanto de elevada carga computacional.

Já no que concerne à complexidade algorítmica o desempenho realizado por estas duas classes inverte-se.

Algoritmos no Espaço Imagem

As operações elementares envolvem apenas simples manipulações das coordenadas espaciais dos pontos de imagem (por exemplo, comparação de profundidades entre dois pixéis) e consequentemente a carga computacional necessária é substancialmente menor.

ESTRATÉGIAS ALGORÍTMICAS

Algoritmos no Espaço Objeto

Para uma cena de **n** objetos a ser desenhada num dispositivo de visualização com **p** pixéis, a complexidade algorítmica de um algoritmo do tipo Espaço Objeto é **n x n.**

Algoritmos no Espaço Imagem

Para uma cena de **n** objetos a ser desenhada num dispositivo de visualização com **p** pixéis, a complexidade algorítmica de um algoritmo do tipo Espaço Imagem é **n x p.**

ALGORITMOS DO TIPO LISTA DE PRIORIDADES

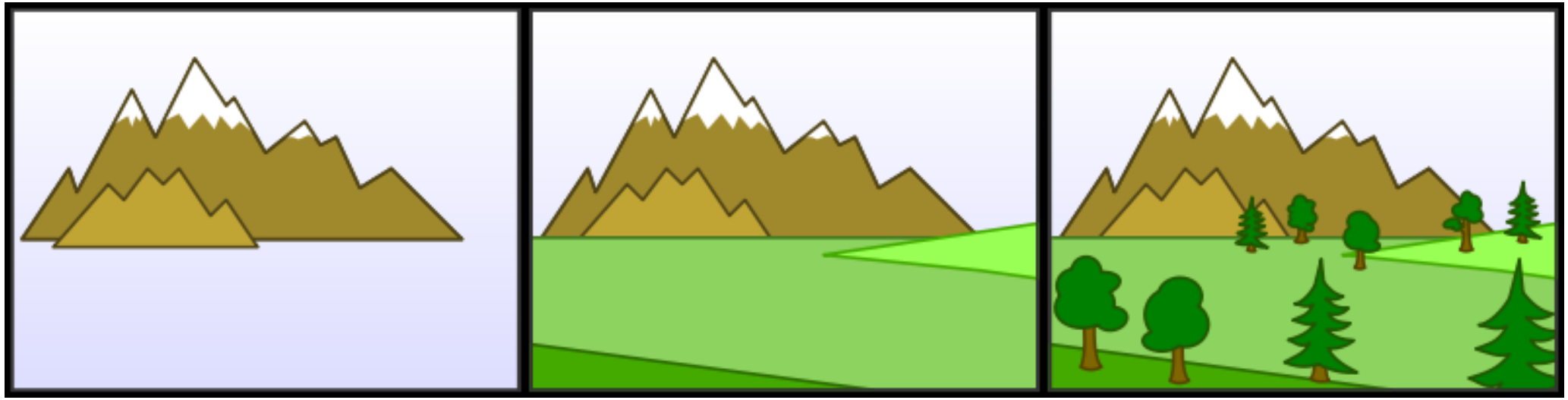
Os algoritmos do tipo Lista de Prioridades ficam situados entre os dois casos anteriores, dado que funcionam parcialmente em cada espaço.

Os cálculos de profundidade são efetuados com precisão elevada, enquanto que as imagens são calculadas com a resolução disponível no sistema de visualização.

ALGORITMO DO PINTOR

Desenha superfícies de trás para a frente, com os polígonos mais próximos a serem “pintados" sobre outros.

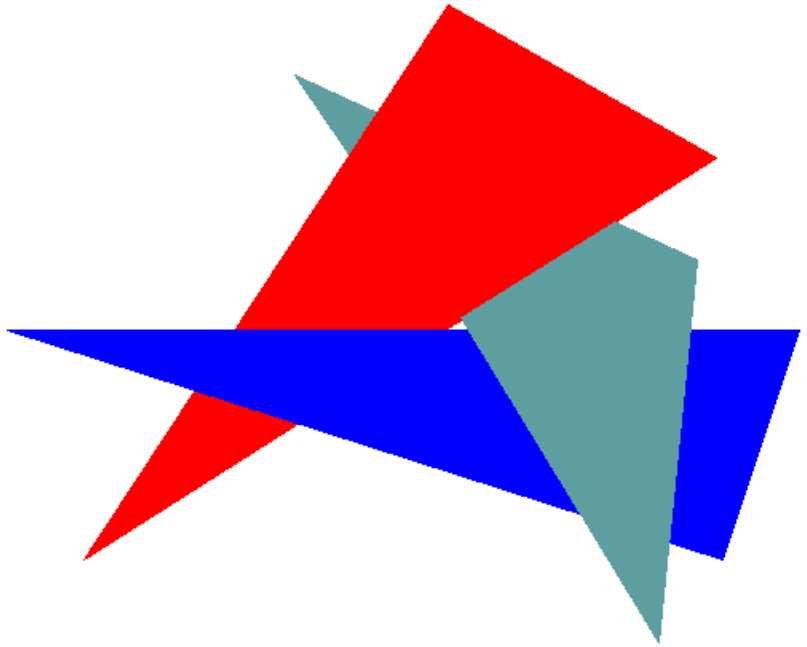
Precisa encontrar a ordem para desenhar objetos



ALGORITMO DO PINTOR

O principal problema é determinar a ordem.

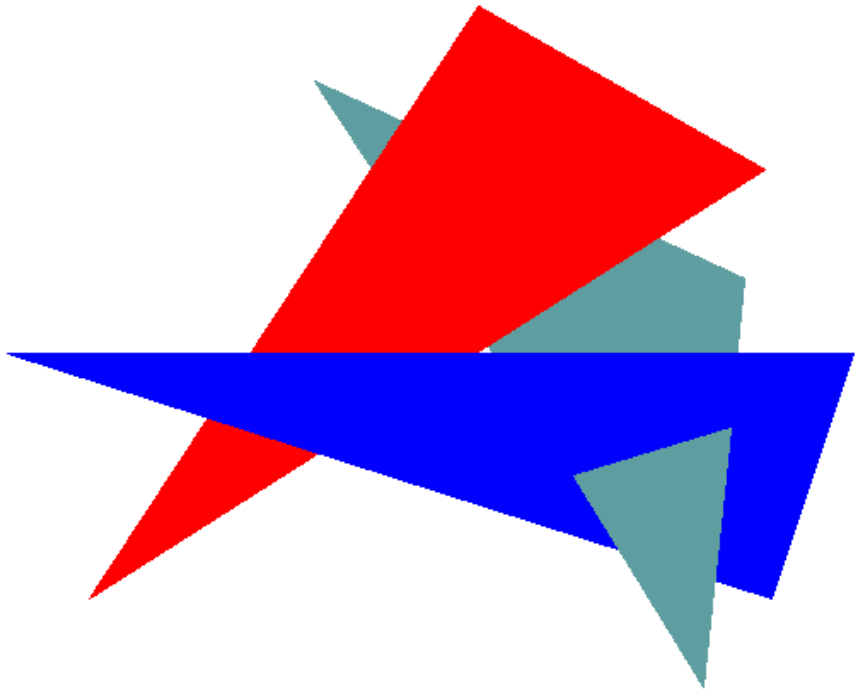
Nem sempre funciona.



ALGORITMO DO PINTOR

Outra situação

É necessário segmentar os triângulos para que eles possam ser classificados



ALGORITMO Z-BUFFER

O algoritmo z-buffer, desenvolvido por Catmull (em 1974*), tornou-se um dos mais utilizados algoritmos de visibilidade.

Este facto deve-se a uma grande simplicidade que lhe permite uma fácil implementação quer em *software* quer em *hardware*.

O algoritmo necessita de duas áreas de memória, uma para a construção da imagem (designada por *frame-buffer*) e outra para armazenamento de profundidades Z (designada por *z-buffer*).

* Na Alemanha no mesmo ano, Wolfgang Straßer também reivindica a autoria do algoritmo

ALGORITMO Z-BUFFER

É efetuado o varrimento linha a linha dos polígonos, um de cada vez, numa ordem arbitrária.

Se, num dado ponto, a profundidade Z de um polígono é menor que a profundidade armazenada anteriormente nas mesmas coordenadas , então o polígono é visível nesse ponto, e o algoritmo substitui , nas duas áreas de memória, os valores anteriores pelos valores calculados para o novo polígono, naquele ponto.

ALGORITMO Z-BUFFER

O conjunto de passos a realizar neste algoritmo são os seguintes:

Inicializar o *Z-buffer* com a profundidade máxima e a memória de imagem (*frame-buffer*) com a cor de fundo

Percorrer todos os polígonos segundo uma ordem arbitrária

Para cada polígono

- Para cada fragmento da posição *(x,y)*
  - Se Z (fragmento) < Z (x, y)
    - WriteZ (x, y, Z(fragmento))
    - WritePixel (x, y, Cor(fragmento))

ALGORITMO Z-BUFFER

O algoritmo *z-buffer* apenas lida com comparação de profundidades pelo que a sua realização é integrada no processo de rasterização de primitivas como, por exemplo, o algoritmo 3D de rasterização de polígonos com coerência de aresta.

Um método baseado em imagem aplicado durante a rasterização.

Abordagem padrão usada em *hardware* e bibliotecas gráficas.

Fácil de implementar em *hardware*.

ALGORITMO Z-BUFFER

Vantagens

Simples de implementar em *hardware*

A memória é relativamente barata

Funciona com quaisquer primitivas

Complexidade ilimitada

Não há necessidade de classificar objetos ou calcular cruzamentos

Desvantagens

Perda de tempo a desenhar objetos ocultos

Erros de precisão em Z (*aliasing*)

DESEMPENHO DO Z-BUFFER

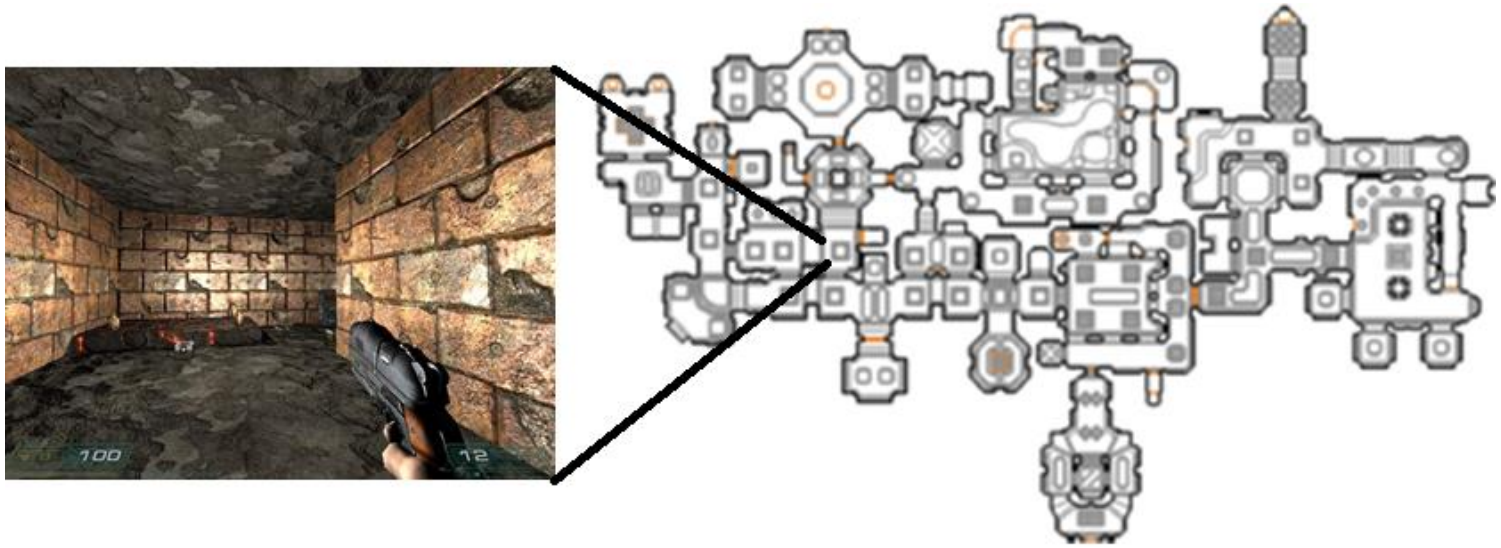
Sobrecarga de memória

Pode ser necessário a sua combinação com outros métodos de recorte para reduzir a complexidade

RENDERIZAÇÃO DE CENAS COMPLEXAS

Não quer desperdiçar recursos renderizando triângulos que não irão contribuir para a imagem final

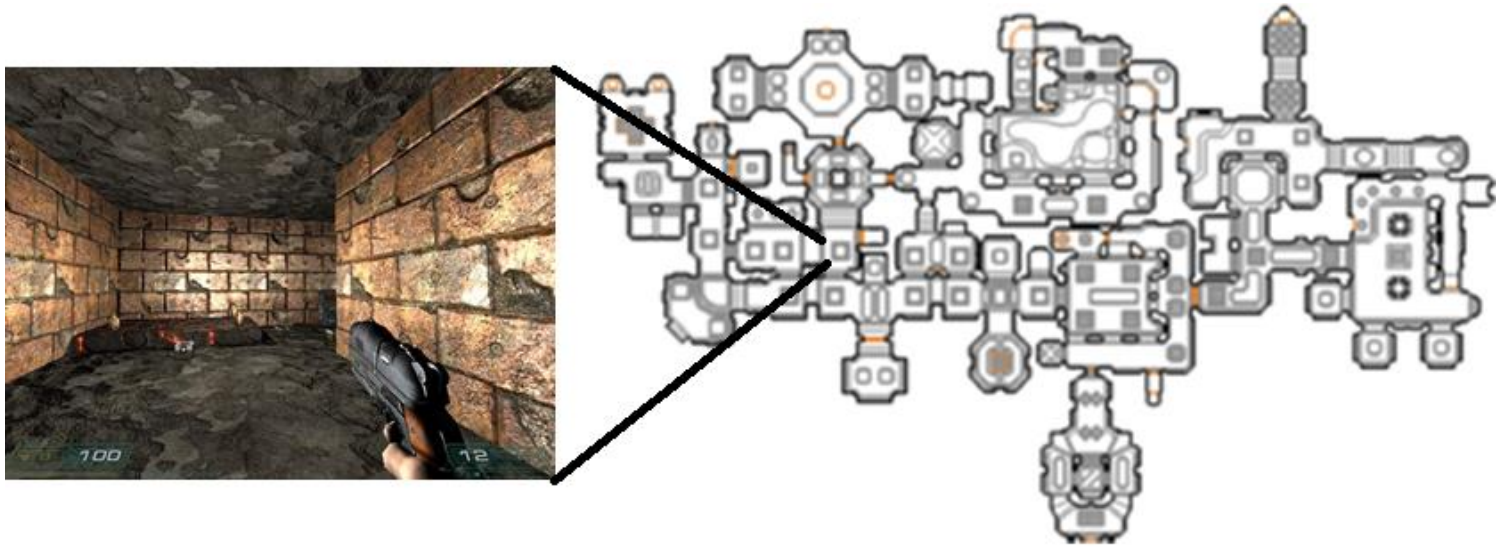
Desenhar cada triângulo requer ciclos de CPU/GPU para calcular iluminação, sombras, etc.



RENDERIZAÇÃO DE CENAS COMPLEXAS

Ordena os polígonos de acordo com a profundidade e desenha apenas aqueles que estão próximos do espectador.

Árvores BSP, seleção de portais (*portal culling*).



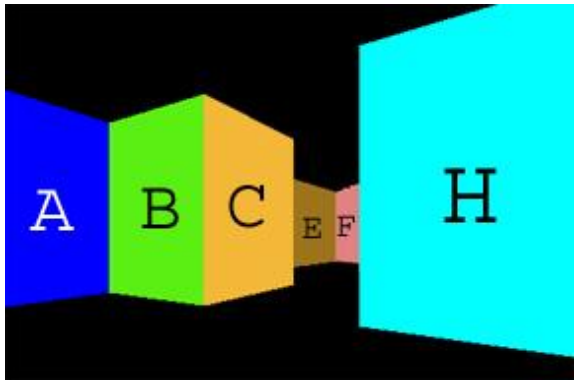
ÁRVORES BSP

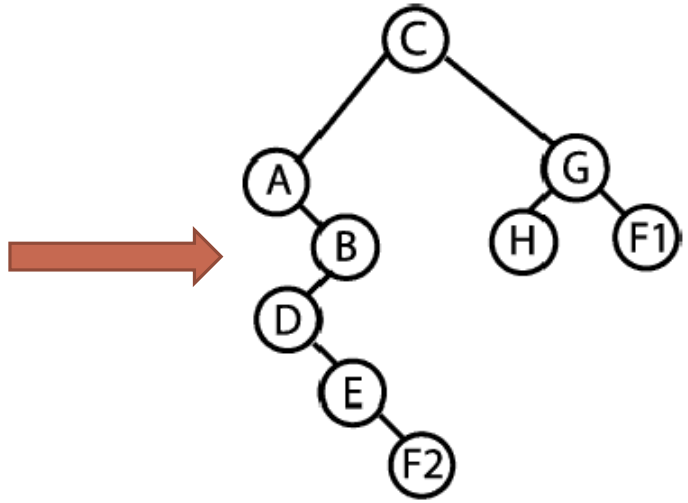
BSP - *Binary space partitioning tree*

Representa a cena com uma árvore

A cena é desenhada atravessando a árvore

Adequado para fazer a renderização de cenas estáticas





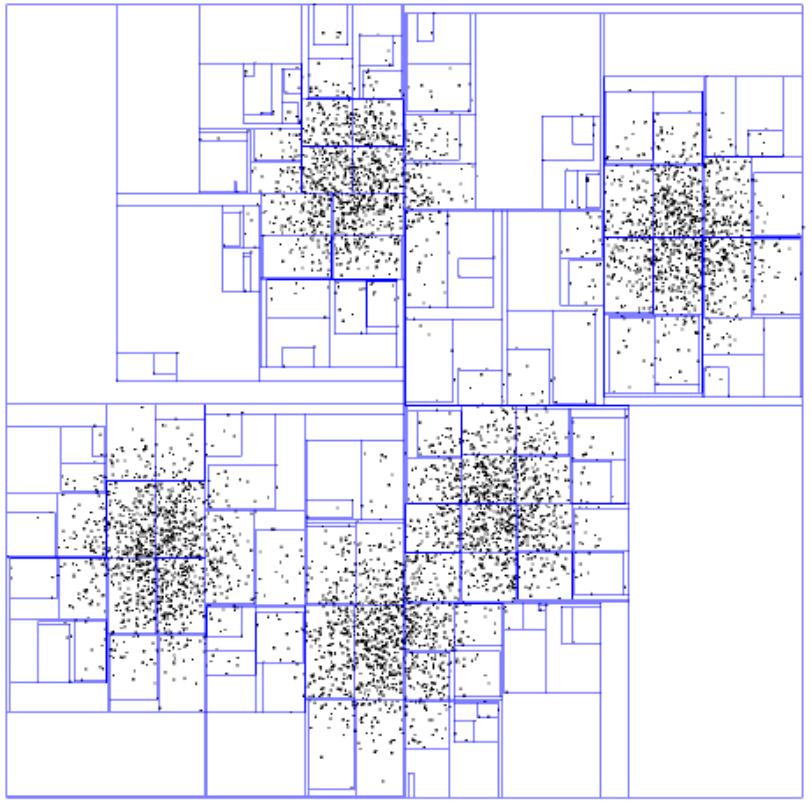
ÁRVORES BSP

Esquemas de divisão:

- Polígono alinhado
- Eixo alinhado

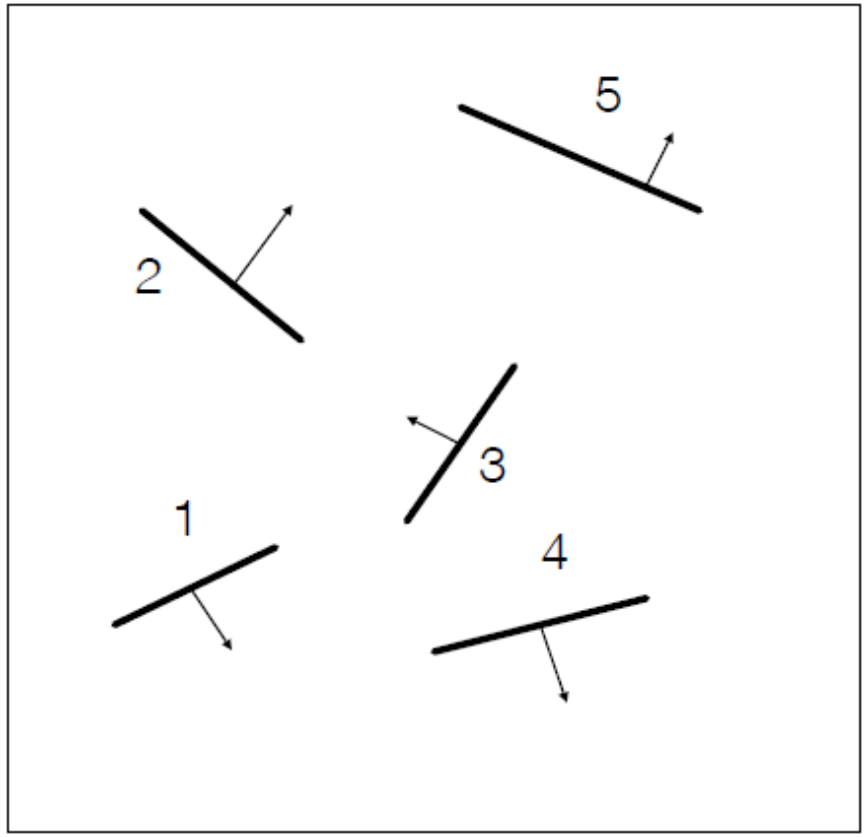
Árvores k-d

Quadtrees, octrees



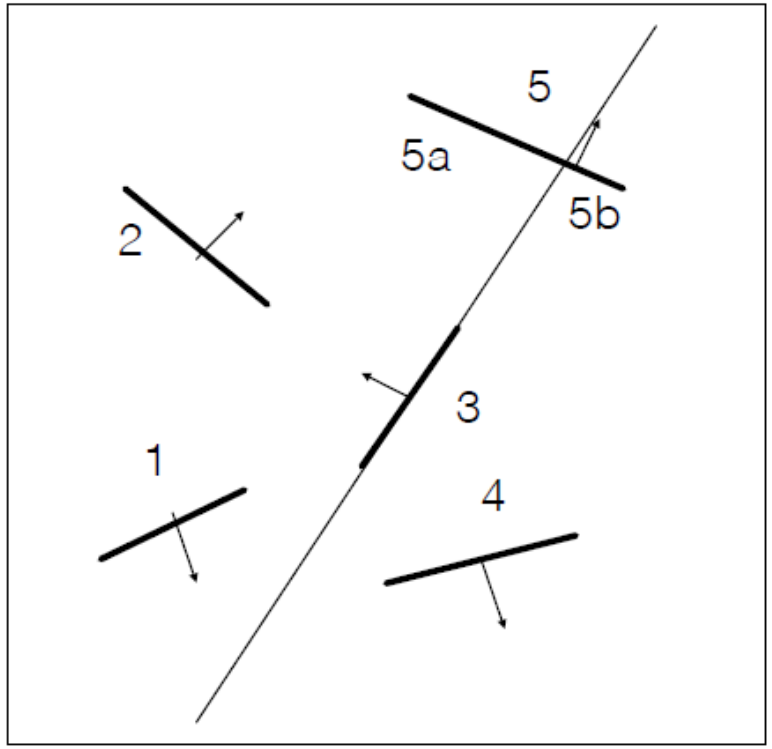
ÁRVORES BSP

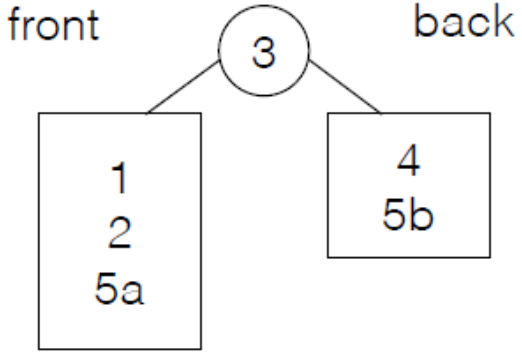
1. Escolher um polígono arbitrariamente
2. Dividir a cena em semi-espaços frontais (em relação à normal) e traseiros.
3. Dividir qualquer polígono que esteja sobre os dois lados.
4. Escolher um polígono de cada lado - dividir a cena novamente.
5. Divida recursivamente cada lado até que cada nó contenha apenas 1 polígono.



ÁRVORES BSP

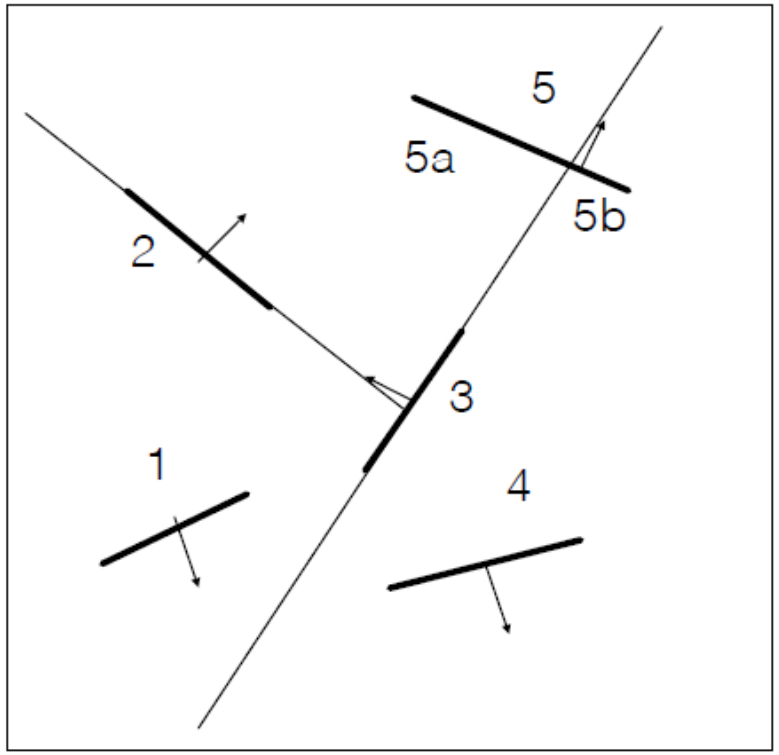
1. **Escolher um polígono arbitrariamente**
2. **Dividir a cena em semi-espaços frontais (em relação à normal) e traseiros.**
3. **Dividir qualquer polígono que esteja sobre os dois lados.**
4. Escolher um polígono de cada lado - dividir a cena novamente.
5. Divida recursivamente cada lado até que cada nó contenha apenas 1 polígono.

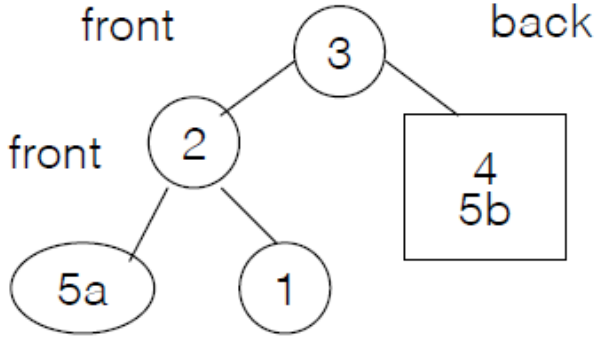




ÁRVORES BSP

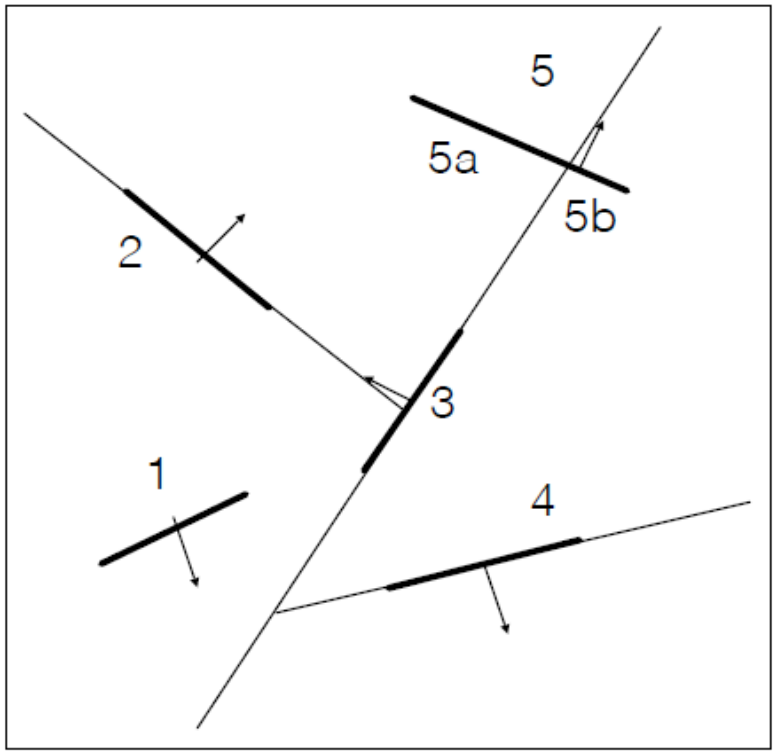
1. Escolher um polígono arbitrariamente
2. Dividir a cena em semi-espaços frontais (em relação à normal) e traseiros.
3. Dividir qualquer polígono que esteja sobre os dois lados.
4. **Escolher um polígono de cada lado - dividir a cena novamente.**
5. Divida recursivamente cada lado até que cada nó contenha apenas 1 polígono.

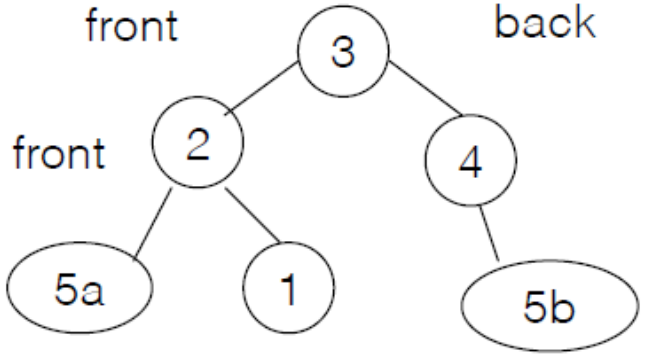




ÁRVORES BSP

1. Escolher um polígono arbitrariamente
2. Dividir a cena em semi-espaços frontais (em relação à normal) e traseiros.
3. Dividir qualquer polígono que esteja sobre os dois lados.
4. Escolher um polígono de cada lado - dividir a cena novamente.
5. **Divida recursivamente cada lado até que cada nó contenha apenas 1 polígono.**





REPRESENTANDO UMA ÁRVORE BSP

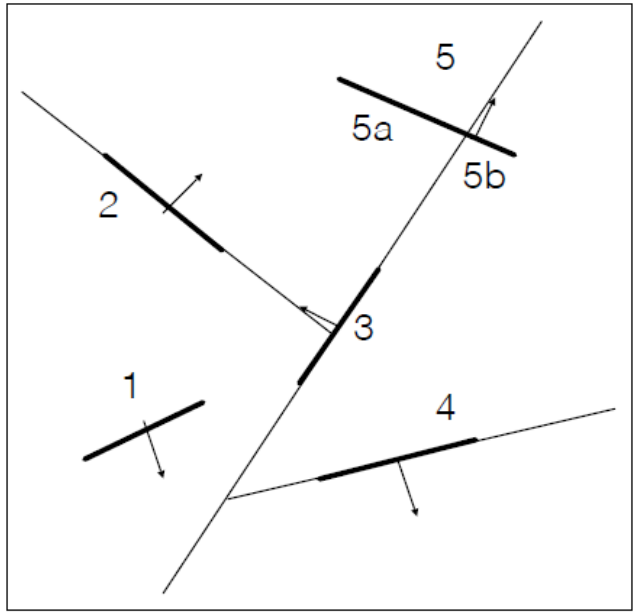
A árvore pode ser atravessada para obter uma ordem dos polígonos para um ponto de vista arbitrário

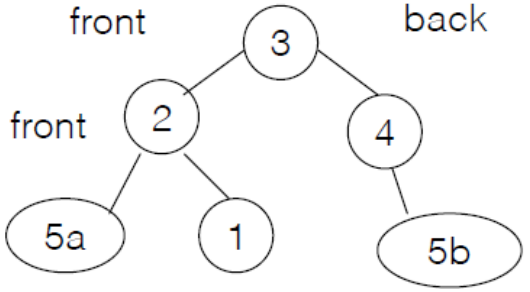
De trás para frente, usando o Algoritmo do Pintor

Da frente para trás - mais eficiente

REPRESENTANDO UMA ÁRVORE BSP: DE TRÁS PARA A FRENTE

1. Começar no polígono raiz.
   - Se o espectador estiver na semi-espaço frontal, desenha os polígonos atrás do polígono raiz primeiro, depois o polígono raiz, e no final os polígonos da frente.
   - Se o espectador estiver no semi-espaço traseiro, desenha primeiro os polígonos que estão à frente da raiz, depois o polígono raiz, depois os polígonos por trás.
   - Desce a árvore recursivamente.
2. Se o espectador estiver no semi-espaço traseiro de um polígono, deve-se fazer a seleção das faces posterior (*back face culling*).
3. Desenhar sempre o lado oposto do espectador primeiro.





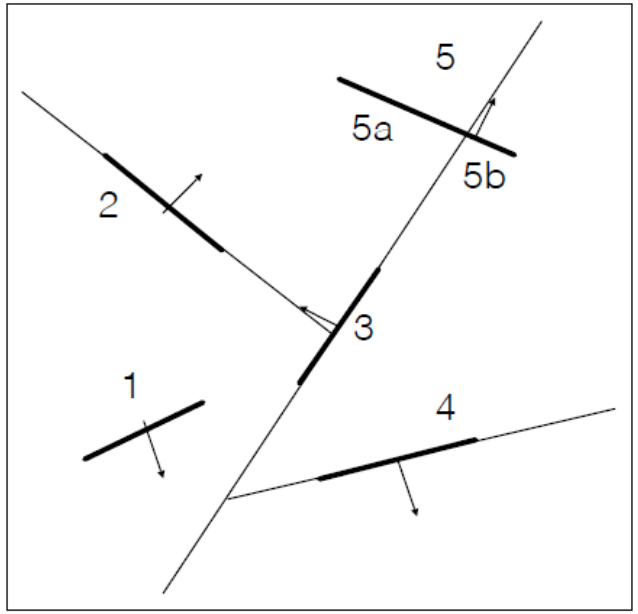
REPRESENTANDO UMA ÁRVORE BSP: DA FRENTE PARA TRÁS

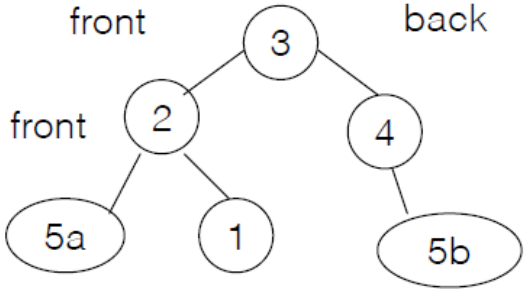
A renderização de trás para a frente vai resultar em muito excesso de desenho.

A travessia da frente para trás é mais eficiente.

Regista quais as regiões do ecrã que foram preenchidas.

Termina quando todas as regiões estiverem preenchidas.





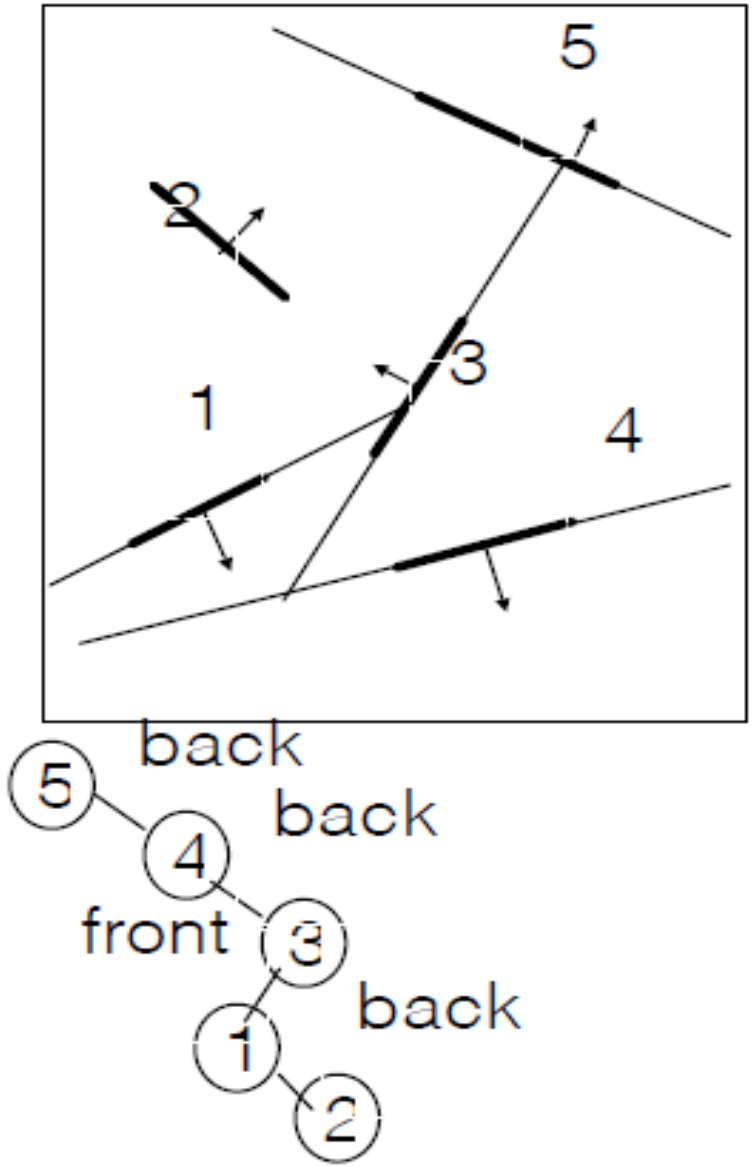
ÁRVORES BSP

A geração da árvore requer muito esforço computacional.

- Há a necessidade de produzir uma árvore equilibrada.
- Vai ser necessário intersectar polígonos para se proceder à sua divisão.

É fácil, verificar a visibilidade depois da árvore ser construída.

Muito eficiente quando a cena não muda frequentemente.



ÁRVORES BSP

Método combinado com Z-buffer.

Renderiza objetos estáticos (da frente para trás) usando o *buffer* de profundidade (Z-buffer).

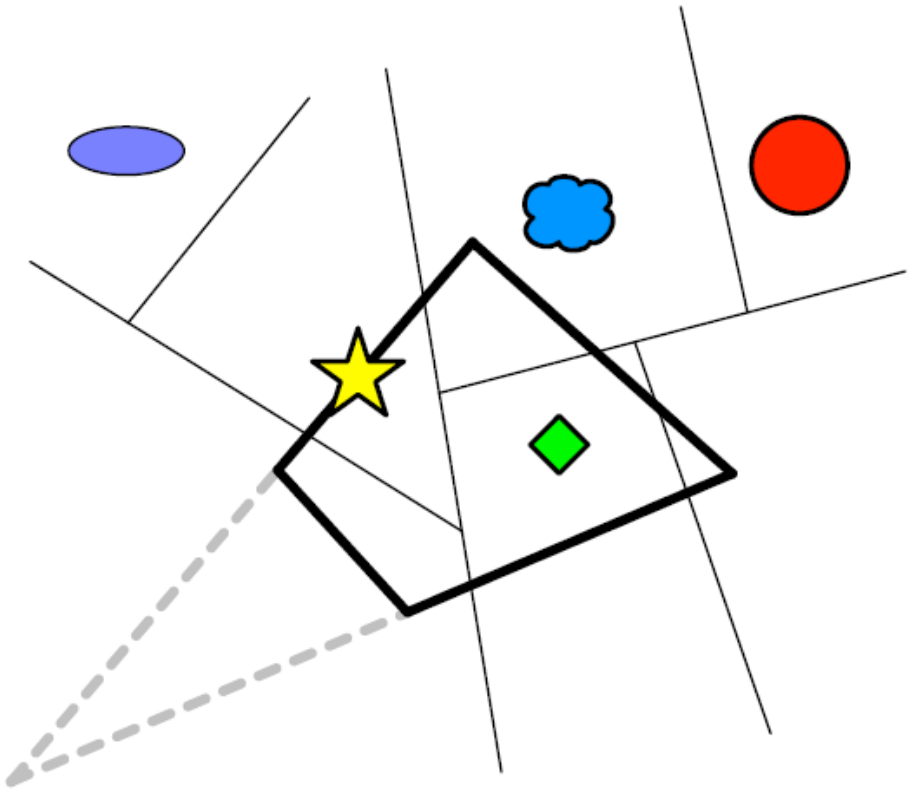
No final, desenha os objetos dinâmicos.

SELEÇÃO DE VISIBILIDADE DE ÁRVORES BSP

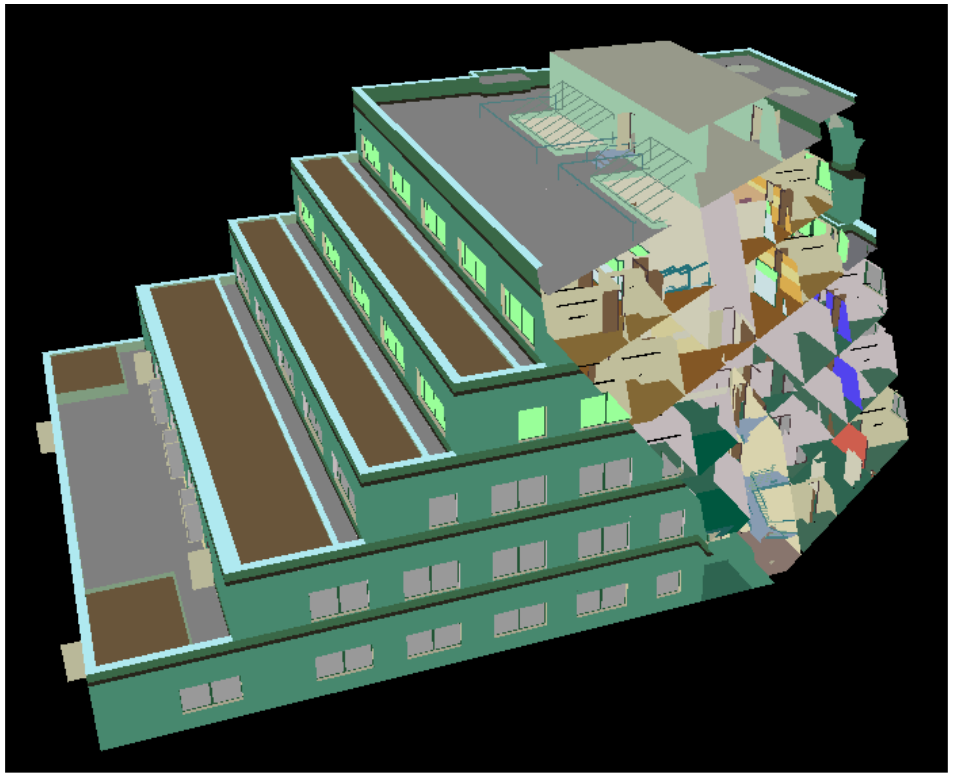
As árvores BSP podem ser usadas para selecionar polígonos que caem fora do volume de visualização.

- Se o plano intersectar o volume, subdivide para ambos os filhos.
- Se o volume de visualização estiver de um dos lados do plano, selecionar objetos do outro lado.

Também é possível realizar o processo recorrendo a *octrees*



EXEMPLO: CENAS ARQUITECTURA

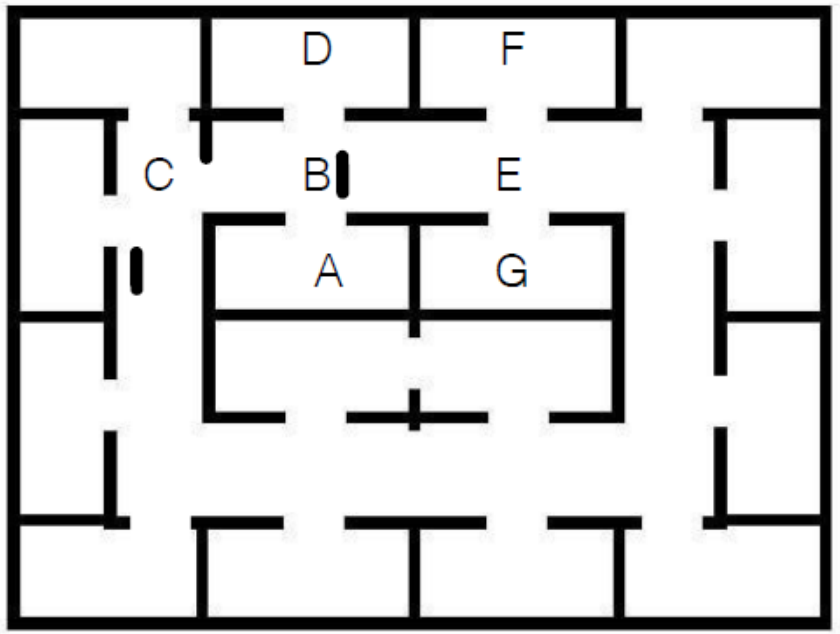


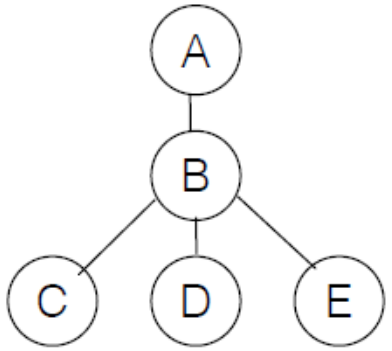
SELEÇÃO DE PORTAIS

Modela a cena como um grafo:

- Nós: células (ou salas)
- Bordas: Portais (ou portas)

O grafo fornece um conjunto potencialmente visível





1. Renderiza a sala
2. Se o portal para a próxima sala for visível, torne a sala ligada numa região do portal
3. Repita o processo ao longo do grafo de cena

CLASSIFICAÇÃO DO ESPAÇO OBJETO E DO ESPAÇO DA IMAGEM

Técnicas do Espaço Objeto - aplicadas à geometria da malha:

- Árvores BSP com o Algoritmo do Pintor, seleção de portais

Técnicas do Espaço Imagem - aplicadas quando os pixéis são desenhados:

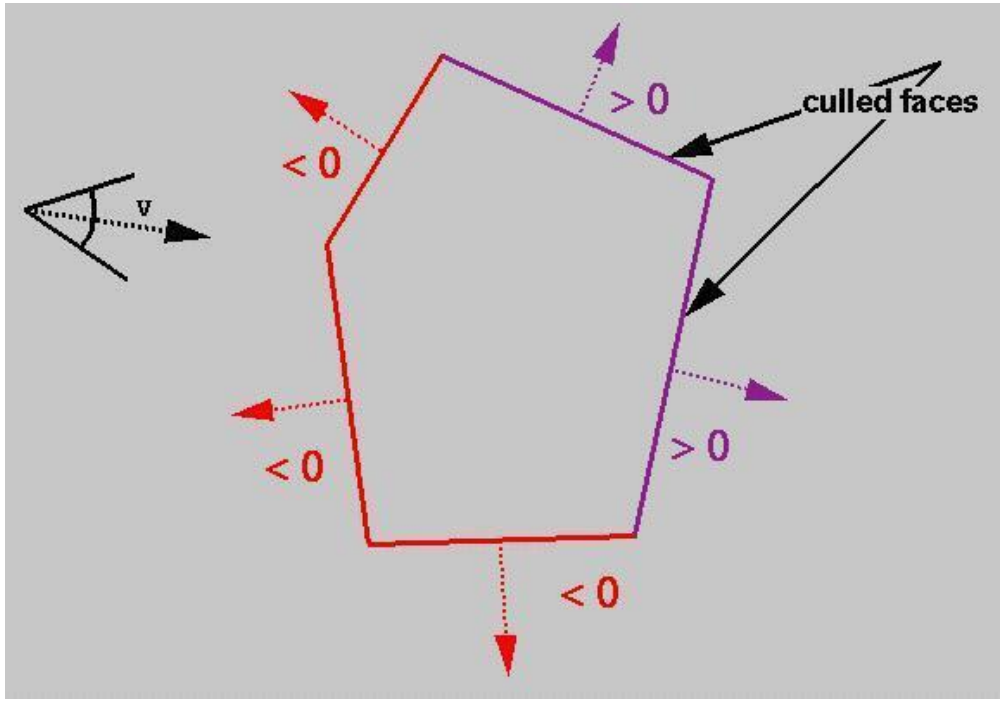
- Z-Buffer

SELEÇÃO DAS FACES TRASEIRAS

Não são desenhados polígonos voltados para a outra direção

Testa a componente z das normais da superfície.

- Se negativo - desenha, já que o normal aponta para longe do espectador.
- Ou, se N.V> 0, está-se a visualizar a face traseira, logo o polígono vai ficar obscurecido.



REMOÇÃO DE SUPERFÍCIES OCULTAS

O algoritmo Z-buffer é fácil de implementar em hardware e é uma técnica padrão.

Há a necessidade de combinar o Z-buffer com uma abordagem baseada em objeto quando existem muitos polígonos - árvores BSP, seleção de portais.

A travessia da frente para trás reduz o custo.

TRANSPARÊNCIA

Mistura Alfa (*alpha blending*)

Transparência "porta de rede"

(screen-door transparency)

Transparência Estocástica

TRANSPARÊNCIA

Às vezes pretende-se desenhar objetos transparentes

Combina-se a cor dos objetos visíveis em cada pixel

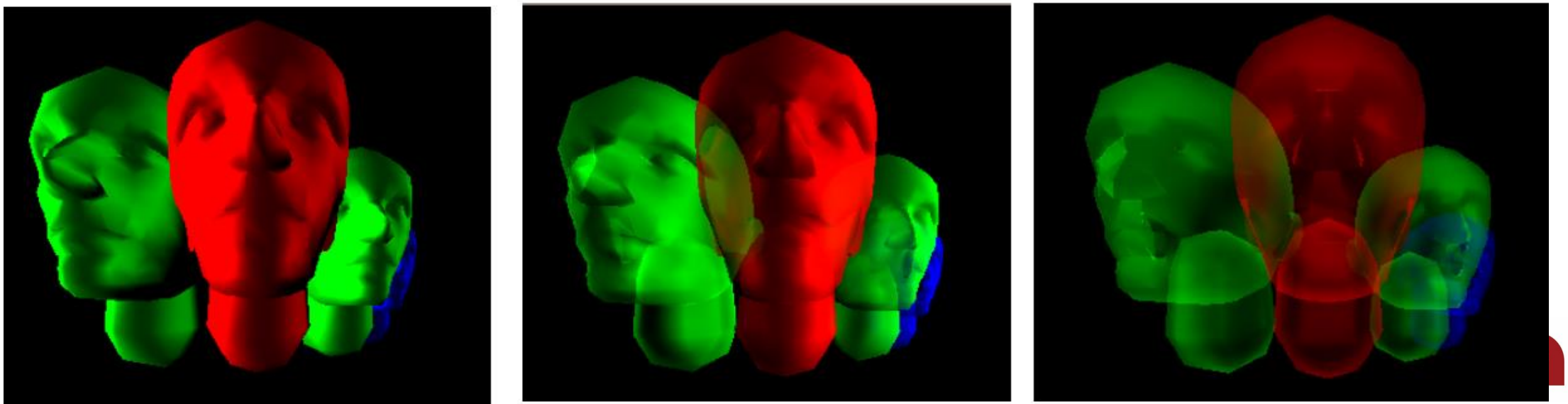
- Mistura Alfa (*alpha blending*)
- Transparência "porta de rede" (screen-door transparency)

ALPHA BLENDING

Valores alfa descrevem a opacidade de um objeto

- 1 significa totalmente opaco
- 0 significa totalmente transparente

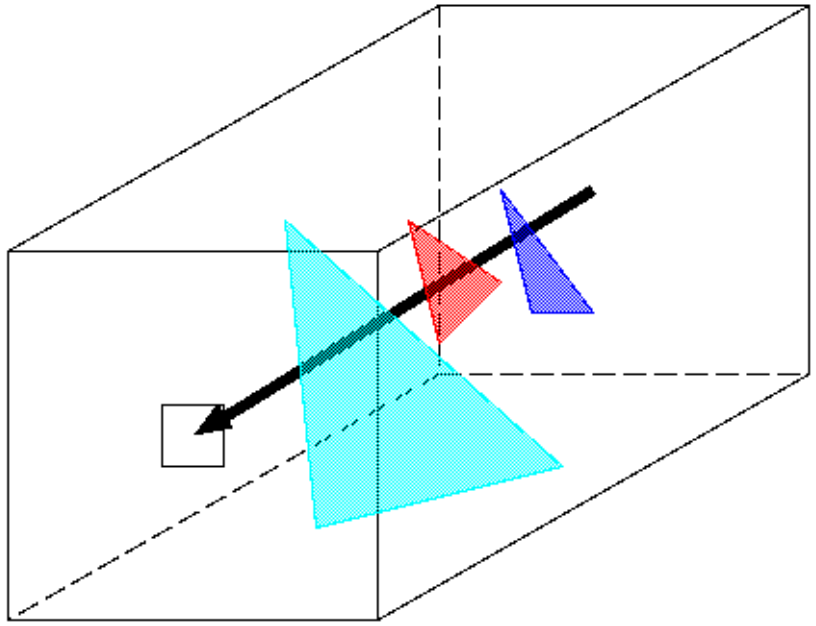
$\alpha=1.0$ $\alpha=0.5$ $\alpha=0.2$



ORDENAR POR PROFUNDIDADE

A profundidade e cor de todos os fragmentos que serão projetados no mesmo pixel é armazenado numa lista.

As cores vão ser misturadas de trás para frente.



MISTURA DE CORES

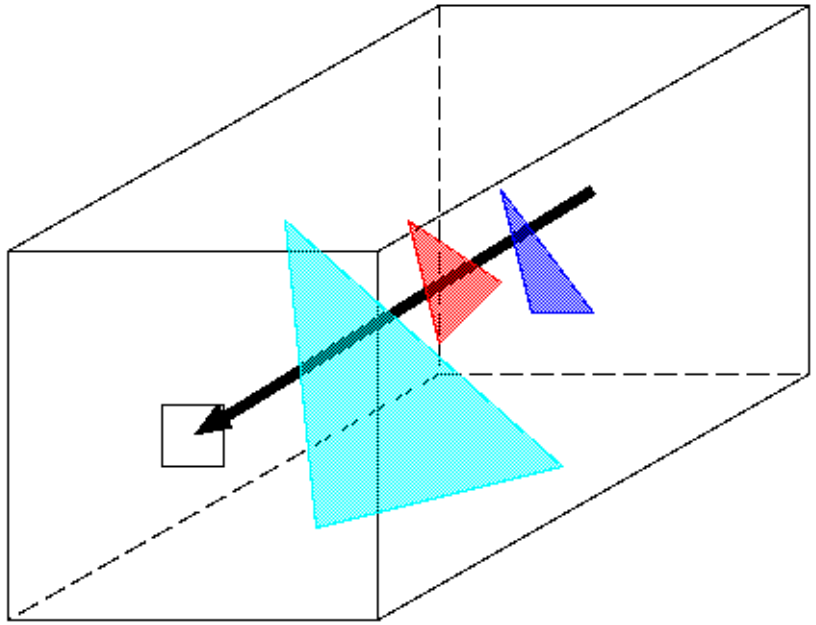
As cores são combinadas da seguinte maneira:

$$C_o = \alpha C_s + (1 - \alpha) C_d$$

$C_o =$ Nova cor no *pixel*

$C_s =$ Cor do objeto transparente

$C_d =$ Cor atual do pixel

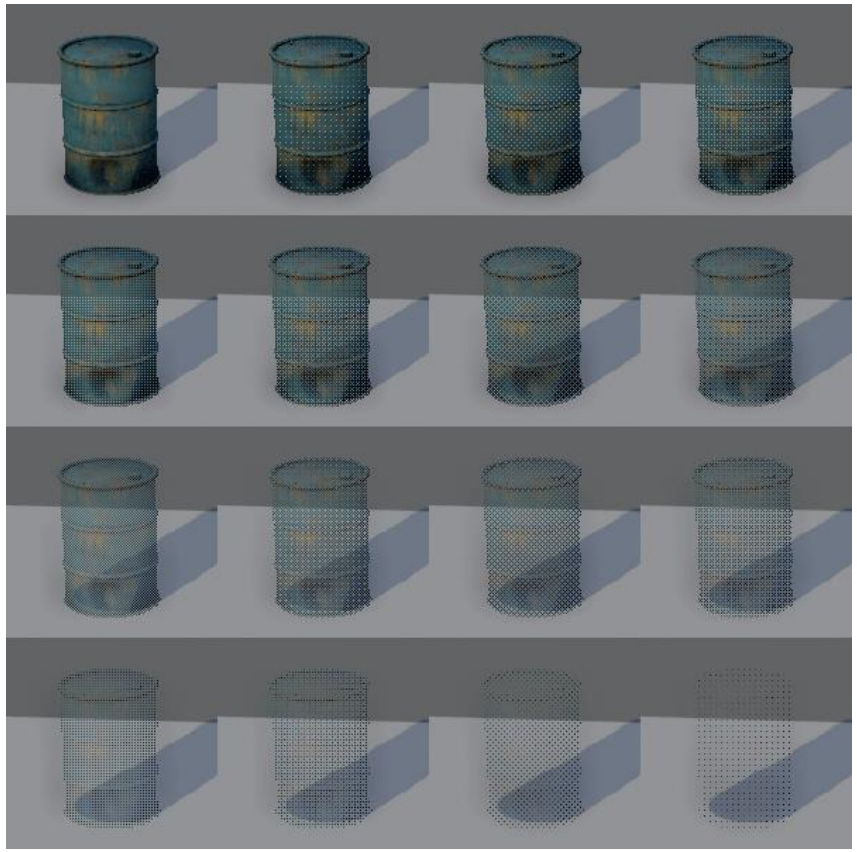


ORDENAMENTO

O ordenamento é caro (árvore BSP)

O ordenamento por pixel é muito caro

Uma solução mais rápida - transparência "porta de rede" *(screen-door transparency)*



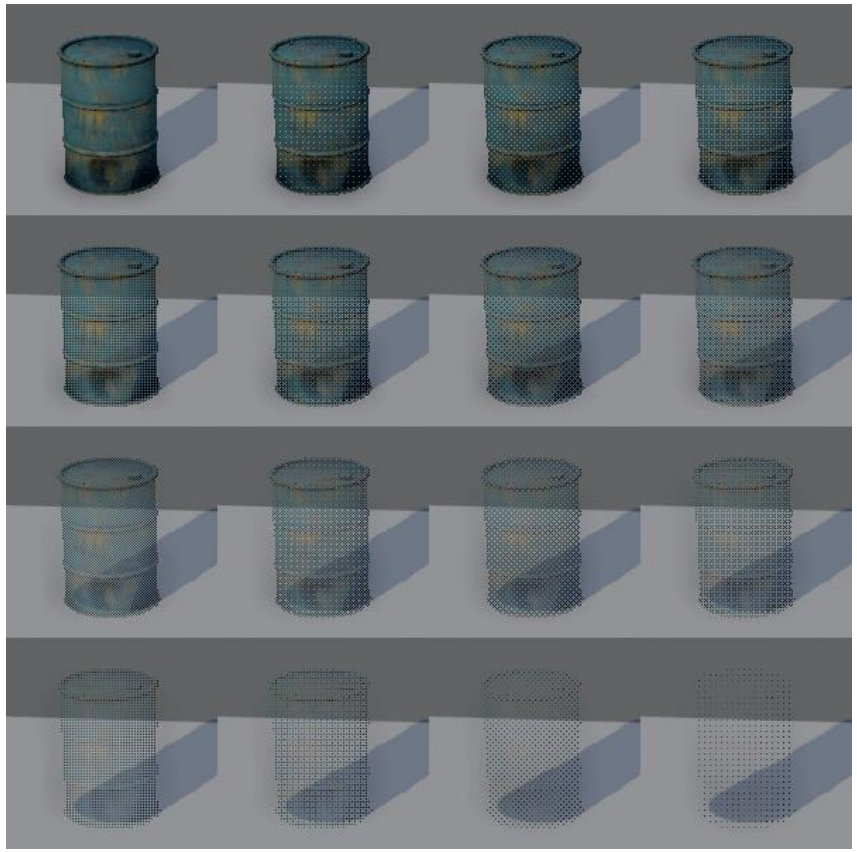
TRANSPARÊNCIA "PORTA DE REDE"

O objeto é sólido, mas é desenhado com furos usando um padrão pontilhado (quadriculado) como uma porta de rede.

A proporção de pixéis desenhados é igual ao valor alfa.

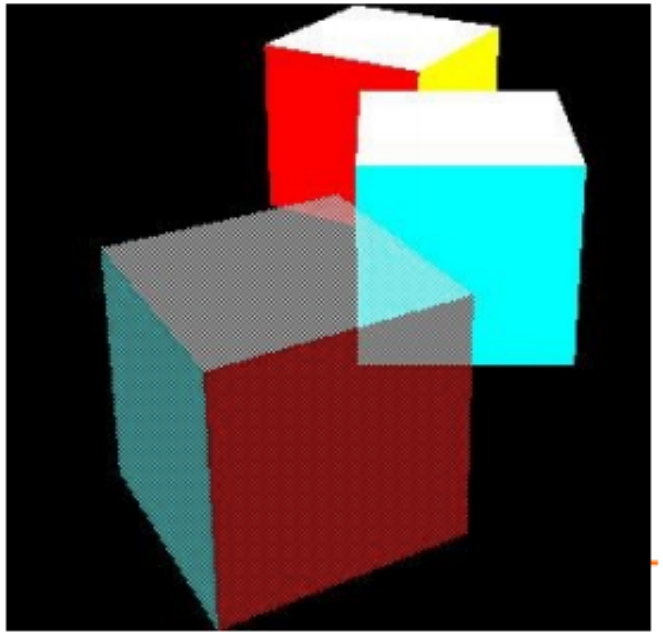
Não há necessidade de realizar o ordenamento, os objetos podem ser desenhados em qualquer ordem.

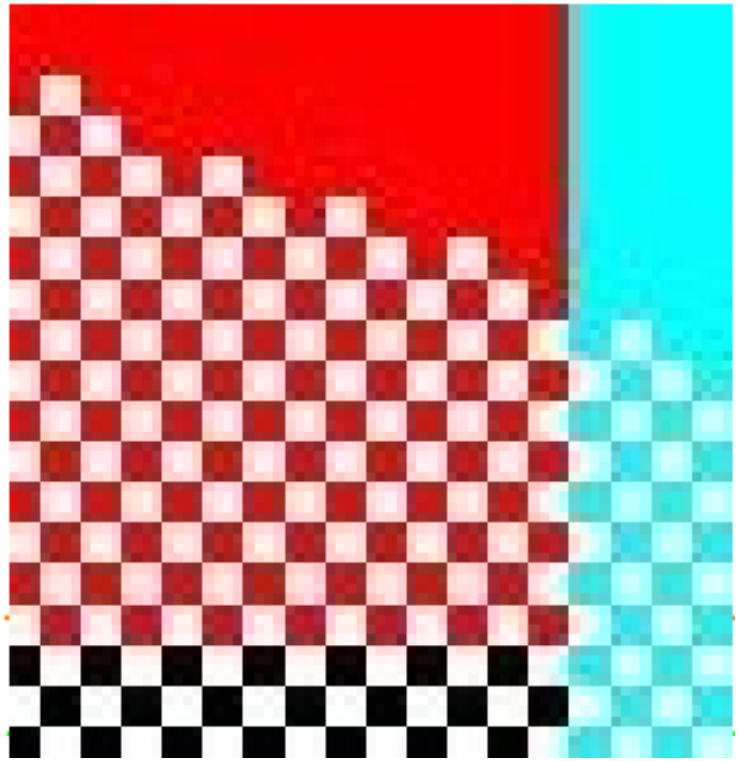
O Z-buffer pode lidar com as sobreposições de superfícies translúcidas.



TRANSPARÊNCIA "PORTA DE REDE"

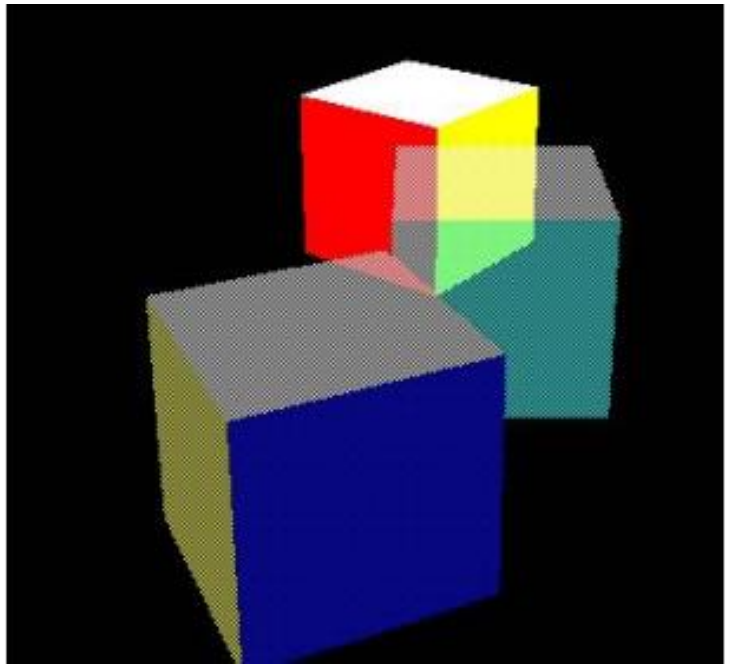
$$\text{alpha} = 0.5$$

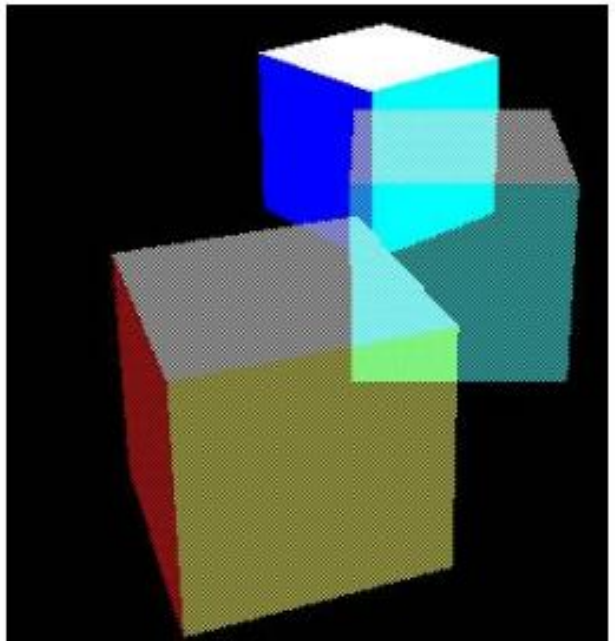




TRANSPARÊNCIA "PORTA DE REDE"

Com um objeto transparente sobre outro, o objeto transparente pode bloquear tudo por trás dele quando for usado o mesmo tipo de padrão pontilhados.





TRANSPARÊNCIA ESTOCÁSTICA

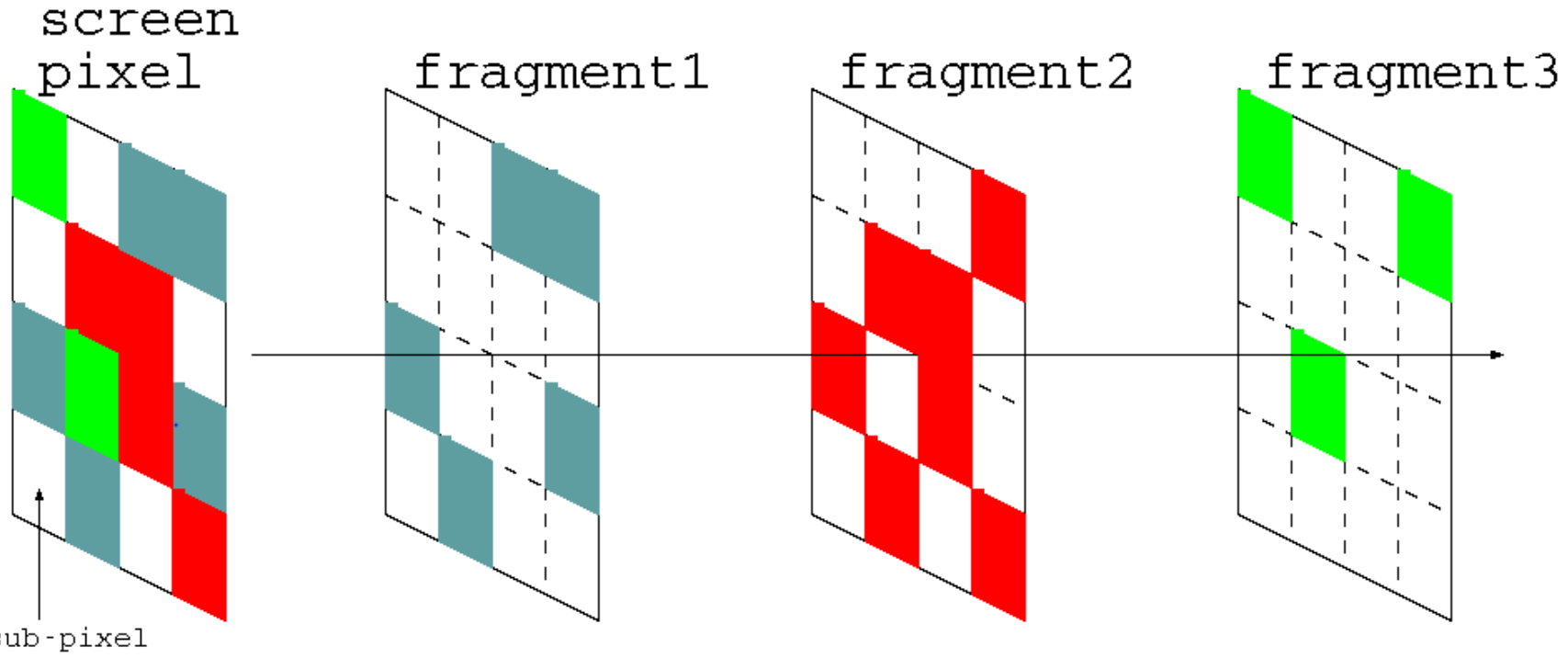
Usando multisampling, os subpixéis são desenhados e a cor do pixel é calculada usando a média dessas cores.

Usa um padrão aleatório pontilhado de sub-pixel

TRANSPARÊNCIA ESTOCÁSTICA

Não é necessário ordenamento

A cor final de um pixel é calculada pela média das cores dos sub-pixéis




HA HA! I HAVE TURNED MYSELF INVISIBLE!
BY REMOVING MY CLOTHING, I CAN PERPETRATE ANY CRIME UNDETECTED!
I HAVE COMPLETE FREEDOM! I CAN GET AWAY WITH ANYTHING!
CALVIN! WHAT ON EARTH ARE YOU DOING IN THE COOKIE JAR WITHOUT YOUR CLOTHES ON?!?
WATTERSON