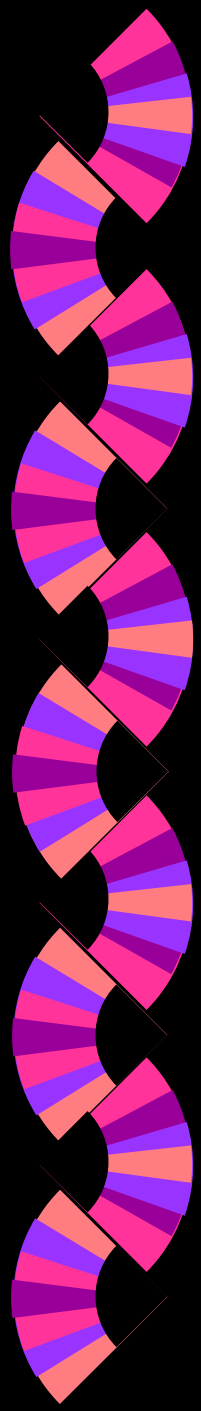
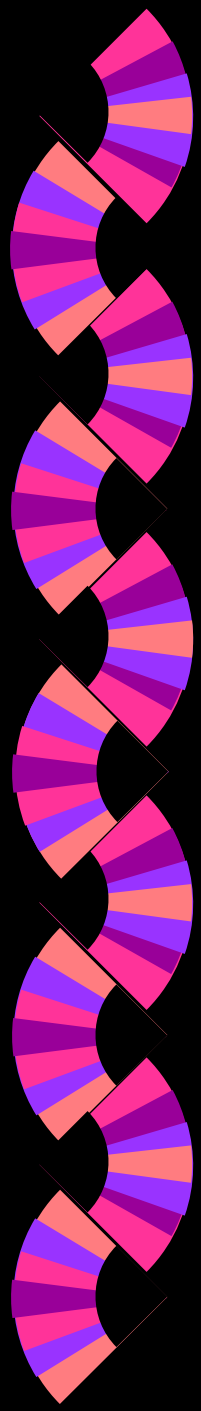


Visualização e Rendering

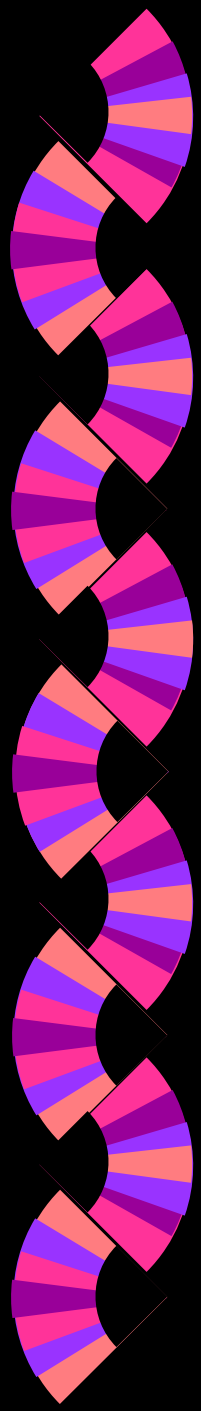
[Azevedo e Conci, 2003]

Caps 7.1.1, 7.2.3, 7.2.4, 7.2.4.1, 7.2.4.3 e 7.3.8

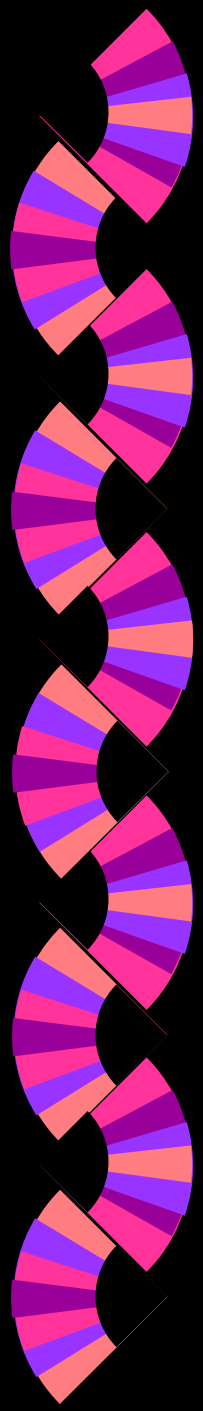


Visualização e Rendering

- Visualização (o que se vê ?)
  - Transformação da Câmera Sintética
  - Projeção e Perspectiva
  - Recorte e (RE)Poligonalização
- *Rendering* (como se pinta ?)
  - Colorização (Zbuffer, Pintor, RCast, SL,...)
  - Tonalização (Shading)
  - Realismo (RayTracing, Textura, Shadows, Efeitos)



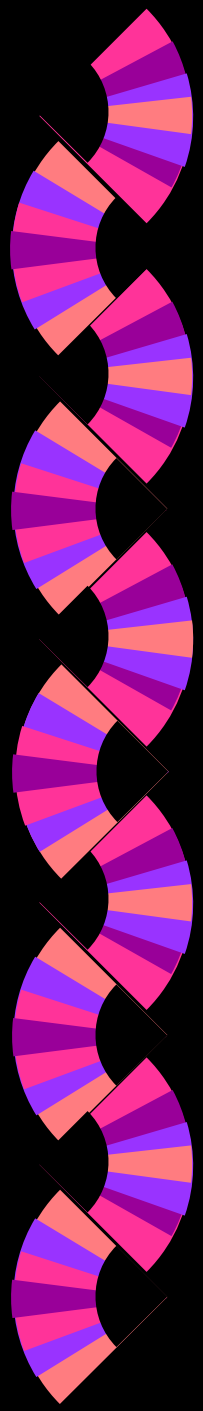
VIEW VOLUME CULLING



View Volume Clipping (VVC)

View Volume Clipping remove poliedros e poligonos que não estão no View Frustum

Às vezes chamado de Frustum clipping



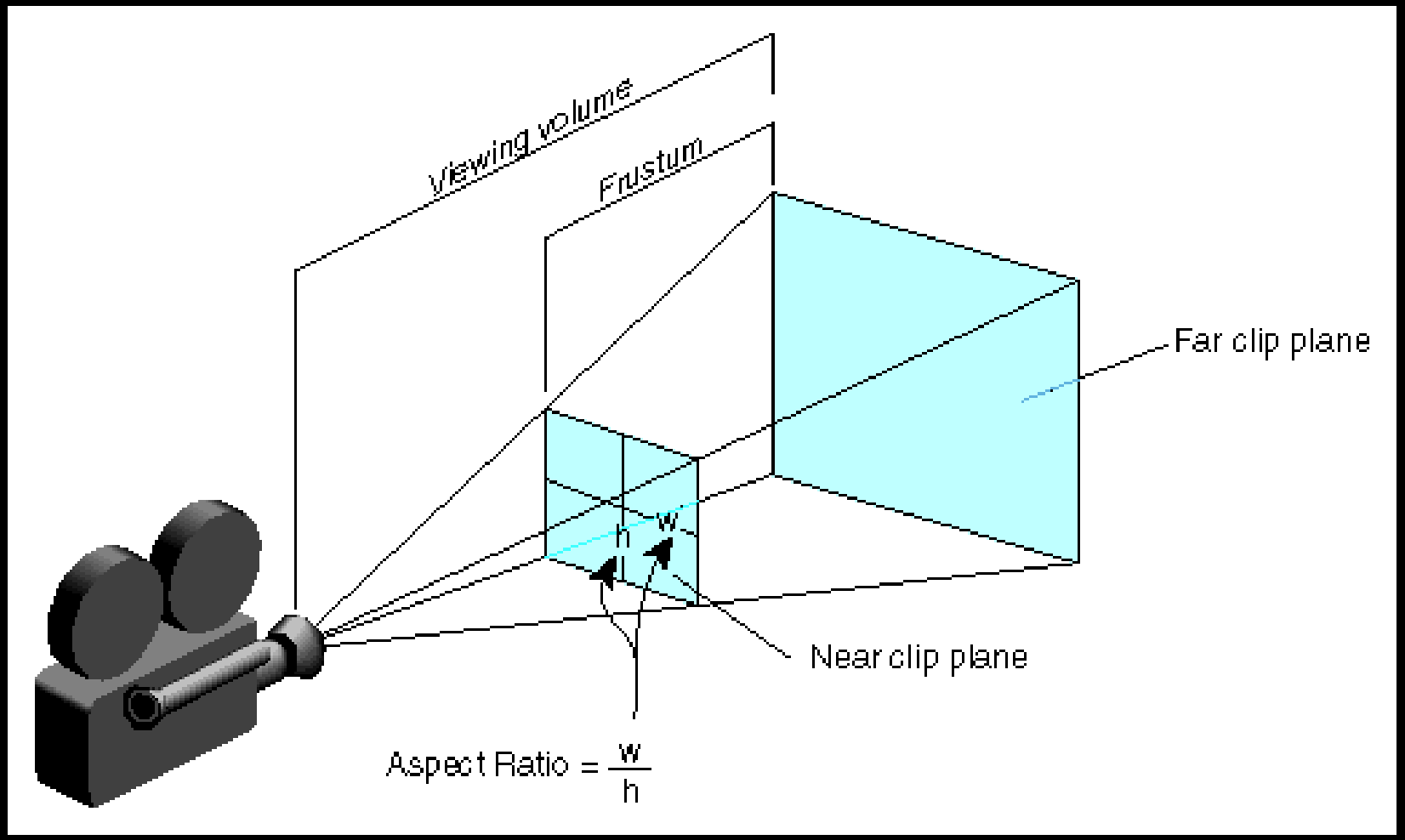
View Volume Clipping (VVC)

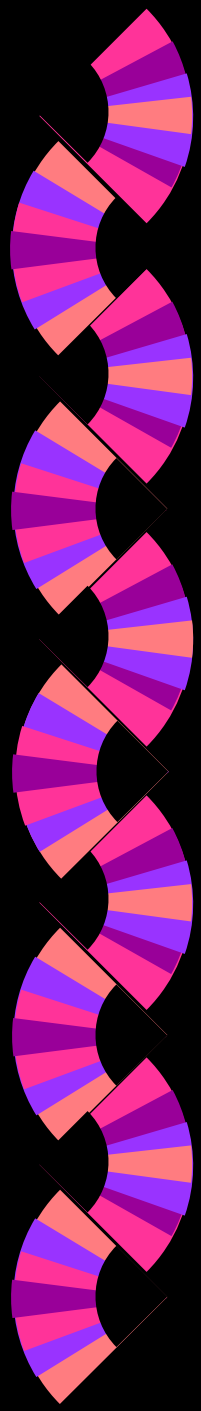
Frustum é definido pelos planos Near, Far e limites superior, inferior, esquerda e direita

Se ocorre uma projeção PARALELA, o procedimento é trivial tanto no espaço 2D quanto 3D

Se ocorre uma projeção PERSPECTIVA, o procedimento é um pouco mais demorado no espaço 3D

View Volume Clipping (VVC)

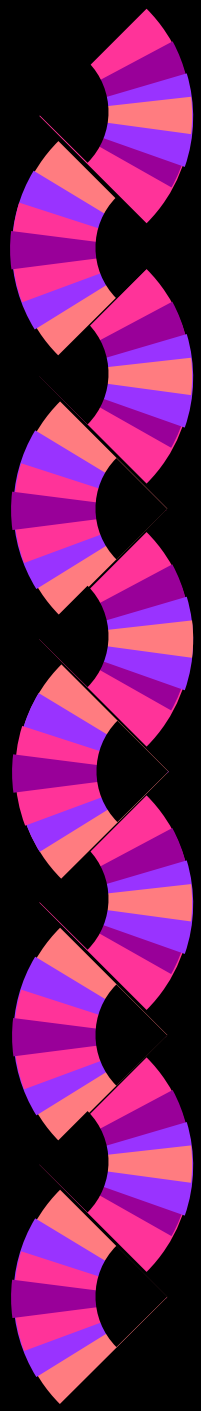




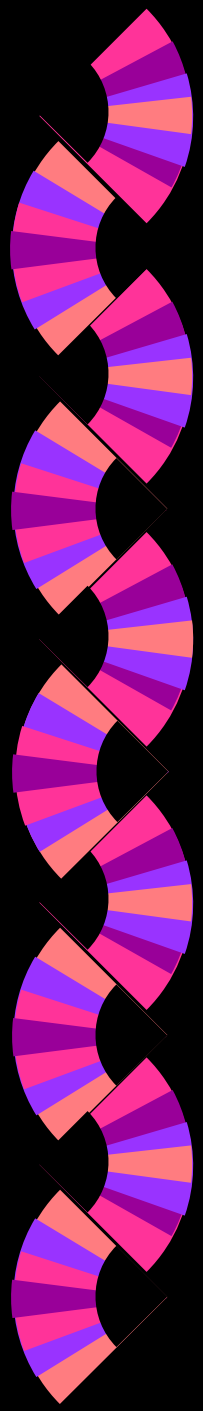
View Volume Clipping

VVC ocorre automaticamente no OpenGL and DirectX

É preciso ficar atento a isso pois pode-se obter **tela preta** se o Frustrum for mal definido



BACK FACE CULLING



Definições

Auxilia a criar resultados mais realistas

Elimina as faces poligonais que não estão voltadas para o observador.

Considera a posição relativa entre os objetos e o observador.

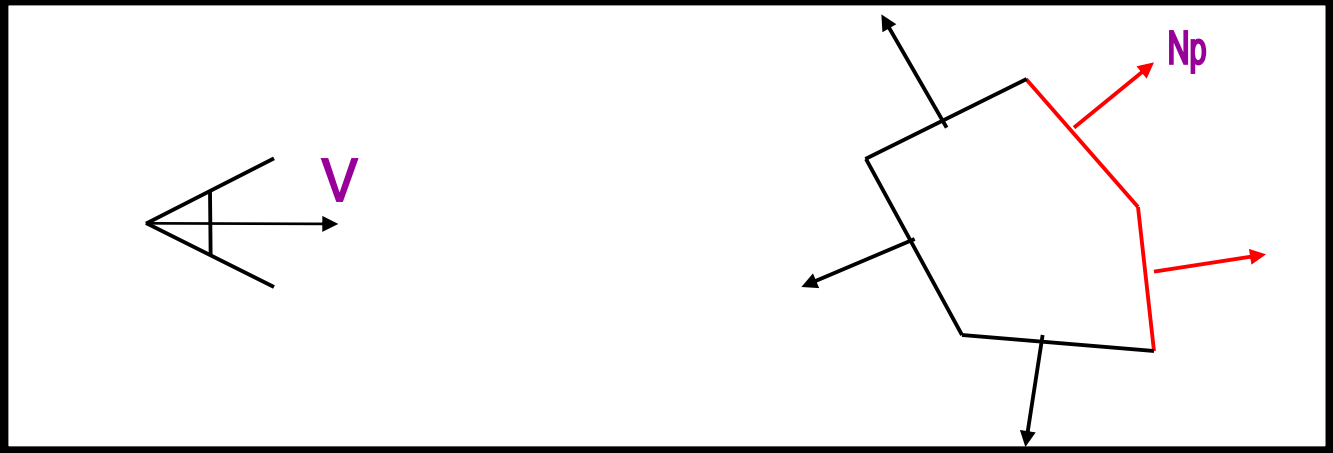
Back-face Culling

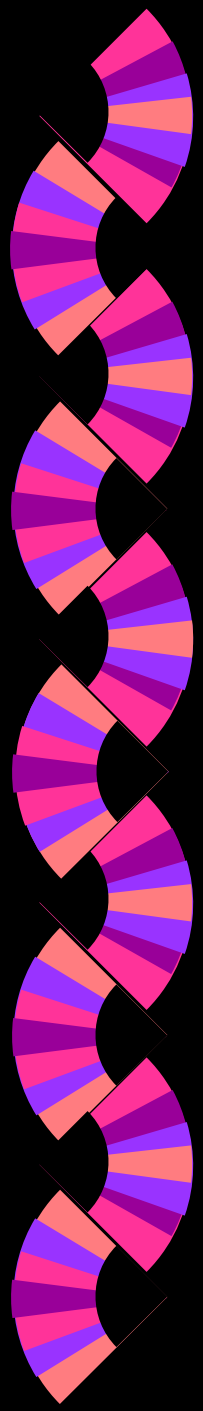
Normal da Face indica o lado de fora

Normalmente as normais assumem o polígono (normalmente triangulos) de serem definidos counter clock-wise (ccw)

Pelo menos para sistemas com WC da mão direita (OpenGL)

DirectX's é mão-esquerda e

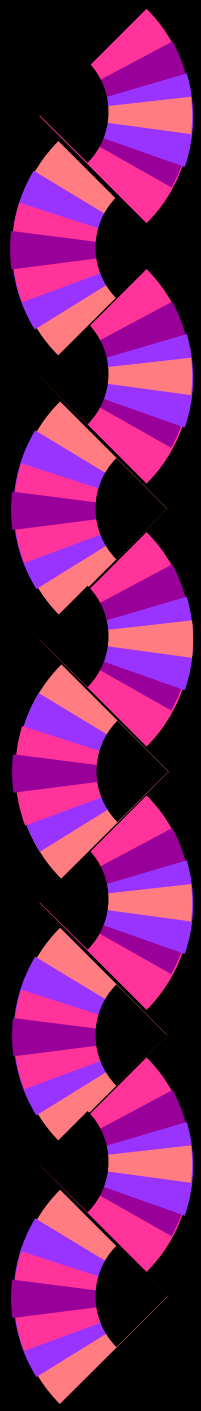




Back-face Culling

Esta técnica remove, em média, a metade dos polígonos numa cena típica

E isto ocorre nas primeiras etapas do pipeline.



Algoritmo

Os algoritmos de BFC devem realizar as seguintes tarefas:

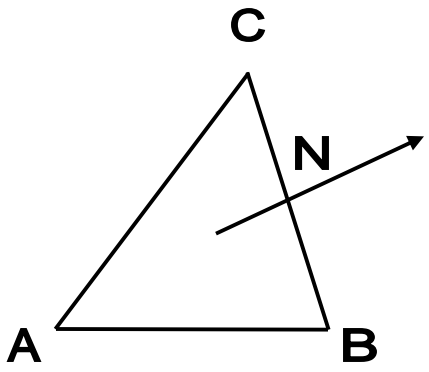
- Localizar no espaço 3D a posição do observador, através da qual definirá os parâmetros de visibilidade.
- Calcular o **vetor normal 3D de cada face** do objeto.
- Calcular o **vetor de visualização** para cada face do objeto.
- Realizar o **teste de visibilidade**. Isso é feito verificando a magnitude do ângulo formado pela normal a face em consideração vetor de visualização.

Normal a uma Superfície

Cada face (polígono/triângulo) tem uma única normal à sua superfície

É a forma mais fácil de determinar a orientação da face

A normal é um “vetor”, não tem posição



Calculando a Normal

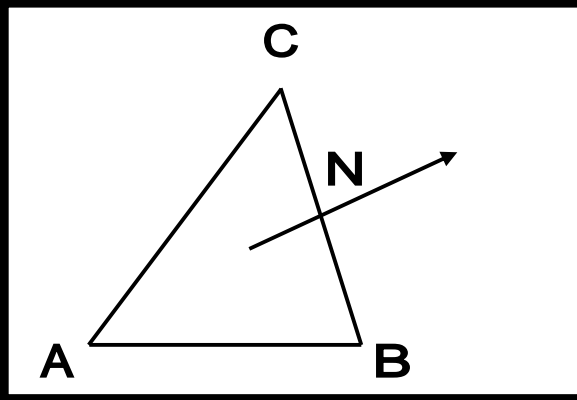
Seja V_1 o vetor de A a B

Seja V_2 o vetor de A a C

$N = V_1 \times V_2$ (mão direita)

Produto vetorial

N é normalizada



A ordem dos vértices é importante

- Triangulo ABC tem normal N
- Triangule ACB tem normal invertida

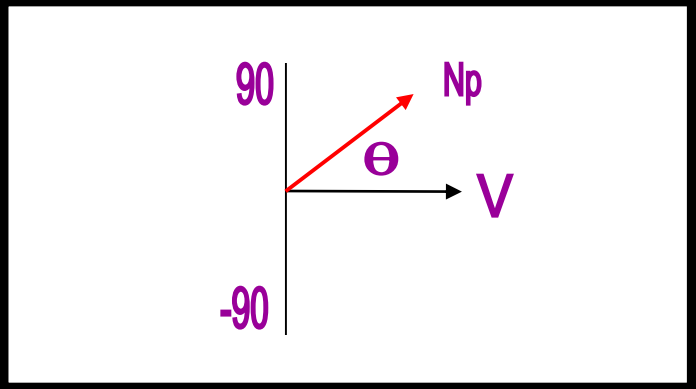
Back-face Culling

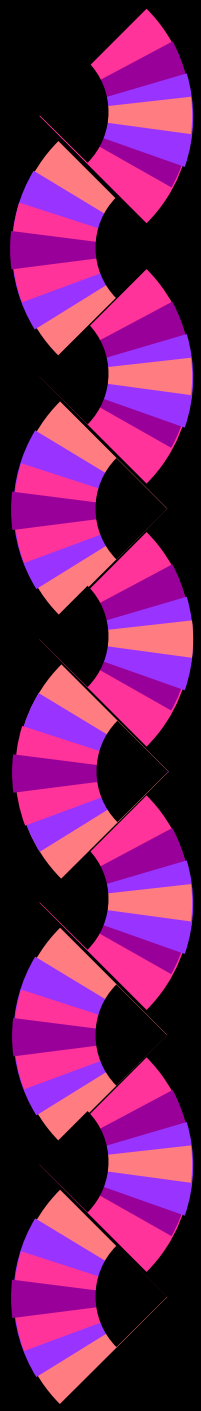
Lebrando que $V_1 \cdot V_2 = |V_1| \ |V_2| \ \cos(\theta)$

Se os vetores estão normalizados então $V_1 \cdot V_2 = \cos(\theta)$

Lembre que $\cos(\theta)$ é positivo se $\theta \in [-90..+90]$

Portanto, se o produto interno (escalar) entre o vetor de visualização (V) e o da Normal do Polígono (N_p) é positivo, pode-se "cull it" (remover)



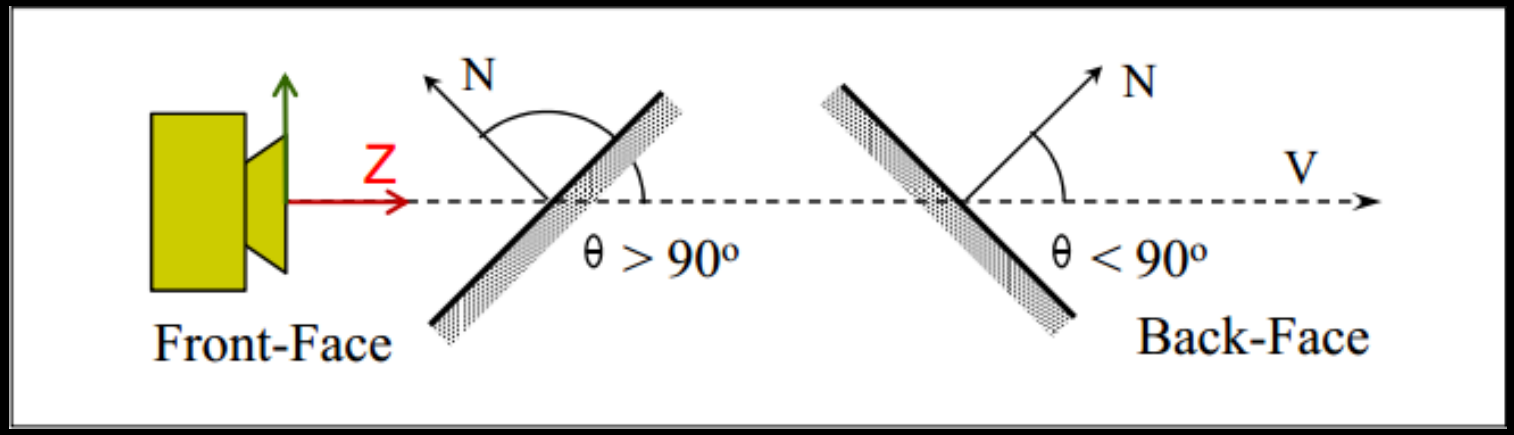


Back-face Culling

O produto interno é rápido de calcular …

… ainda, pode ser mais otimizado pois o que se precisa, na verdade, é apenas o sinal

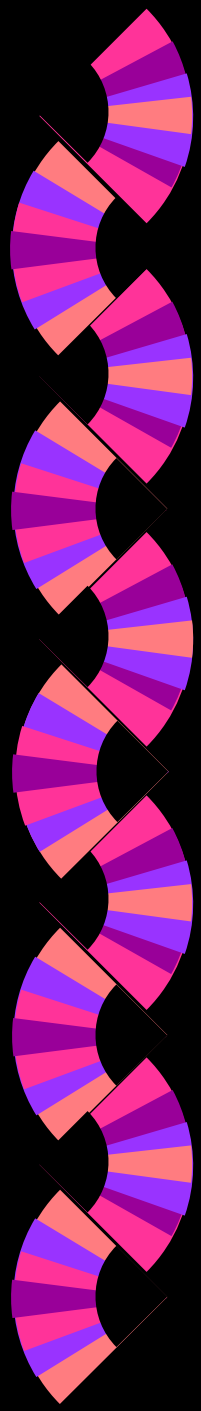
Algoritmo



Se o valor absoluto do ângulo Θ estiver entre 90° e 180° , a superfície está visível.

A superfície está invisível se Θ estiver entre 0° e 90° .

Softwares que utilizam Back-Face Culling

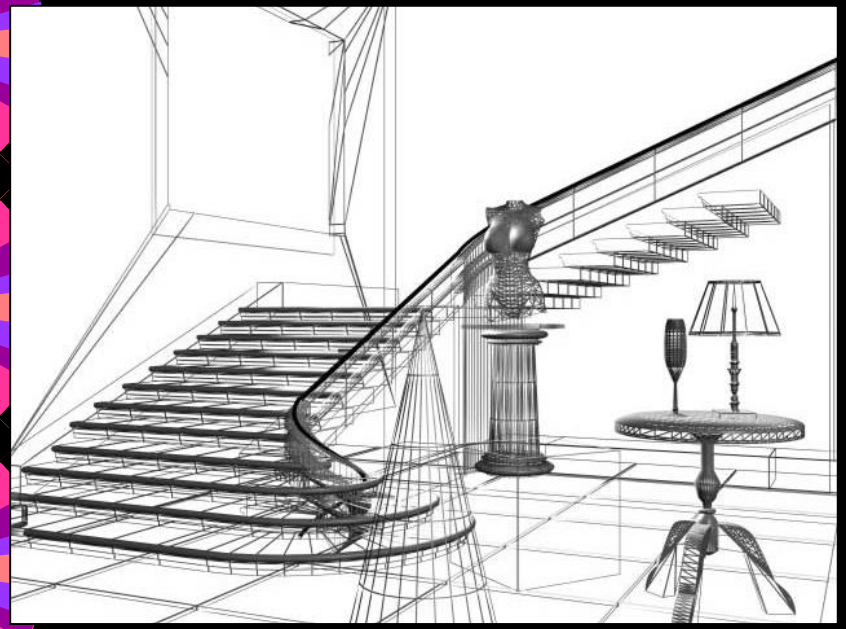


Visualização e Rendering

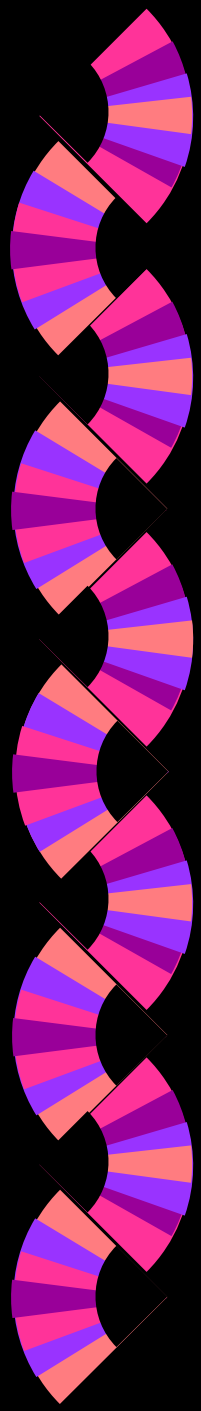
- Visualização (o que se vê ?)
  - Transformação da Câmera Sintética
  - Projeções (Perspectivas)
  - Recorte e (RE)Poligonalização
  - HLHS (BFC, VVC)
- *Rendering* (como se pinta ?)
  - Colorização (Zbuffer, Pintor, RCast, SL,...)
  - Tonalização (Shading)
  - Realismo (RayTracing, Textura, Shadows, Efeitos)

Algoritmos de Visibilidade
Escondimento de Superfícies

Visualização em wire-frame

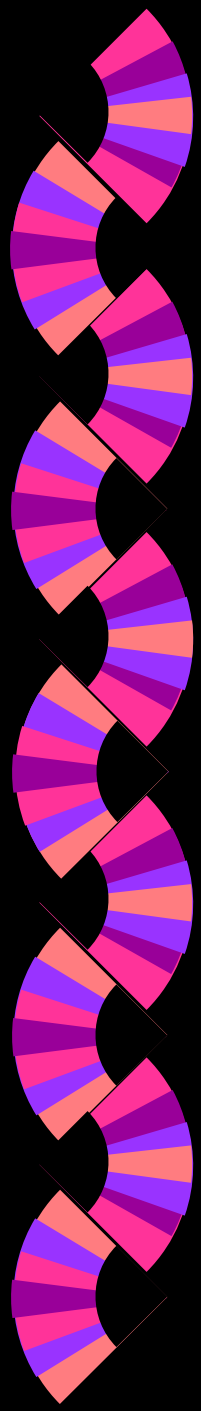


Visualização com HLHS



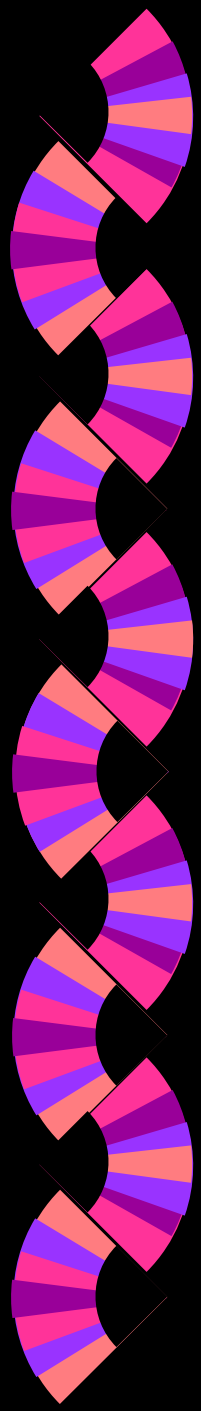
Rendering

- É a busca do **realismo fotográfico**
- Vai além da simples exposição dos modelos gráficos 3D na tela
- É feito através do uso de algoritmos específicos
- Não existe uma tradução específica
- Alguns textos consideram os algoritmos de visualização como parte do rendering



Rendering:

- Principais Algoritmos :
  - Técnica do Pintor (*Detph-Priority*)
  - *Z-buffer* (*Depth-buffer)*
  - *Scanline*
  - *Ray-casting (Raytracing)*
  - Radiosidade



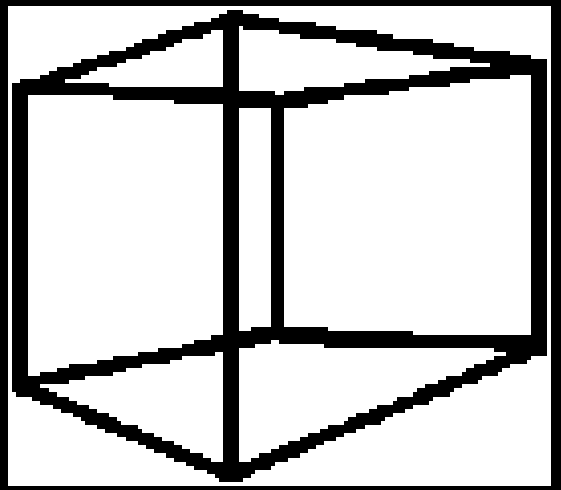
PINTOR OU DEPTH PRIORITY

ALGORITMO DO PINTOR

Primeiro pintam-se as montanhas distantes.
Depois, pinta-se o campo.
Finalmente, pintam-se as árvores, as quais são os objetos mais próximos.

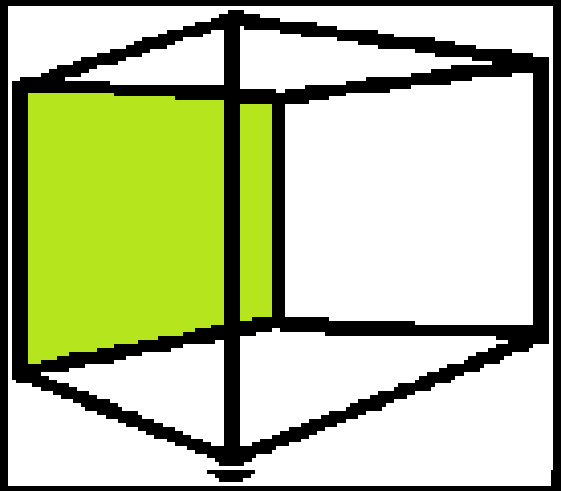
Funcionamento

- Calcular a que distância do observador está cada face na cena.
- Ordenar todos os polígonos pelo valor da sua distância.
- Resolver ambiguidades (distâncias iguais).
- Desenhar todas as faces em ordem decrescente.



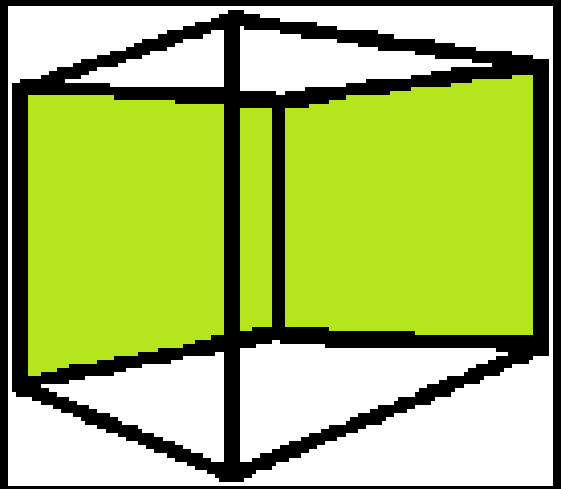
Funcionamento

- Calcular a que distância do observador está cada face na cena.
- Ordenar todos os polígonos pelo valor da sua distância.
- Resolver ambiguidades (distâncias iguais).
- Desenhar todas as faces em ordem decrescente.



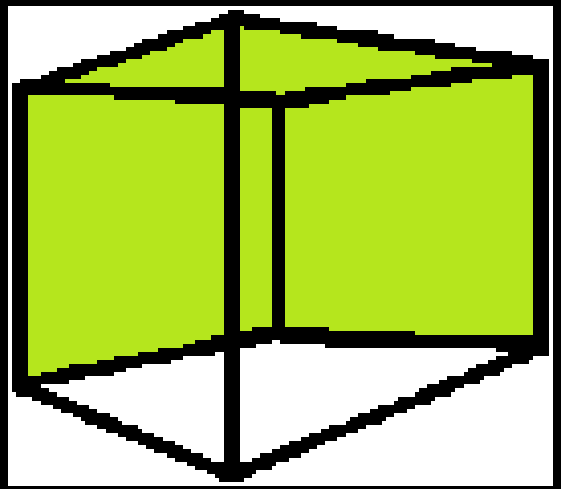
Funcionamento

- Calcular a que distância do observador está cada face na cena.
- Ordenar todos os polígonos pelo valor da sua distância.
- Resolver ambiguidades (distâncias iguais).
- Desenhar todas as faces em ordem decrescente.



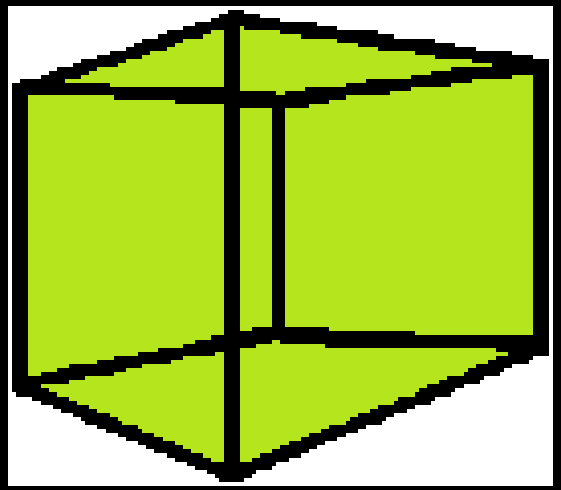
Funcionamento

- Calcular a que distância do observador está cada face na cena.
- Ordenar todos os polígonos pelo valor da sua distância.
- Resolver ambiguidades (distâncias iguais).
- Desenhar todas as faces em ordem decrescente.



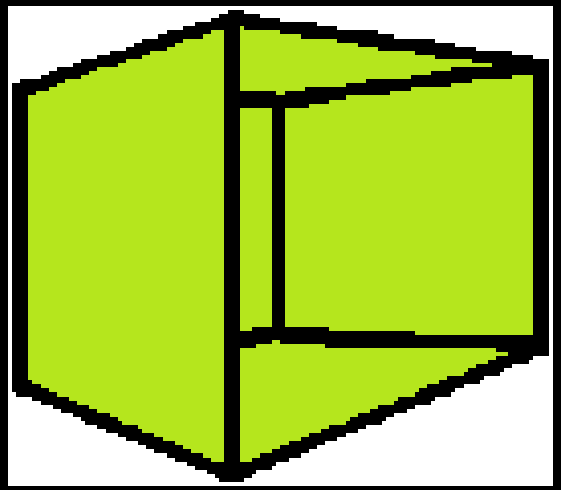
Funcionamento

- Calcular a que distância do observador está cada face na cena.
- Ordenar todos os polígonos pelo valor da sua distância.
- Resolver ambiguidades (distâncias iguais).
- Desenhar todas as faces em ordem decrescente.



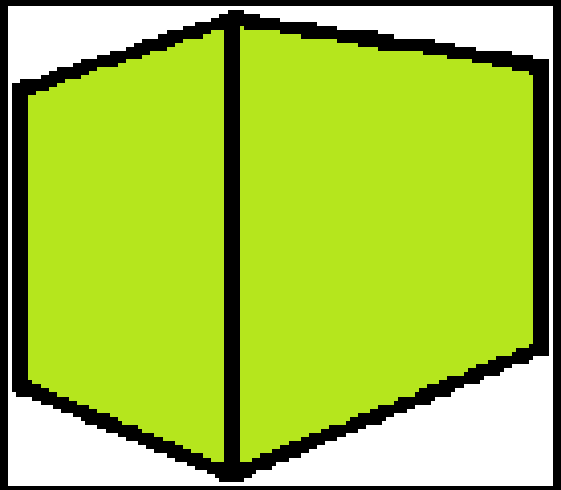
Funcionamento

- Calcular a que distância do observador está cada face na cena.
- Ordenar todos os polígonos pelo valor da sua distância.
- Resolver ambiguidades (distâncias iguais).
- Desenhar todas as faces em ordem decrescente.

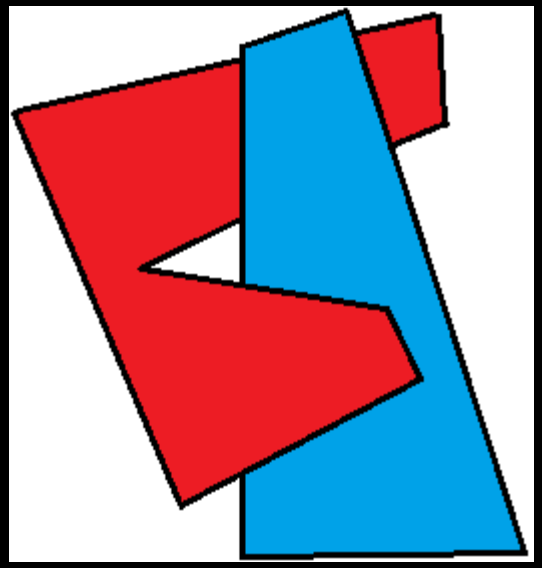


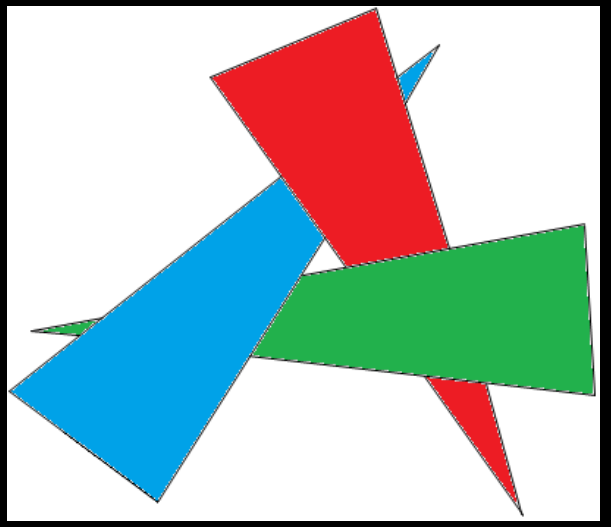
Funcionamento

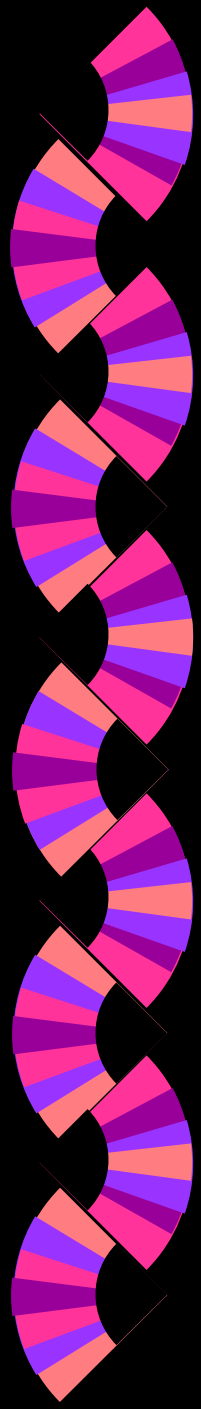
- Calcular a que distância do observador está cada face na cena.
- Ordenar todos os polígonos pelo valor da sua distância.
- Resolver ambiguidades (distâncias iguais).
- Desenhar todas as faces em ordem decrescente.



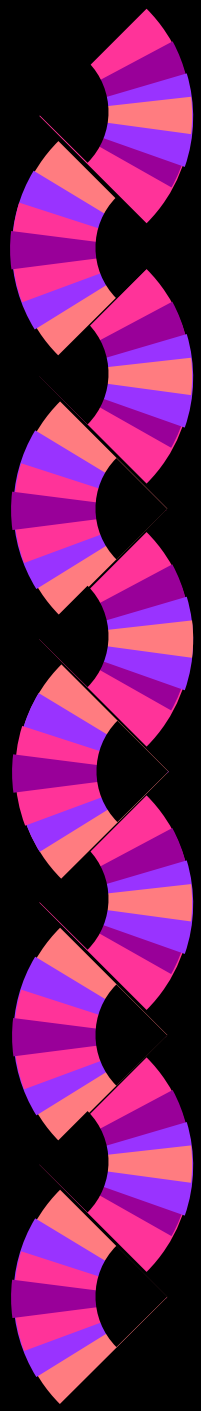
Resolver ambigüidades





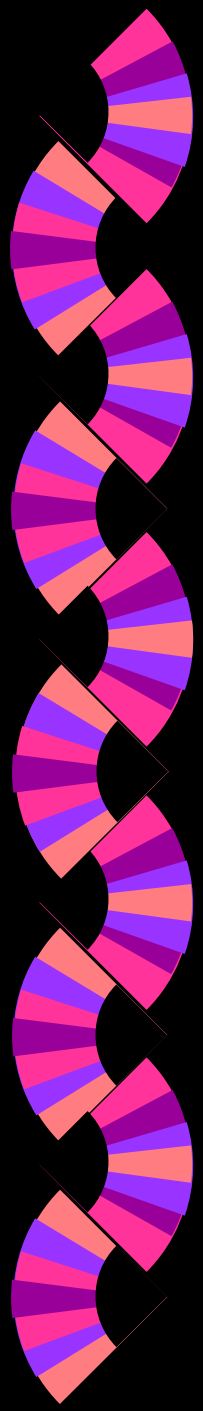


ZBUFFER OU DEPTH BUFFER



Rendering:

- *Z-buffer ou Depth-buffer*
  - Face a face, para a tela inteira
  - Não faz nenhuma ordenação
  - Compara o Z do objeto com o buffer, em cada ponto
  - Simples, rápido, preciso, mas requer muita memória
  - Fácil de implementar em HW
- Algoritmo do Pintor e Z-buffer só permitem "pseudo-transparência" e tem problemas com *anti-aliasing (discretização)*



Z-BUFFER

Registra a profundidade para todos os pixels do buffer e sempre que surgir um novo na mesma posição, se for mais a frente sobrescreve, caso contrário descarta.

Resumindo:

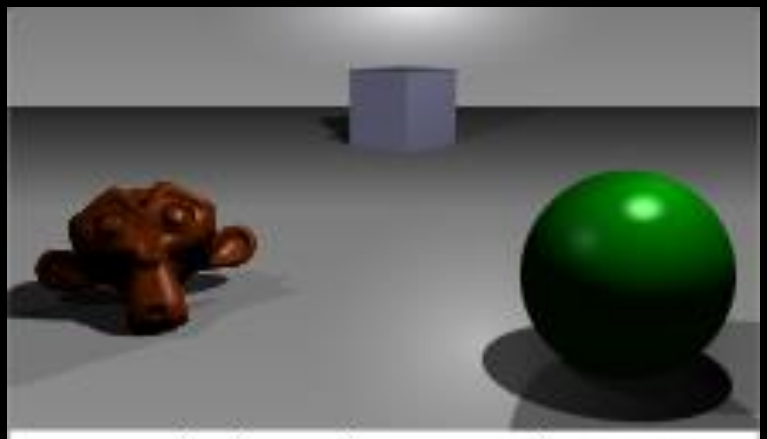
Matriz 2D que salva a

profundidade de cada pixel.

Z-buffer

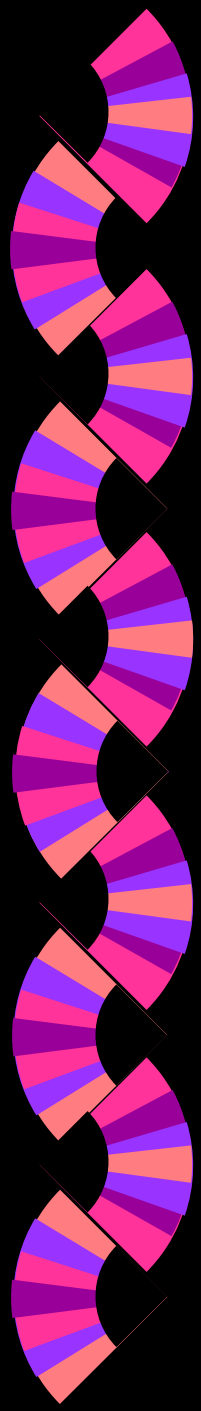
A simple three dimensional scene

Z-buffer representation



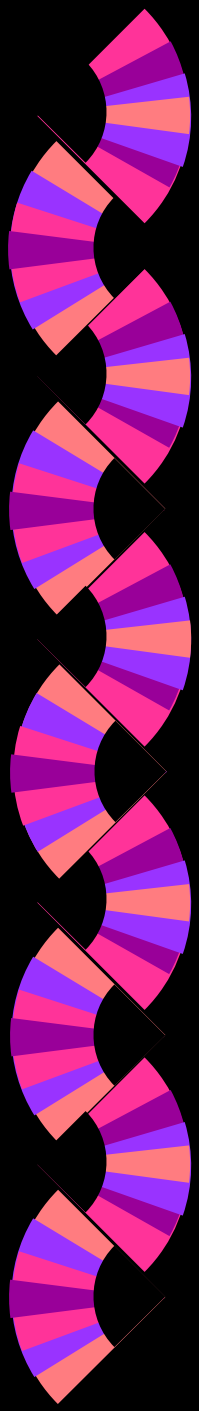
A simple three-dimensional scene

Z-buffer representation

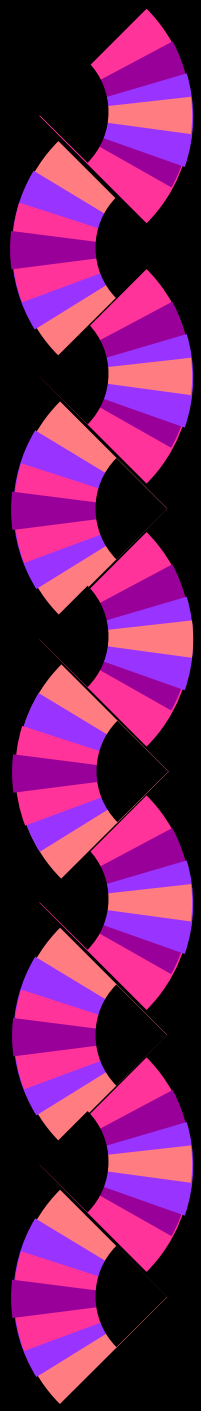


ZBUFFER - PSEUDO-CÓDIGO

```
Dados:
Lista of polígonos {P1, P2, ..., Pn}
Matriz z-buffer[x,y] inicializado com -∞
Matriz Intensidade[x,y]
Início
para cada polígono P na lista de polígonos  faça {
      para cada pixel (x,y) que intercepta P faça {
            calcule profundidade-z de P na posição (x,y)
            se prof-z < z-buffer[x,y] então {
                  Intensidade[x,y] = intensidade de P em (x,y)
                  z-buffer[x,y] = prof-z
            }
      }
}
Desenhe Intensidade
fim
```



SCANLINE



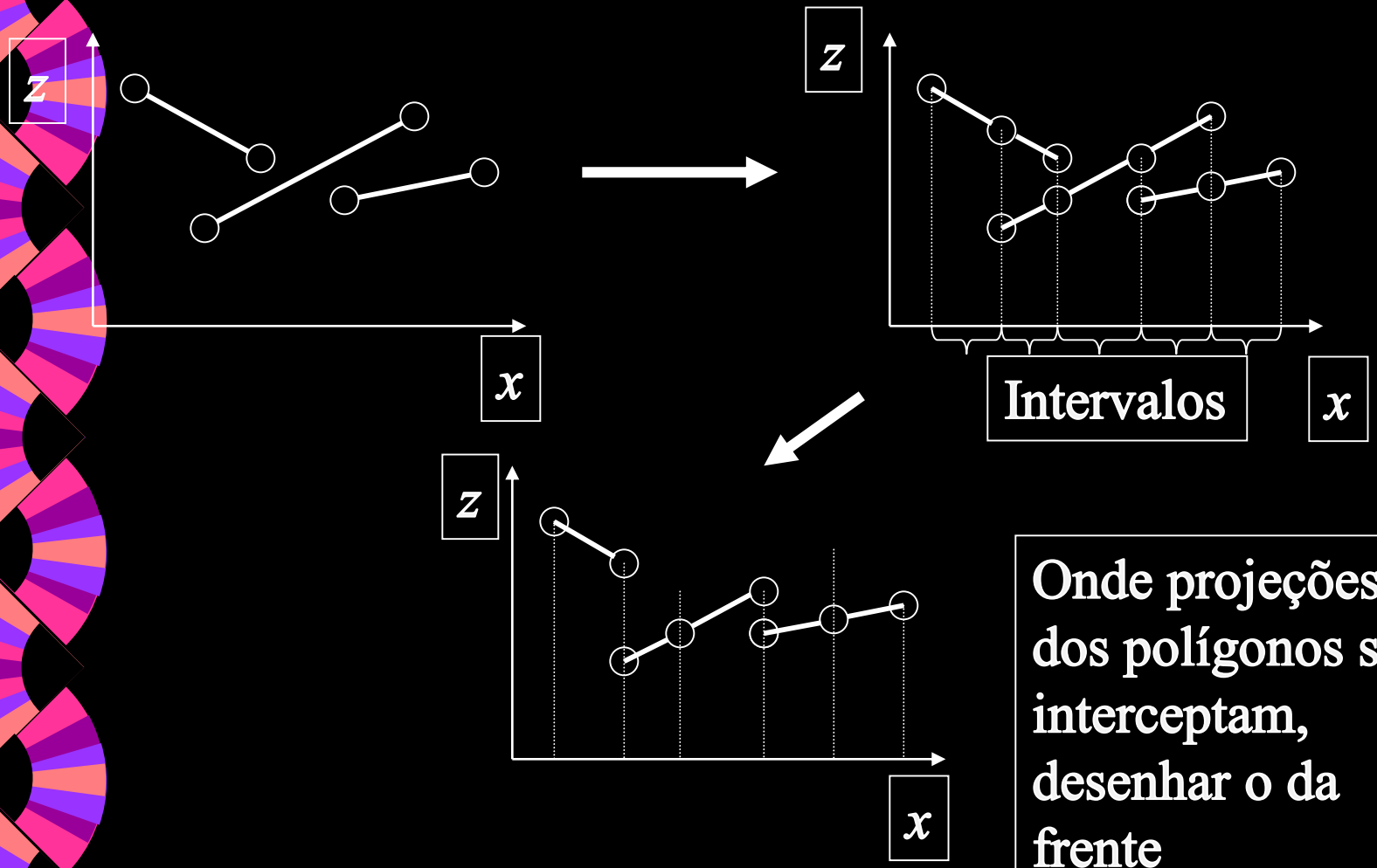
Rendering:

- Scanline
  - Linha de Exploração/Rastreamento, gera a imagem linha por linha
  - Mantém uma lista ordenada dos objetos e utiliza-se muito da coerência existente de uma linha para a seguinte
  - É como se fosse um Z-buffer otimizado e numa única linha
  - De uma linha para outra, de um pixel para outro as características mudam incrementalmente.

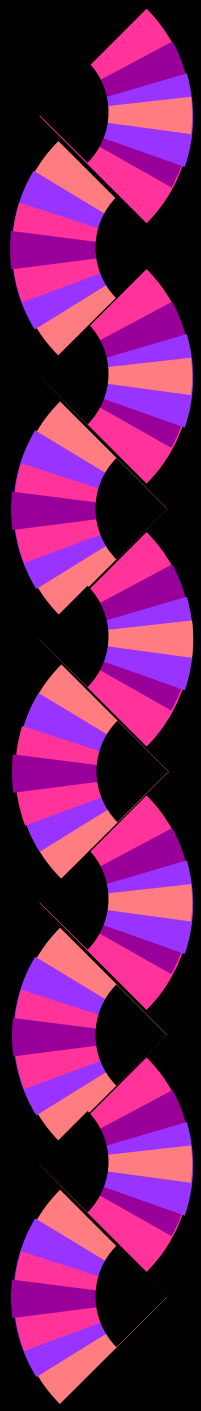
SCAN LINE

- Algoritmo de rasterização de polígonos
- Consiste em transformar uma região plana delimitada por uma sequência fechada de segmentos em um conjunto de pixels conexos
- Em uma linha de varredura os pixels são divididos em “spans”, separadas por pixels que representam as arestas da borda

SCAN LINE



Onde projeções dos polígonos se interceptam, desenhar o da frente

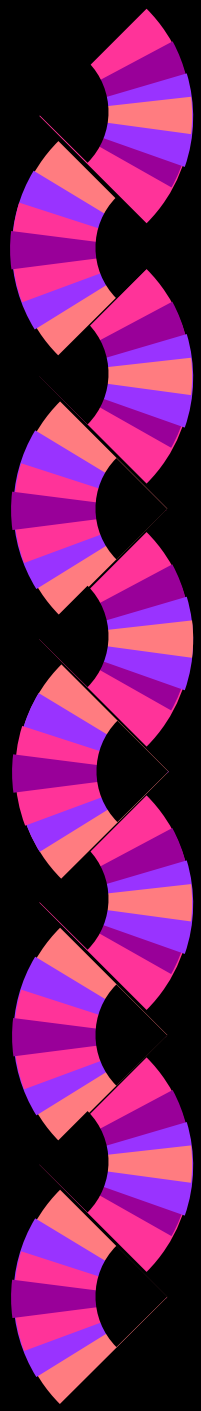


SCAN LINE

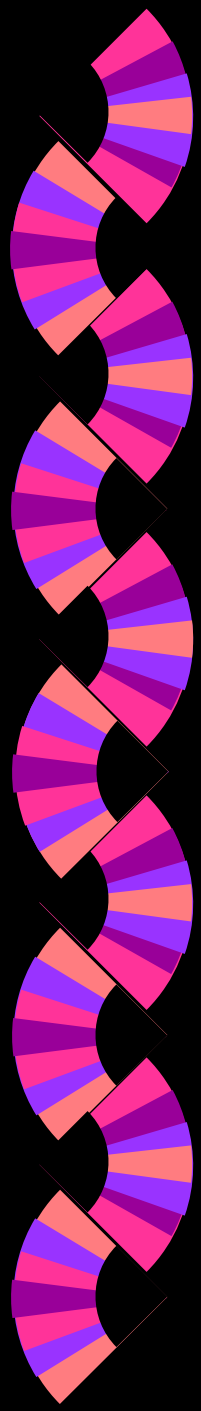
Ordena-se todas as arestas de todos os polígonos por *ymin*

Para cada plano de varredura *y*

- Para cada polígono
  - Determinar intervalos x_i de interseção com plano de varredura
- Ordenar intervalos de interseção por *zmin*
- Para cada linha de varredura *z*
  - Inserir arestas na linha de varredura respeitando inclinação *z*/*x*
- Renderizar resultado da linha de varredura

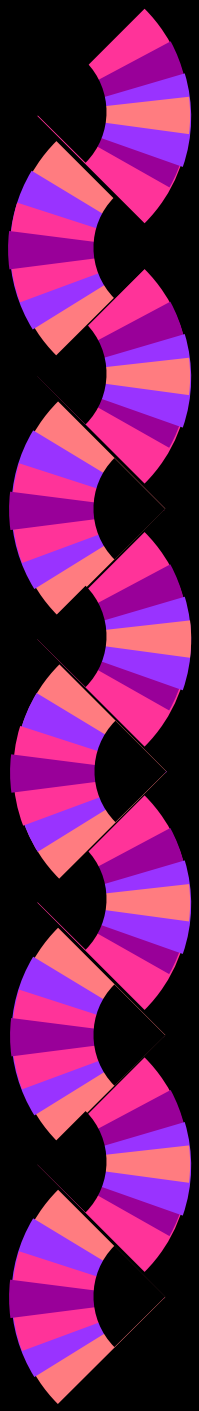


RAY CASTING/TRACING



Rendering:

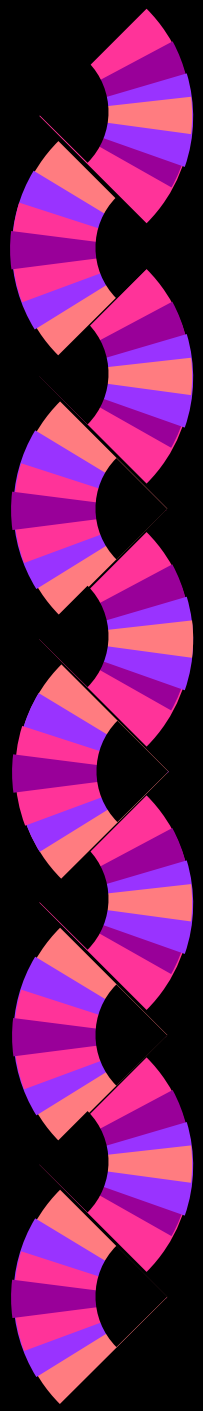
- Ray Casting/Tracing
  - Calcula pixel a pixel lançando raios a partir do observador
  - Algoritmo que melhor calcula transparência, espelhamento e sombras
  - Melhor imagem, muito lento, maior tempo !!!



Características especiais

Ray tracing não depende da transformação dos objetos de toda a cena em polígonos

Realiza implicitamente a remoção de superfícies ocultas, pois só o ponto de intersecção mais próximo do raio é visível a partir do centro de projeção

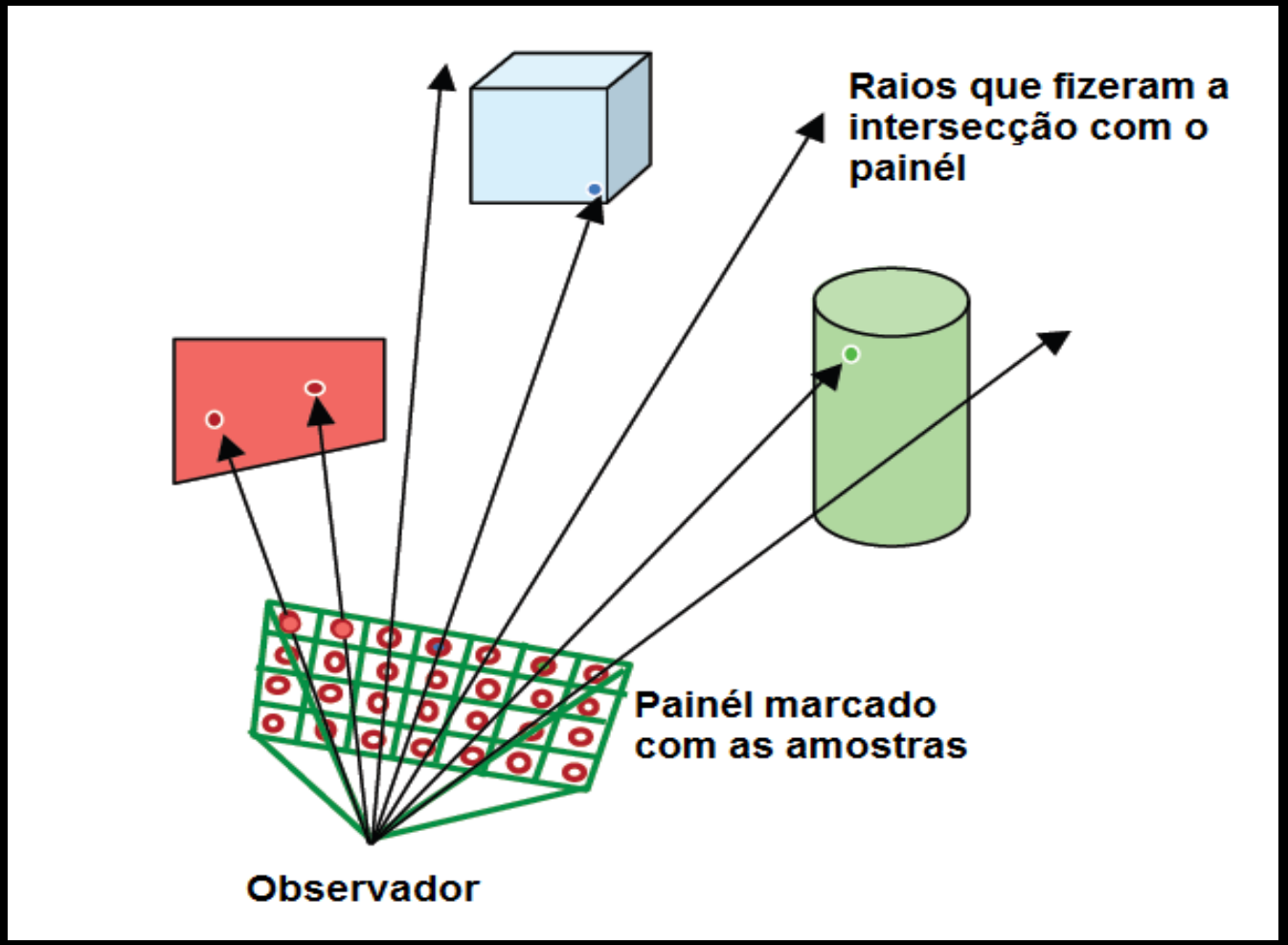


Funcionamento

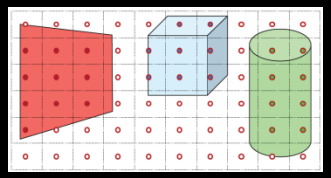
A ideia básica deste algoritmo consiste em traçar, para cada pixel na janela de visualização, um raio a partir do centro de projeção até o centro do pixel da cena.

Assim, a cor do pixel será definida como a cor do ponto de intersecção mais próximo encontrado

Ray Casting


Raios que fizeram a intersecção com o painél
Painél marcado com as amostras
Observador

Ray Casting
(pontos vermelhos confundem!!!)

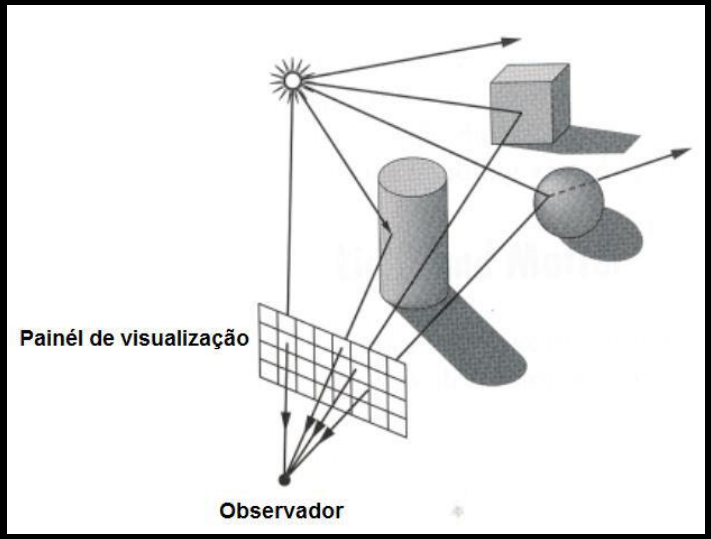


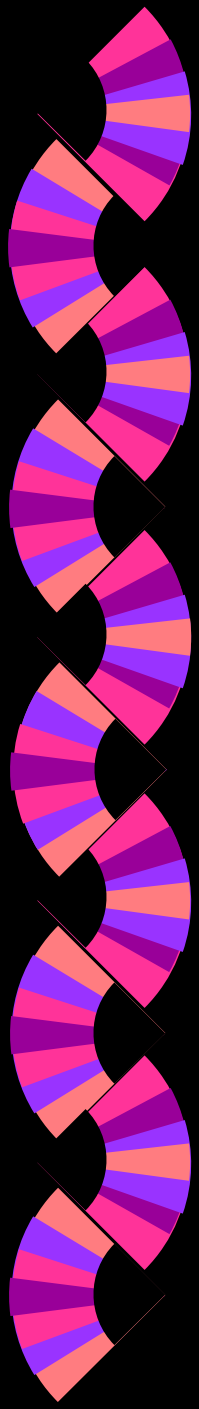
Pseudo Código

```
Seleccionar o centro de projecção e a janela de recorte sobre o plano de projecção
Para cada linha horizontal (de varrimento) da imagem
  {
    Para cada pixel da linha de varrimento
    { Determinar o raio que une o centro de projecção com o pixel
      Para cada objecto da cena
        { Se o raio intersecta o objecto e o ponto de intersecção encontra-se
          mais próximo do centro de projecção do que o ponto de
          intersecção até agora encontrado
            Registar o ponto de intersecção e o objecto intersectado
        }
      Atribuir ao pixel a cor do objecto intersectado no ponto de intersecção
      registado
    }
  }
```

Ray Casting/Tracing

Ray casting não é sinônimo de Ray tracing mas pode ser entendido como sendo uma versão abreviada e significativamente mais rápida do que o algoritmo de Ray tracing





Vantagens e Desvantagens

Pode-se gerar imagens com alta qualidade;

Ainda não se consegue gerar animações em tempo real;

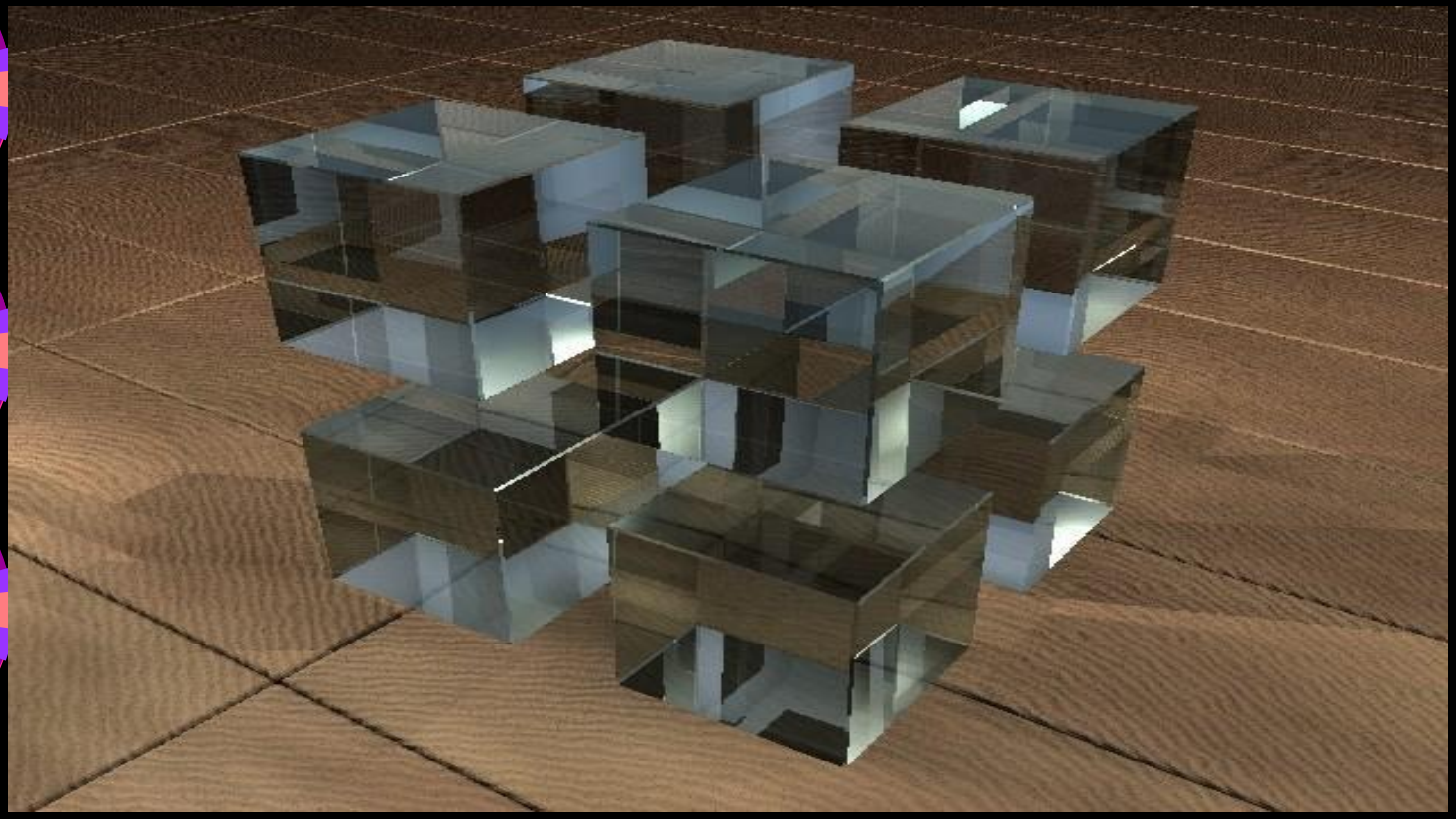
Algoritmo altamente paralelizável.

Ray Tracing

O Ray tracing oferece uma imagem final com uma riqueza de detalhes bem maior se comparado com cenas rasterizadas

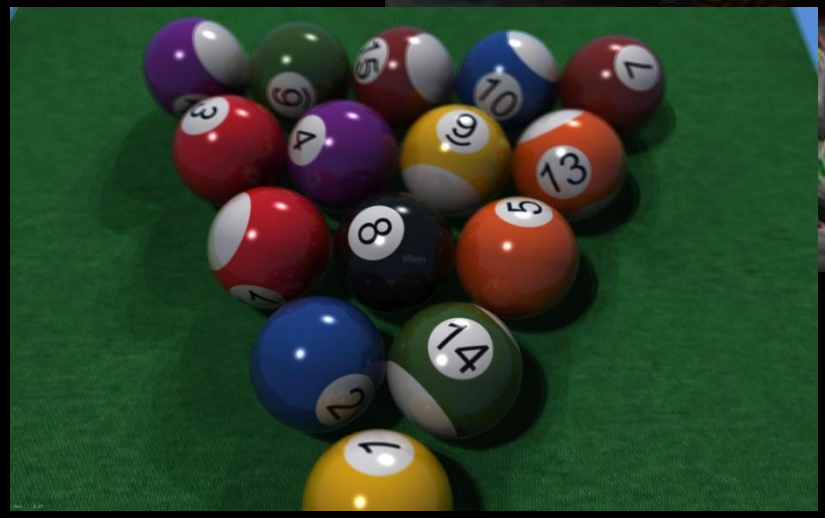
Ray Casting/Tracing

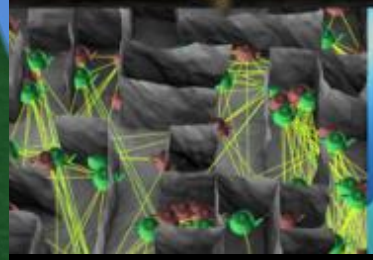
Blender

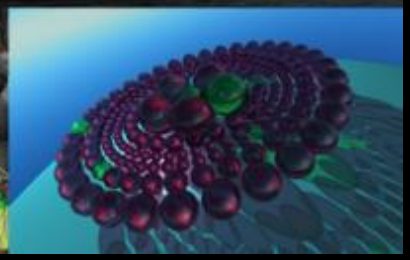


Ray Casting/Tracing

Nvidia Optix







Softwares que utilizam Ray casting/tracing