# Determinação de Superfícies Visíveis

Uéliton Freitas

Universidade Católica Dom Bosco - UCDB

*freitas.ueliton@gmail.com*

10 de novembro de 2014

# Sumário

# Introdução

## Rendering de Polígonos

- Por eficiência, queremos renderizar apenas as faces poligonais que são visíveis para a câmera.

- Existem diversos algoritmos para **detecção de superfícies visíveis** (ou eliminação de superfícies ocultas) que variam conforme:
  - Complexidade da cena.
  - Tipo de objeto desenhado.
  - Equipamento disponível.
  - etc.

# Introdução

## Classificação dos Algoritmos

- Os algoritmos podem ser classificados em dois grandes grupos:
  - Métodos de **espaço do objeto**.
  - Métodos de **espaço da imagem**.

## Espaço do Objeto

- Compara objetos entre si, ou partes de objetos, para determinar a visibilidade.

## Espaço da Imagem

- Compara pixel por pixel no plano de projeção para determinar a visibilidade.

# Introdução

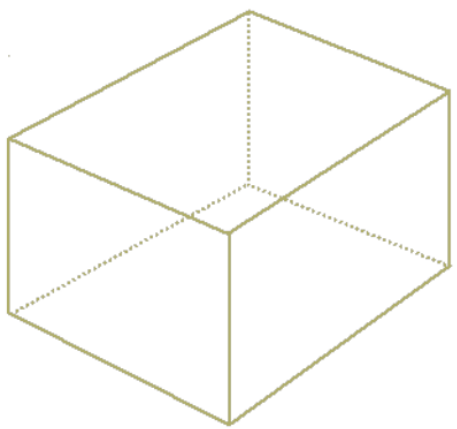
## Classificação dos Algoritmos

- Discutiremos dois algoritmos de visibilidade:
  - Back-face culling.
  - Z-buffer.

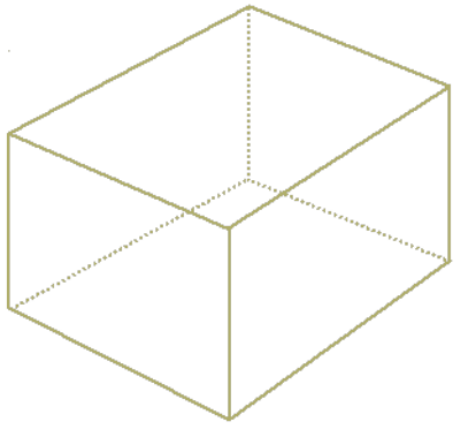
# Back-Face Culling

## Back-Face Culling

- Se as faces pertencem a um objeto sólido (um poliedro, por exemplo), não é necessário renderizar as faces de trás (não visíveis).



# Back-Face Culling

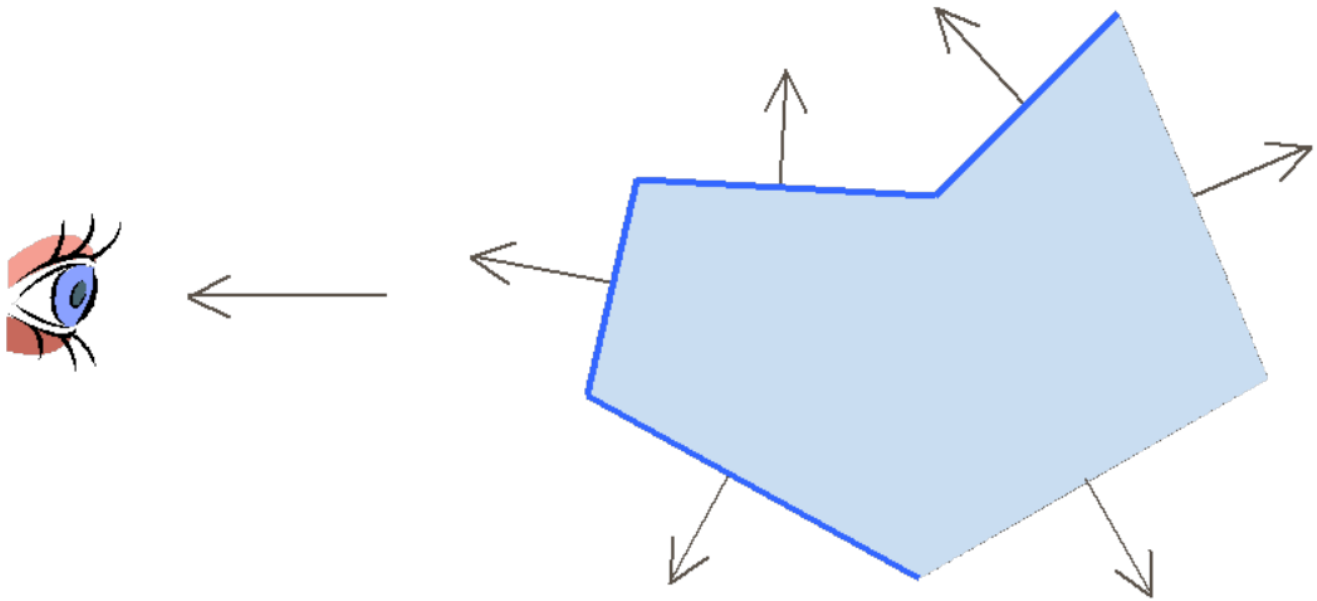


## Back-Face Culling

- Apenas três faces precisam ser traçadas.

- As faces “de trás” podem ser removidas do pipeline.

# Back-Face Culling



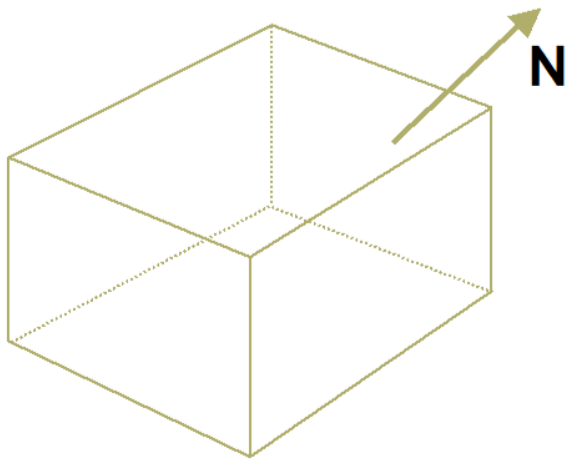
## Back-Face Culling

- Assume-se que a cena é composta por poliedros fechados.

# Back-Face Culling

## Back-Face Culling

- Como descobrir quais são as "faces de trás"?

# Back-Face Culling

## Back-Face Culling

- Eficiente quando os testes são feitos no sistema de coordenadas de visão.
  - Vetor de direção de observação paralelo ao eixo $z_v$
  - Assim $V = (0, 0, V_z)$, e $V \cdot N = V_z \cdot V_n$

- Portanto, para fazer o teste $V \cdot N > 0$ basta verificar o sinal do componente $z$ do vetor normal a face.

# Back-Face Culling

## Back-Face Culling

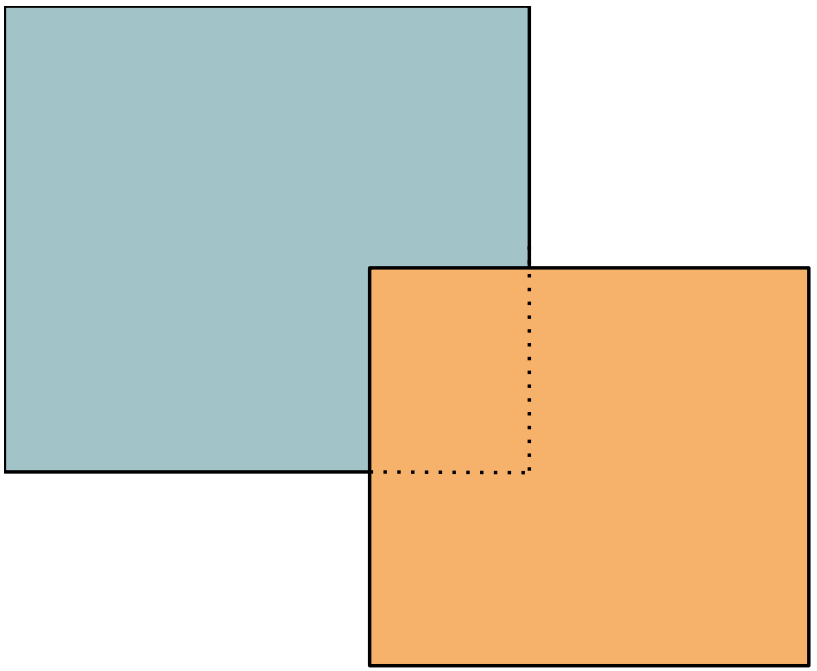
- Muito importante para *rendering* mais eficiente (simplifica muito a cena) em geral é o primeiro passo do processo.

- Assim, restam apenas os polígonos/faces potencialmente visíveis para a câmera.

# Z-Buffer

## Z-Buffer

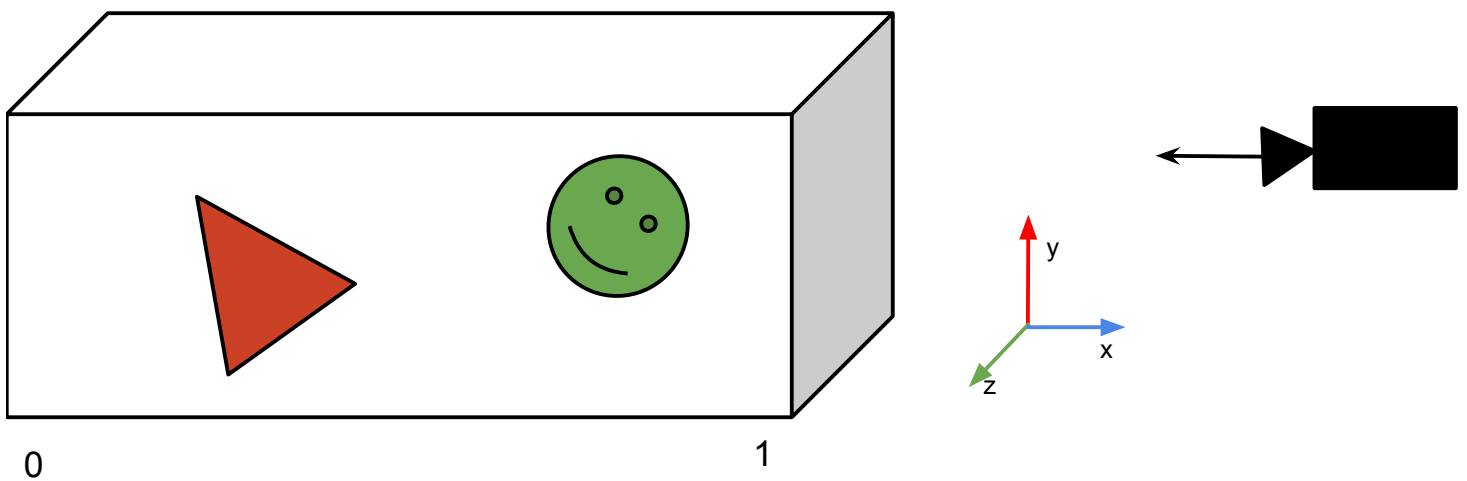
- Algumas faces ficam ocultas atrás das outras: só queremos renderizar (desenhar) as faces (ou partes delas) totalmente visíveis.



# Z-Buffer

## Z-Buffer

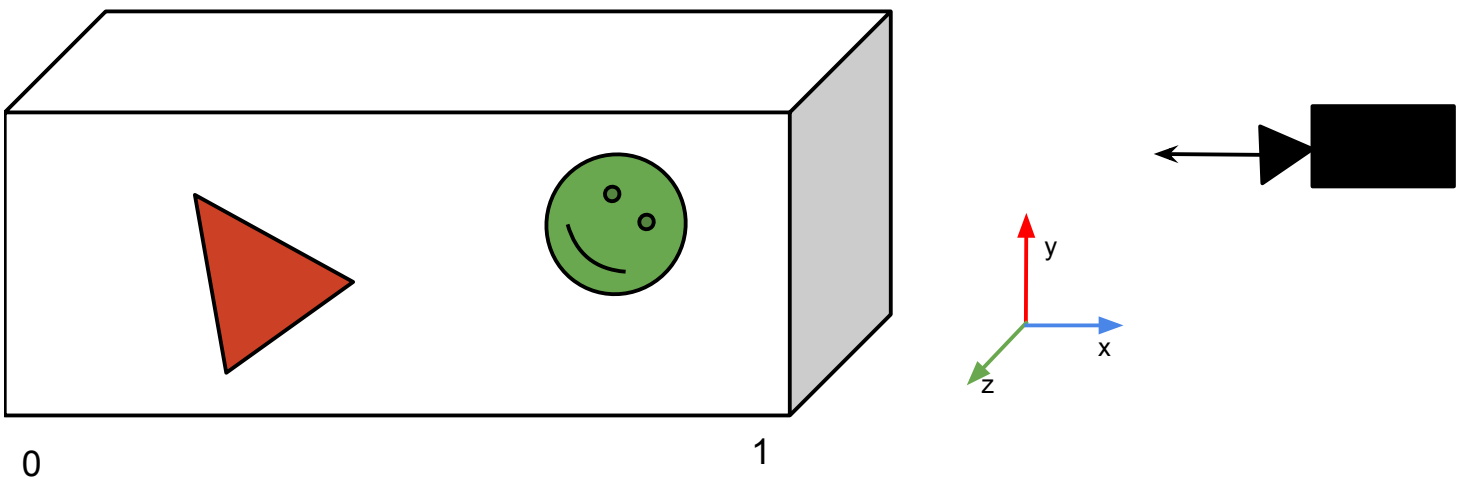
- Considere que as faces passaram pela transformação de projeção, e tiveram suas coordenadas $z$ armazenadas – suponha valores de $z$ normalizados no intervalo 0 e 1 (0 plano *near* e 1 plano *far*)



# Z-Buffer

## Z-Buffer

- Para cada pixel $(x, y)$, queremos traçar a face mais próxima da câmera, i.e, com menor valor $z$.



# Z-Buffer

## Z-Buffer

- O algoritmo usa dois buffers:
  - **Frame Buffer**: Armazena os valores RGB que definem a cor de cada pixel, tipicamente 24 bits, mais 8 bits de transparência (alfa).
  - **Z-Buffer**: para manter informações de profundidade associada a cada pixel, tipicamente 16, 24 ou 32 bits.

## Inicialização

- Todas as posições do Z-Buffer são inicializadas são inicializadas com a maior profundidade.
  depthBuffer(x,y) = 1.0
- O frame buffer com a cor do fundo da cena.
  frameBuffer(x,y) = cor de fundo

# Z-Buffer

## Z-Buffer

- A medida em que cada face é renderizada, ou seja, os pixel são determinados através do algoritmo *scanline*.

```
1 //calcula (se necessario) a profundidade de z
2 //para cada pixel (x,y) da face.
3 if (z < depth(x,y)){
4      deph_buffer(x,y) = z;
5      frame_buffer(x,y) = cor pixel;
6 }
```

# Z-Buffer

## Z-Buffer

- Os valores de profundidade estão normalizados entre 0.0 e 1.0 com o plano de visão na profundidade 0.0.

```
1 //calcula (se necessario) a profundidade de z
2 //para cada pixel (x,y) da face.
3 if (z < depth(x,y)){
4      deph_buffer(x,y) = z;
5      frame_buffer(x,y) = cor pixel;
6 }
```

# Z-Buffer

## Z-Buffer

- **Implementação eficiente:** o valor de **profundidade de um pixel** em uma *scanline* pode ser calculado usando **o valor do pixel precedente** usando uma única adição.

- Depois de todas as faces processadas, o **depth buffer** contém as **profundidades** das superfícies visíveis , e o *frame buffer* contém as **cores** dessas superfícies.
  - Cena pronta para ser exibida.

# Z-Buffer

## Vantagem

- Simplicidade.

## Desvantagens

- Quantidade de **memória** necessária (em um sistema 1280 x 1024 precisa de 1.3mi de posições).
- Alguns **cálculos desnecessários**...por que?
- **Precisão limitada** para o cálculo de profundidade em cenas complexas pode ser um problema: quantização de valores de profundidade pode introduzir artefatos.

- Placas gráficas otimizam operações do Z-buffer.