Determinação de Superfícies Visíveis

SCC0250 - Computação Gráfica

Profa. Maria Cristina F. Oliveira

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

22 de novembro de 2023

Sumário

Sumário

Introdução

Rendering de Polígonos

- Apenas as faces que sobreviveram ao processo de recorte em relação ao *view volume* canônico serão consideradas nas etapas finais do pipeline gráfico
- Destas, por eficiência, apenas as faces poligonais que são visíveis para a câmera serão renderizadas

- Existem diversos algoritmos para **detecção de superfícies visíveis** (ou eliminação de superfícies ocultas) que variam conforme a...
  - complexidade da cena
  - tipo de objeto desenhado
  - equipamento disponível
  - etc.

Classificação dos Algoritmos

- Os algoritmos existentes podem ser classificados em duas categorias
  - Métodos que operam no **espaço do objeto**
  - Métodos que operam no **espaço de imagem**

Espaço do Objeto

- Comparam objetos entre si, ou partes deless, para determinar a visibilidade (abordagem opera no espaço geométrico)

Espaço de Imagem

- Compara os pixels sendo projetados no plano de projeção para determinar a visibilidade (abordagem opera no espaço da matriz de pixels)

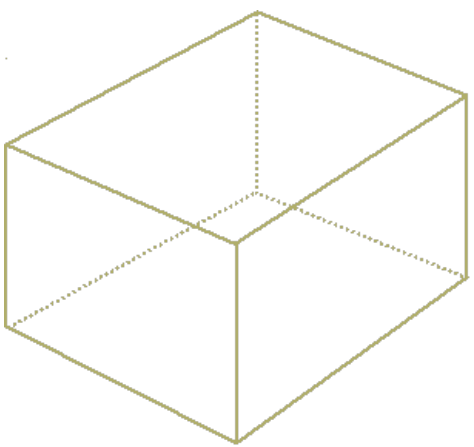
Classificação dos Algoritmos

- Discutiremos dois algoritmos para determinação de visibilidade
  - *Back-face culling*
  - *Z-buffer*
- *Back-face culling*: determina quais faces são visíveis para a câmera (serão rasterizadas)
- *Z-buffer*: determina, para as faces rasterizadas, quais são visíveis para o observador

Sumário

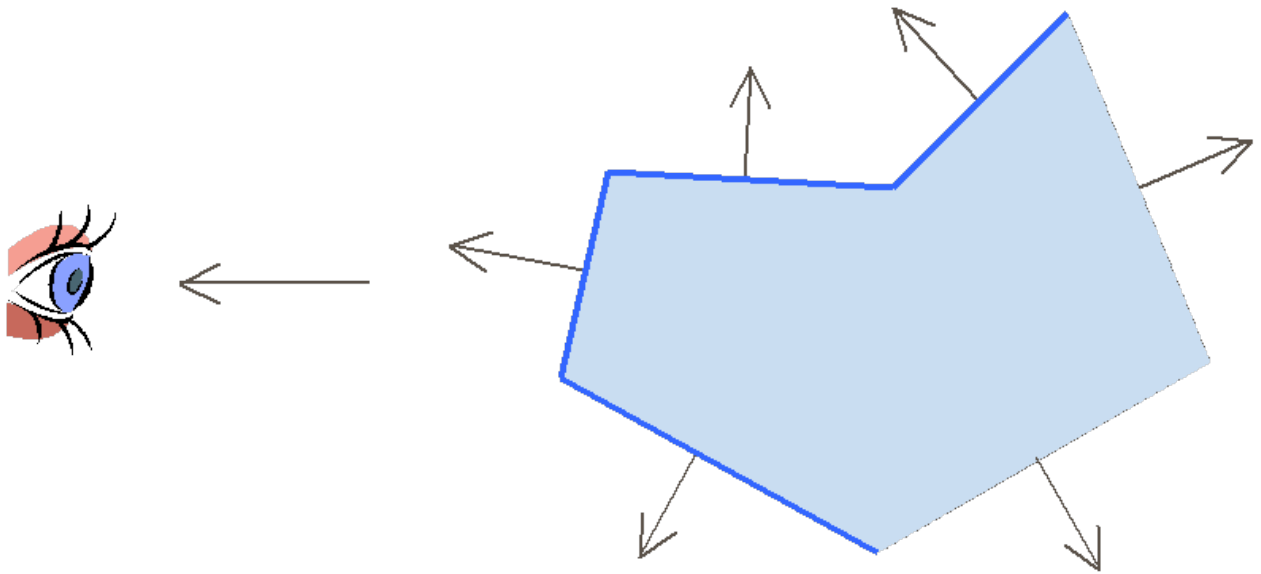
Back Face Culling

- Se as faces descrevem a superfície de um objeto sólido fechado (um poliedro, por exemplo), não é necessário renderizar as faces de trás do objeto (não visíveis)



- Apenas 3 faces precisam ser traçadas
- Faces "de trás" podem ser removidas do pipeline

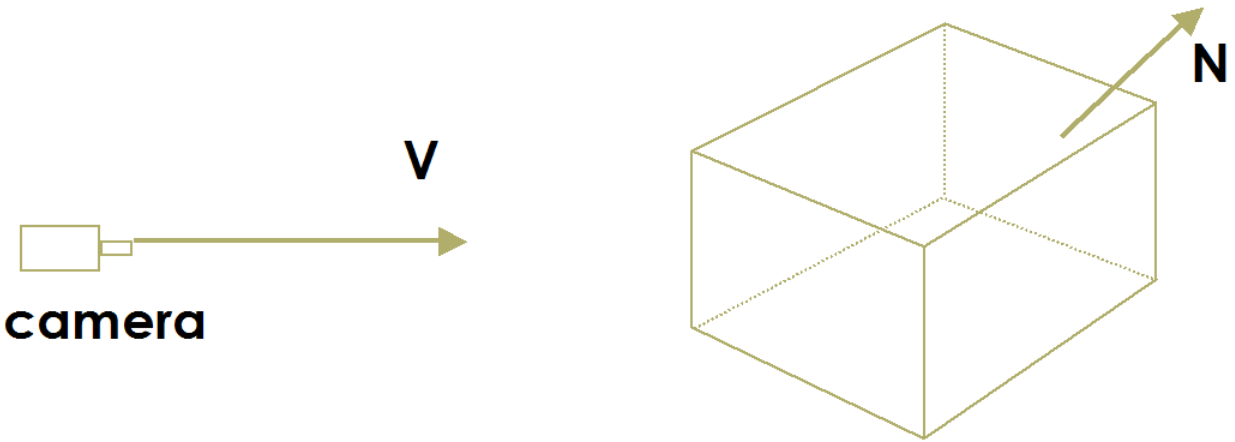
Back Face Culling



- Nota: processo assume que cena é composta por objetos poliéricos fechados

Back Face Culling

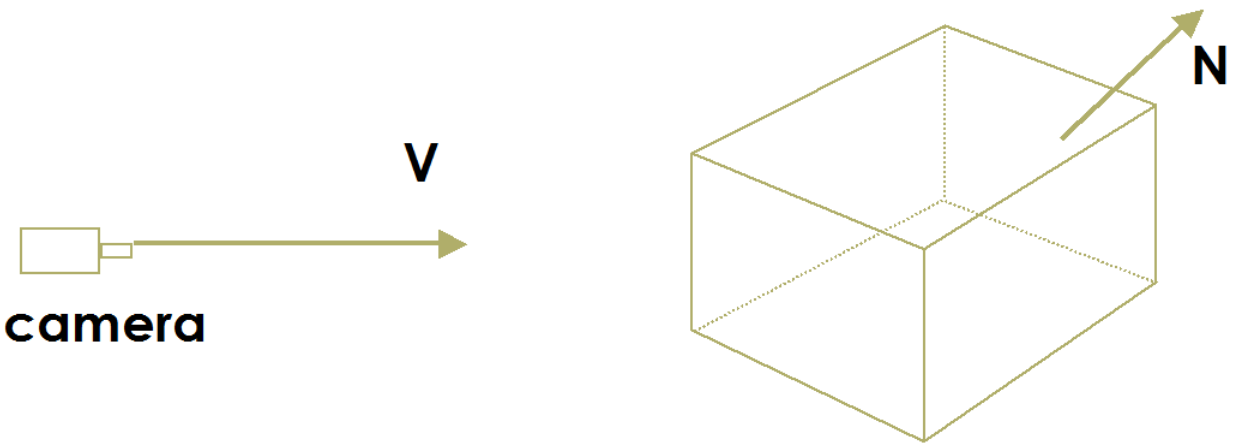
- Como descobrir quais são as "faces de trás"?



Back Face Culling

- Uma **face** é uma "face de trás" (**não visível**) de um polígono se o **ângulo** entre o vetor normal à face $\mathbf{N}$ e o vetor direção de observação $\mathbf{V}$ é **menor do que** 90^0

$$\mathbf{V} \cdot \mathbf{N} > 0$$



Back Face Culling

- Este teste pode ser feito de maneira eficiente após a transformação da cena para o sistema de coordenadas de observação (VCS)
  - Vetor direção de observação paralelo ao eixo z_v
  - Assim $\mathbf{V} = (0, 0, v_z)$, $\mathbf{N} = (n_x, n_y, n_z)$, $|\mathbf{V}| = |\mathbf{N}| = 1$
  - $\mathbf{V} \cdot \mathbf{N} = v_z \cdot n_z$

- Portanto, para fazer o teste $\mathbf{V} \cdot \mathbf{N} > 0$ basta verificar o sinal da componente z do vetor normal à face

Sumário

Back Face Culling

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

```
1 gl.glEnable(GL.GL_CULL_FACE);
2 gl.glCullFace(GL.GL_BACK);
```

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

Código OpenGL

```
1  public class BackFaceCulling implements GLEventListener {
2    ...
3
4    public static void main(String[] args) {
5      //cria o painel e adiciona um ouvinte GL
6      GLJPanel panel = new GLJPanel();
7      panel.addGLEventListener(new BackFaceCulling());
8
9      //cria uma janela e adiciona o painel
10     JFrame frame = new JFrame("Aplicao JOGL Simples");
11     frame.add(panel);
12     frame.setSize(400, 400);
13     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     frame.setLocationRelativeTo(null);
15     frame.setVisible(true);
16   }
17 }
```

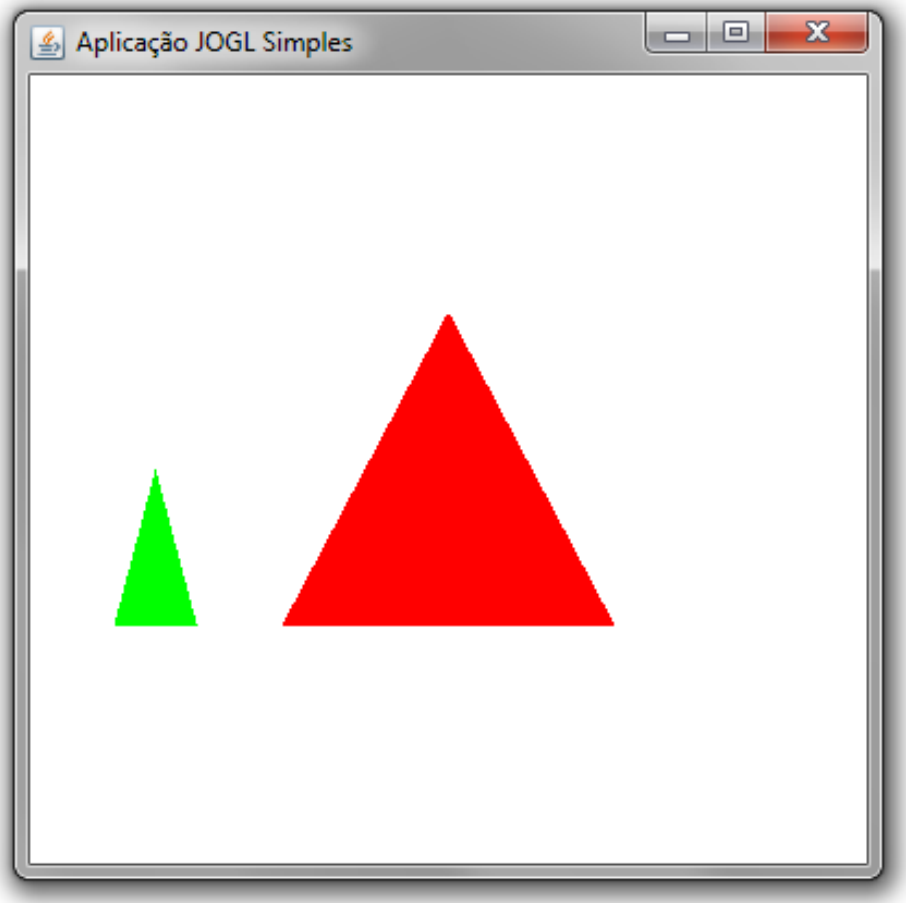
Código OpenGL

```
1  public class BackFaceCulling implements GLEventListener {
2    ...
3
4    public void display(GLAutoDrawable drawable) {
5      GL gl = drawable.getGL();
6
7      //limpa o buffer
8      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
9
10     //define que a matrix  a de modelo
11     gl.glMatrixMode(GL.GL_MODELVIEW);
12
13     gl.glColor3f(1.0f, 0.0f, 0.0f); //N . V < 0 (visvel)
14     gl.glBegin(GL.GL_TRIANGLES);
15     gl.glVertex3f(-2, -2, 0);
16     gl.glVertex3f(2, -2, 0);
17     gl.glVertex3f(0, 2, 0);
18     gl.glEnd();
19
20     gl.glColor3f(0.0f, 1.0f, 0.0f); //N . V > 0 (no visvel)
21     gl.glBegin(GL.GL_TRIANGLES);
22     gl.glVertex3f(-4, -3, -4);
23     gl.glVertex3f(-3.5f, -1, -4);
24     gl.glVertex3f(-2, -1, -4);
25     gl.glEnd();
26
27     //fora o desenho das primitivas
28     gl.glFlush();
29   }
30 }
```

Código OpenGL

```
1  public class BackFaceCulling implements GLEventListener {
2    ...
3
4    public void init(GLAutoDrawable drawable) {
5      GL gl = drawable.getGL();
6
7      gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo
8
9      gl.glMatrixMode(GL.GL_PROJECTION); //define que a matrix  a de projeo
10     gl.glLoadIdentity(); //carrega a matrix de identidade
11     gl.glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0); //define uma projeo ortogonal
12   }
13 }
```

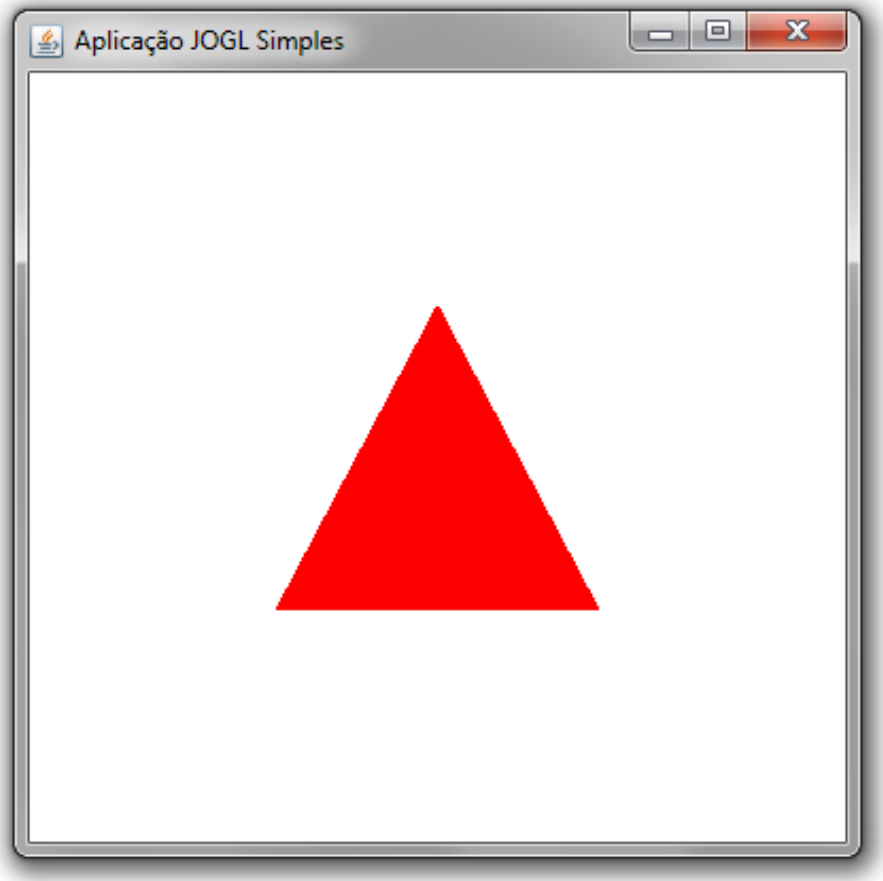
Código OpenGL



Código OpenGL

```
1
2  public class BackFaceCulling implements GLEventListener {
3    ...
4
5    public void init(GLAutoDrawable drawable) {
6      GL gl = drawable.getGL();
7
8      gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo
9
10     //ativa remoo de face oculta
11     gl.glEnable(GL.GL_CULL_FACE);
12     gl.glCullFace(GL.GL_BACK);
13
14     gl.glMatrixMode(GL.GL_PROJECTION); //define que a matrix  a de projeo
15     gl.glLoadIdentity(); //carrega a matrix de identidade
16     gl.glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0); //define uma projeo ortogonal
17   }
18 }
```

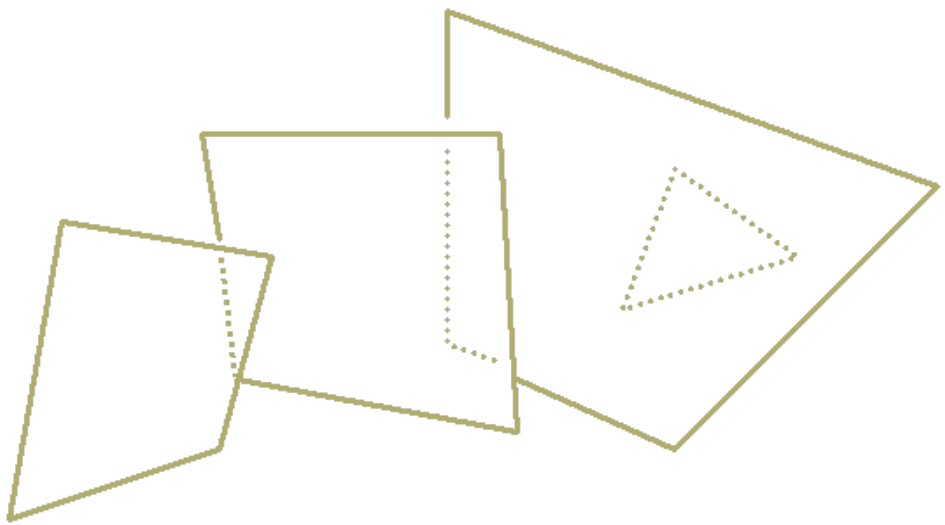
Código OpenGL



Sumário

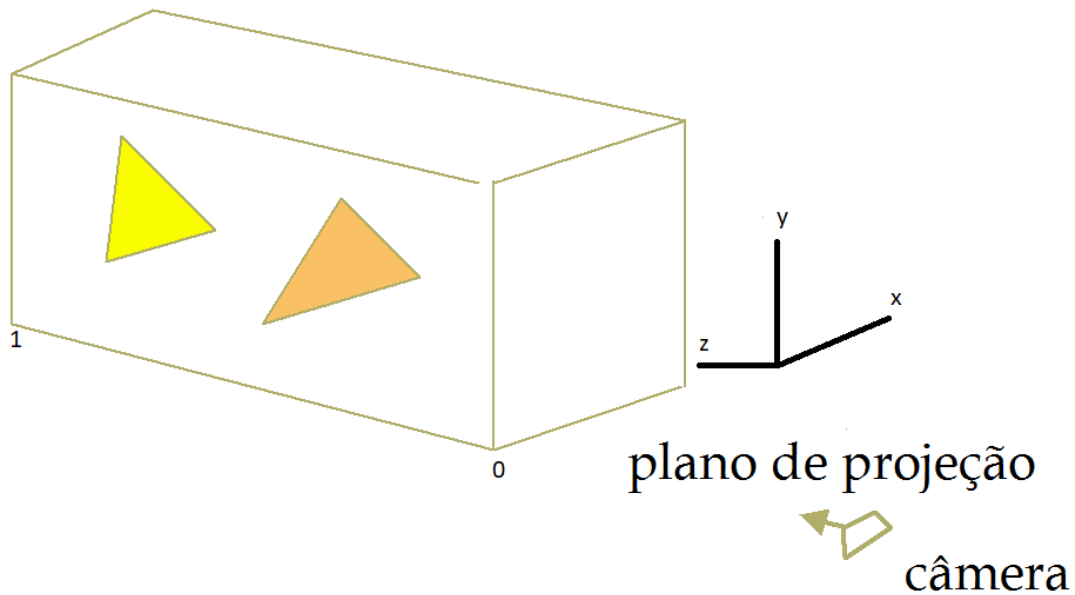
Z-Buffer: Determinação das Faces Visíveis/Ocultas

- Algumas faces da cena ficam ocultas atrás de outras: só os pixels das faces visíveis (ou das partes visíveis delas) serão de fato renderizados no frame buffer



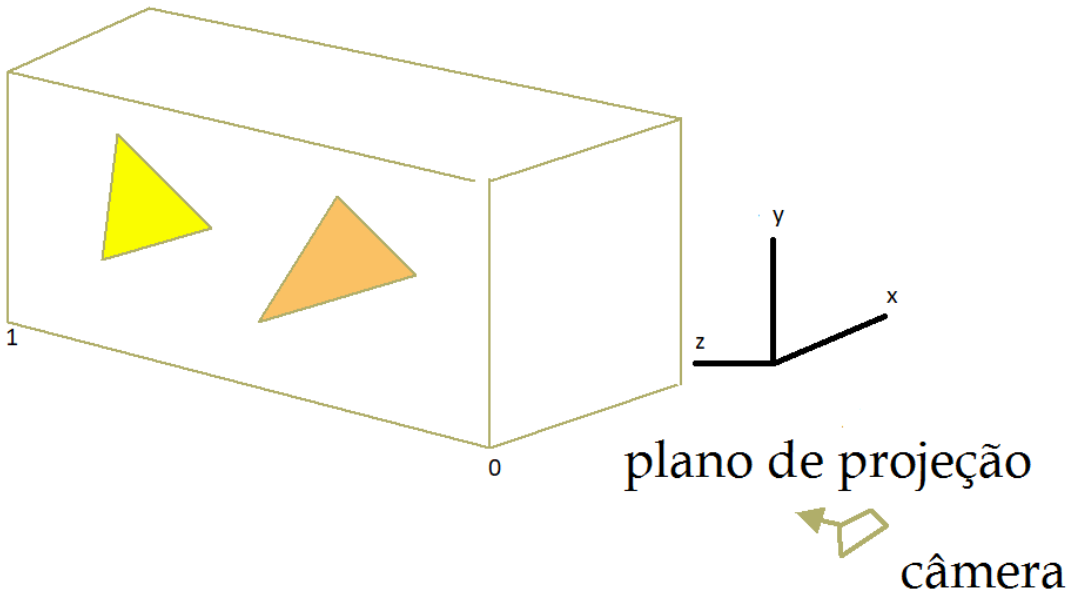
Algoritmo Z-Buffer

- Considere que as faces passaram pela transformação de projeção, e tiveram suas coordenadas z armazenadas
- Suponha os valores de z normalizados no intervalo 0 a 1 (0 plano *near* e 1 plano *far*)
- Múltiplas faces podem ser projetadas nos mesmos pixels da *viewport*



Algoritmo Z-Buffer

- Cada pixel (x, y) deve receber a cor calculada para a face mais próxima da câmera, i.e., com menor valor de coordenada z



Algoritmo Z-Buffer

- Algoritmo de rasterização usa dois *buffers*
  - **Frame Buffer**: armazena os valores RGB que definem a cor de cada pixel, tipicamente 24 bits, mais 8 bits para transparência (alfa)
  - **Z-Buffer**: mantém informação de profundidade associada a cada pixel, tipicamente 16, 24 ou 32 bits

Inicialização

- Todas posições do z-buffer são inicializadas com a maior profundidade

  depth_buffer(x,y) = 1.0

- O frame buffer é inicializado com a cor de fundo da cena

  frame_buffer(x,y) = cor de fundo

Algoritmo Z-Buffer

- A medida em que cada face é rasterizada, ou seja, os pixels são determinados com o algoritmo *scanline*

```
1  //determina (se necessrio) a profundidade z associada a cada pixel (x,y) da face
2
3  if(z < depth(x,y)){
4    depth_buffer(x,y) = z;
5    frame_buffer(x,y) = cor do pixel;
6  }
```

- Note que os valores de profundidade (z) estão normalizados entre 0.0 e 1.0, assumindo o plano de visão na profundidade 0.0

Algoritmo Z-Buffer

- **Implementação eficiente**: o valor de **profundidade de um pixel** em uma *scanline* pode ser calculado a partir do **valor de profundidade do pixel precedente**, com uma única operação de adição
- Cálculo incremental

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

Algoritmo Z-Buffer

Vantagem

- Simplicidade

Desvantagens

- Quantidade de **memória** necessária (em um sistema 1.280×1.024 precisa de 1.3 milhões de posições)
- Alguns **cálculos desnecessários** são executados... porque?
- A **precisão limitada** disponível para armazenar os valores de z pode ser um problema em cenas complexas: quantização de valores de profundidade pode introduzir artefatos

- Placas gráficas otimizam as operações no z-buffer

Sumário

Algoritmo Z-Buffer

- Habilitar z-buffer

```
1 gl.glEnable (GL.GL_DEPTH_TEST);
```

- Ao gerar um novo quadro, o z-buffer deve ser limpo

```
1 gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT)
```

Código OpenGL

```
1  public class ZBuffer implements GLEventListener {
2    ...
3
4    public static void main(String[] args) {
5      //cria o painel e adiciona um ouvinte GL
6      GLJPanel panel = new GLJPanel();
7      panel.addGLEventListener(new ZBuffer());
8
9      //cria uma janela e adiciona o painel
10     JFrame frame = new JFrame("Aplicao JOGL Simples");
11     frame.add(panel);
12     frame.setSize(400, 400);
13     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     frame.setLocationRelativeTo(null);
15     frame.setVisible(true);
16   }
17 }
```

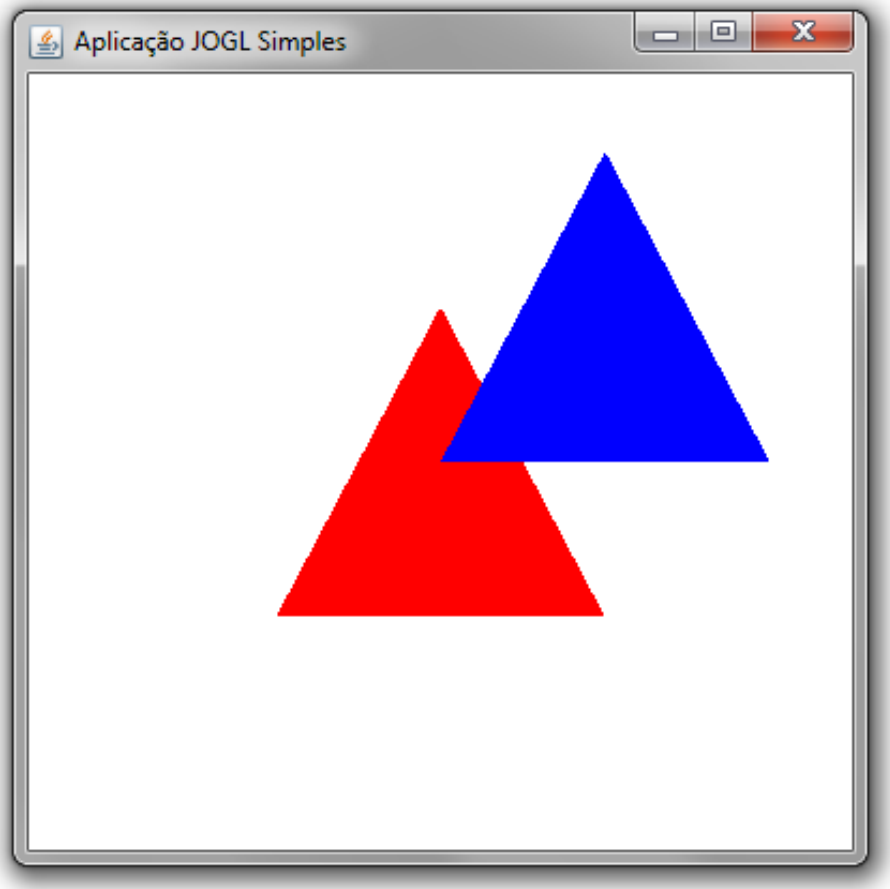
Código OpenGL

```
1  public class ZBuffer implements GLEventListener {
2
3    public void display(GLAutoDrawable drawable) {
4      GL gl = drawable.getGL();
5
6      //limpa o buffer
7      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
8
9      //define que a matrix  a de modelo
10     gl.glMatrixMode(GL.GL_MODELVIEW);
11
12     //tringulo na frente
13     gl.glColor3f(1.0f, 0.0f, 0.0f); //N . V < 0 (visvel)
14     gl.glBegin(GL.GL_TRIANGLES);
15     gl.glVertex3f(-2, -2, 0);
16     gl.glVertex3f(2, -2, 0);
17     gl.glVertex3f(0, 2, 0);
18     gl.glEnd();
19
20     //tringulo atrs
21     gl.glColor3f(0.0f, 0.0f, 1.0f); //N . V < 0 (visvel)
22     gl.glBegin(GL.GL_TRIANGLES);
23     gl.glVertex3f(0, 0, -2);
24     gl.glVertex3f(4, 0, -2);
25     gl.glVertex3f(2, 4, -2);
26     gl.glEnd();
27
28     //fora o desenho das primitivas
29     gl.glFlush();
30   }
31 }
```

Código OpenGL

```
1  public class ZBuffer implements GLEventListener {
2    ...
3
4    public void init(GLAutoDrawable drawable) {
5      GL gl = drawable.getGL();
6      gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo
7
8      //habilita remoo de faces ocultas
9      gl.glEnable(GL.GL_CULL_FACE);
10     gl.glCullFace(GL.GL_BACK);
11
12     gl.glMatrixMode(GL.GL_PROJECTION); //define que a matrix  a de projeo
13     gl.glLoadIdentity(); //carrega a matrix de identidade
14     gl.glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0); //define uma projeo ortogonal
15   }
16 }
```

Código OpenGL



Código OpenGL

```
1  public class ZBuffer implements GLEventListener {
2    ...
3
4    public void init(GLAutoDrawable drawable) {
5      GL gl = drawable.getGL();
6      gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo
7
8      //habilita o teste de profundidade
9      gl.glEnable(GL.GL_DEPTH_TEST);
10
11     //habilita remoop de faces ocultas
12     gl.glEnable(GL.GL_CULL_FACE);
13     gl.glCullFace(GL.GL_BACK);
14
15     gl.glMatrixMode(GL.GL_PROJECTION); //define que a matrix  a de projeo
16     gl.glLoadIdentity(); //carrega a matrix de identidade
17     gl.glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0); //define uma projeo ortogonal
18   }
19 }
```

Código OpenGL