UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Superfícies Visíveis

Disciplina: Computação Gráfica (BCC35F)
Curso: Ciência da Computação

Prof. Walter T. Nakamura
waltertakashi@utfpr.edu.br

Campo Mourão - PR

Baseados nos materiais elaborados pelas professoras Aretha Alencar (UTFPR) e Rosane Minghim (USP)

Sumário

1) Introdução
2) Back Face Culling
   - Programação OpenGL
3) Algoritmo Z-Buffer
   - Programação OpenGL

Sumário

1) Introdução
2) Back Face Culling
   - Programação OpenGL
3) Algoritmo Z-Buffer
   - Programação OpenGL

Introdução

- **Rendering de Polígonos**
  - Por eficiência, só queremos **renderizar** 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

- Os algoritmos existentes podem ser classificados em dois grandes grupos:
  - **Métodos de espaço do objeto** – Compara objetos entre si, ou partes de objetos, para determinar a visibilidade
  - **Métodos de espaço da imagem** – Compara pixel por pixel no plano de projeção para determinar a visibilidade

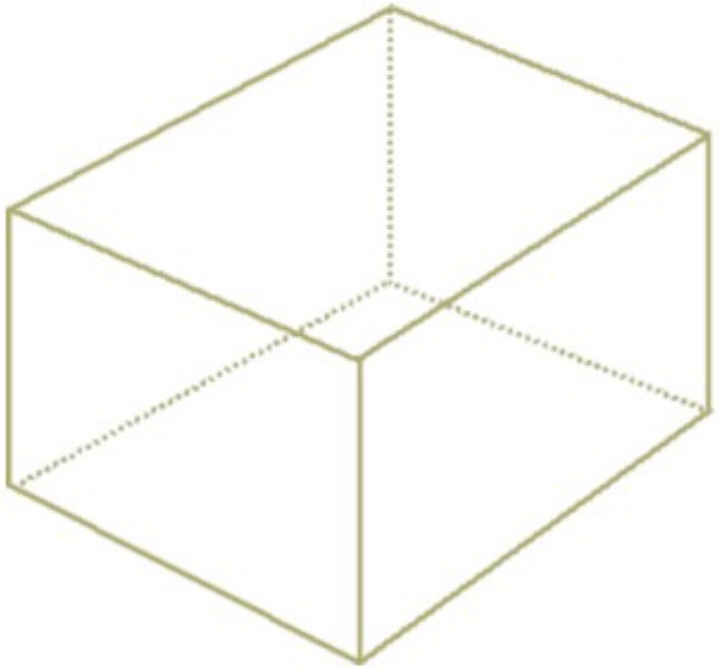
Introdução

- Discutiremos dois algoritmos para visibilidade:
  - **Back Face culling:** método de espaço de objeto simples e rápido para identificar as faces traseiras de um poliedro
  - **Z-buffer:** método de espaço de imagem comumente usado para detectar superfícies visíveis, que compara profundidades das superfícies para cada pixel projetado no plano de projeção

Sumário

Back Face Culling

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

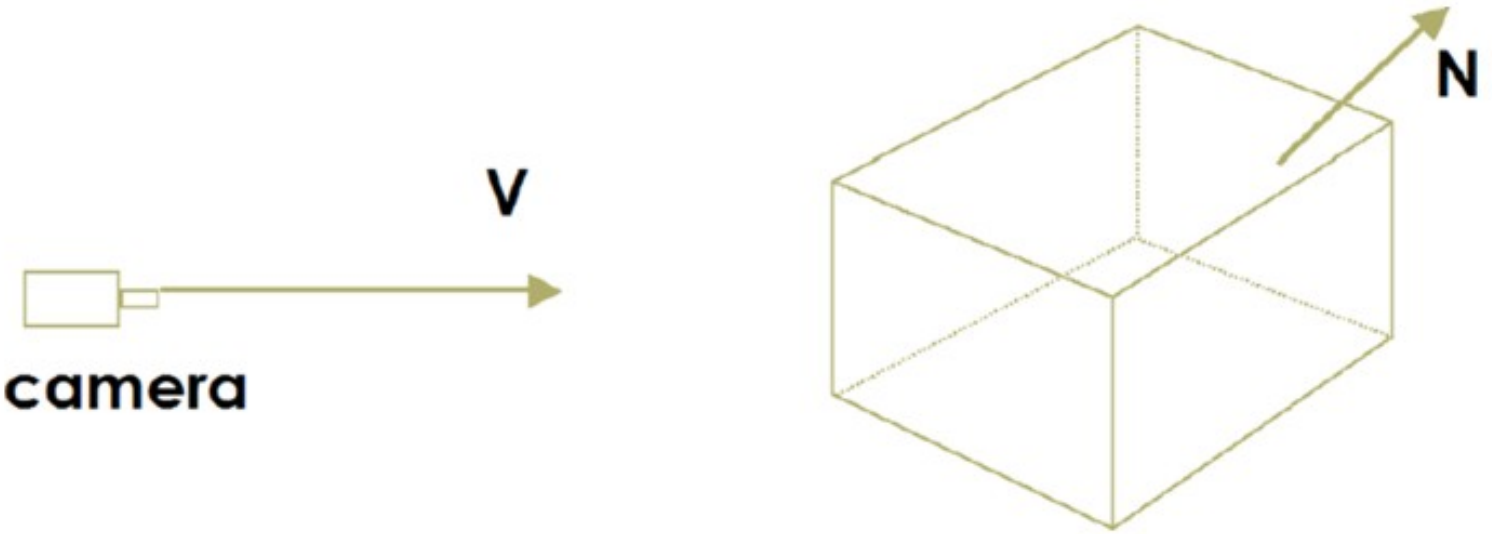


- Apenas 3 faces precisam ser traçadas
- Faces de trás podem ser removidas do pipeline

Back Face Culling

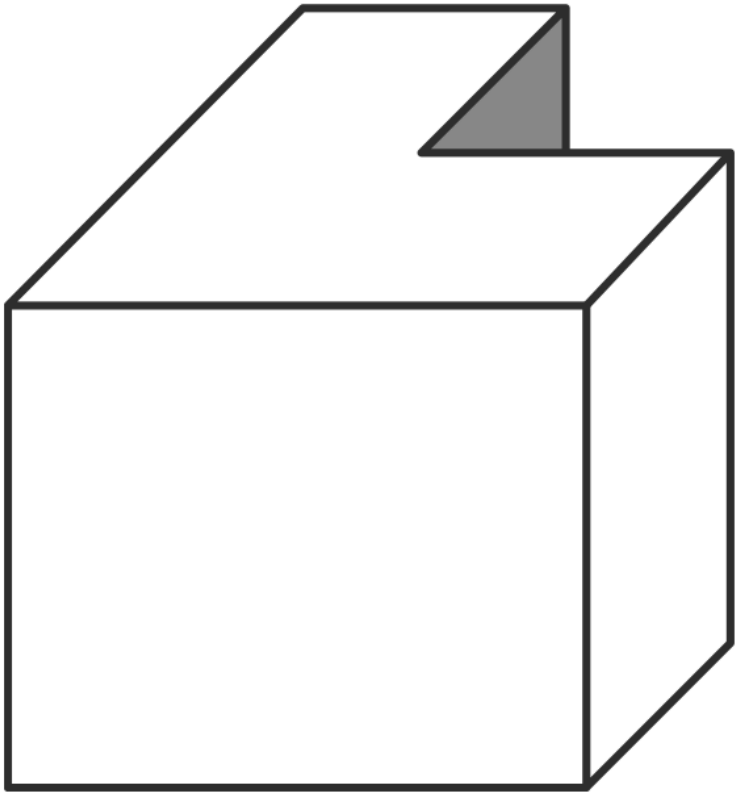
- Como descobrir quais são as “**faces de trás**”?
  - Uma face é uma “face de trás” (**não visível**) de um polígono se o ângulo entre o vetor normal à face N e o vetor direção de observação V é menor do que 90º.

$$V \cdot N > 0$$



Back Face Culling

- O processo assume que a cena é composta por objetos poliédricos convexos fechados



Sumário

1) Introdução
2) Back Face Culling
   - Programação OpenGL
3) Algoritmo Z-Buffer
   - Programação OpenGL

Back Face Culling

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

```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```

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

Exemplo 1 – Back Face Culling

```
#include <GL/glut.h>

void init(){
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo
    glMatrixMode(GL_PROJECTION); //define que a matriz é a de projeção
    glLoadIdentity(); //carrega a matriz de identidade
    glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0); //define uma projeção
ortogonal
}
```

Exemplo 1 – Back Face Culling

```
void display(void){
    //limpa o buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //define que a matriz é a de modelo
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glColor3f(1.0f, 0.0f, 0.0f); //N . V < 0 (visível)
    glBegin(GL_TRIANGLES);
        glVertex3f(0, 2, 0);
        glVertex3f(-2, -2, 0);
        glVertex3f(2, -2, 0);
    glEnd();

    glColor3f(0.0f, 1.0f, 0.0f); //N . V > 0 (não visível)
    glBegin(GL_TRIANGLES);
        glVertex3f(-3.5f, 0, -4);
        glVertex3f(-3, -2, -4);
        glVertex3f(-4, -2, -4);
    glEnd();

    //força o desenho das primitivas
    glFlush();
}
```

Exemplo 1 – Back Face Culling

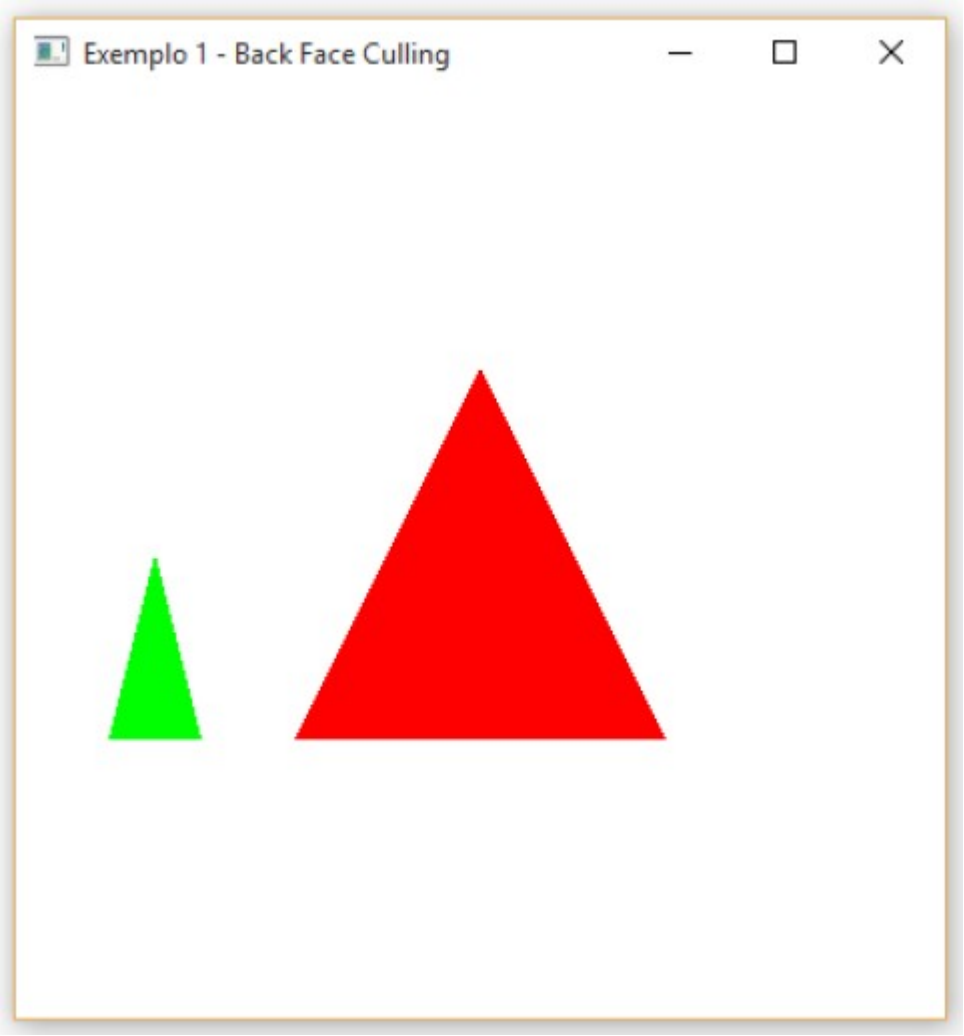
```
int main(int argc, char **argv){
  /* cria uma janela */
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH); /* single buffering */
  glutInitWindowSize(400, 400);
  glutCreateWindow("Exemplo 1 – Back Face Culling");

  glutDisplayFunc(display);

  init(); //executa funcao de inicializacao

  glutMainLoop(); //mostre tudo e espere
  return(0);
}
```

Exemplo 1 – Back Face Culling



Exemplo 1 – Back Face Culling

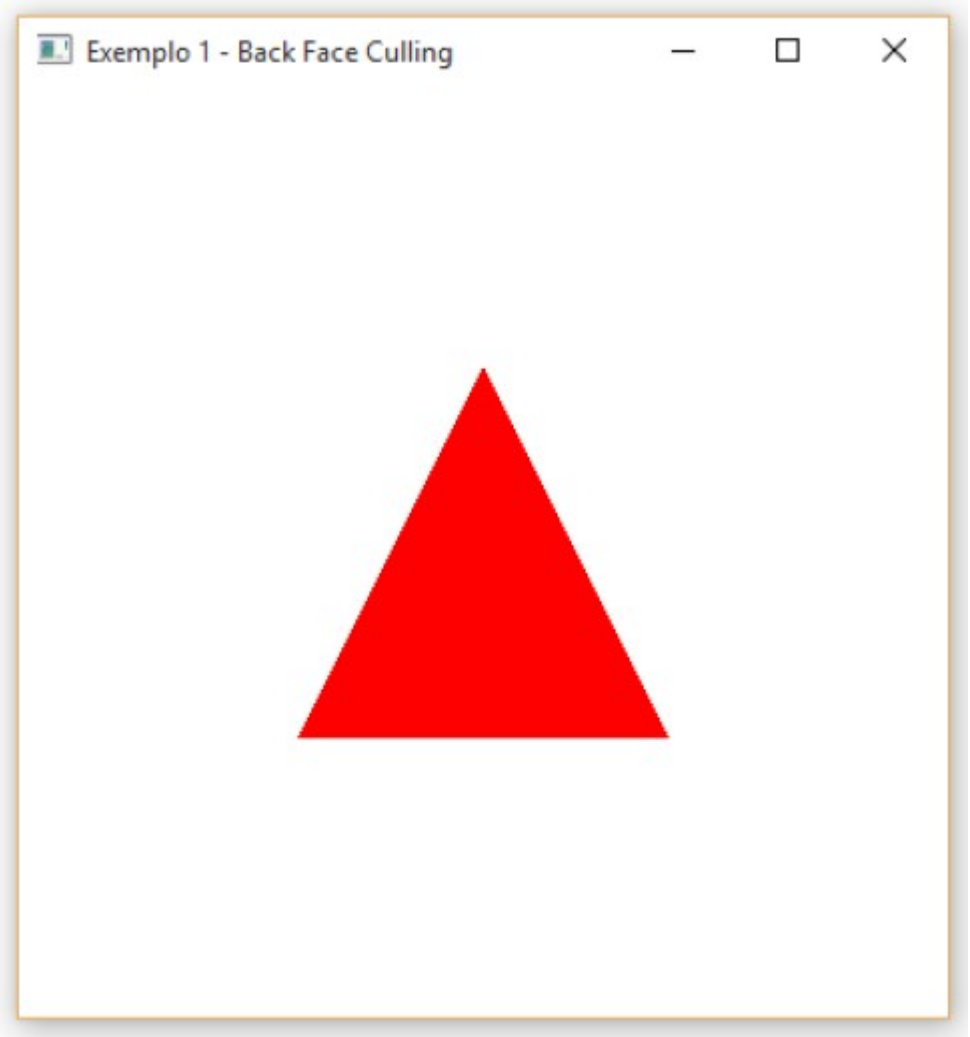
```
#include <GL/glut.h>

void init(){
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo

    //habilita remoção de faces ocultas
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);

    glMatrixMode(GL_PROJECTION); //define que a matriz é a de projeção
    glLoadIdentity(); //carrega a matriz de identidade
    glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0); //define uma projeção
ortogonal
}
```

Exemplo 1 – Back Face Culling



Exemplo 1 – Back Face Culling

- Por padrão, o Face Culling considera que a polígonos cujos vértices são definidos no **sentido anti-horário** representam a face frontal
  - Polígonos cujos vértices são definidos no **sentido horário** são considerados como a face traseira
  - É possível alterar o padrão utilizando o comando:

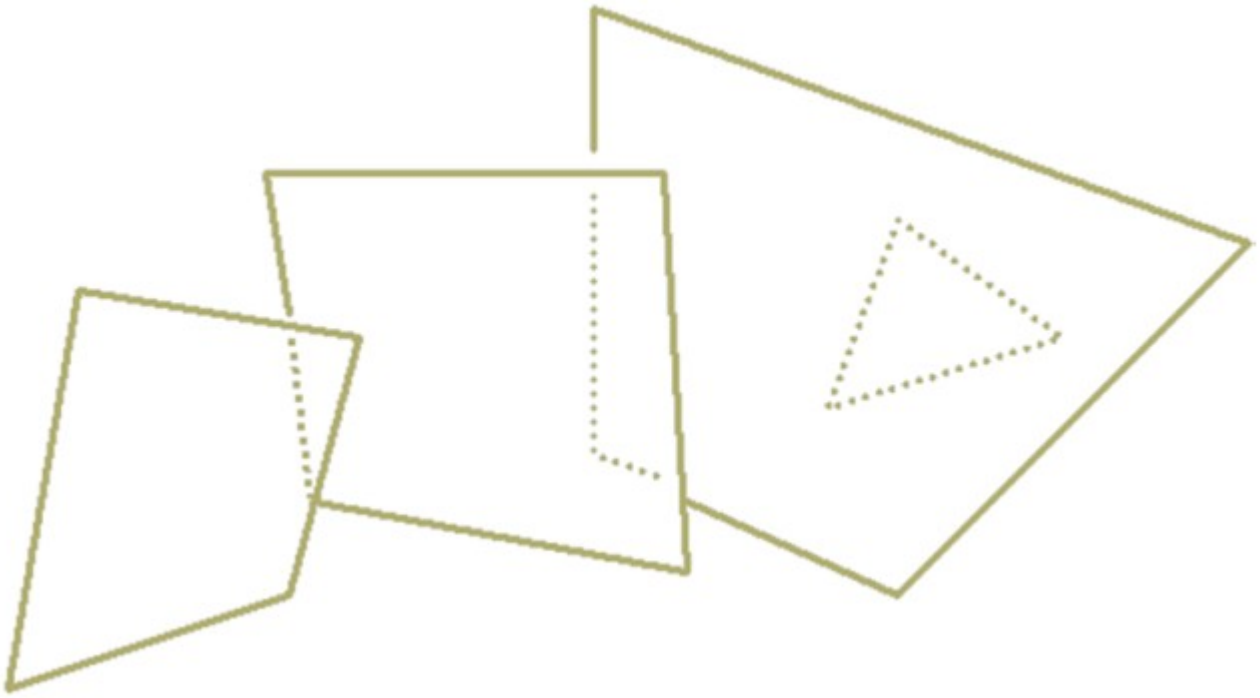
```
glFrontFace(GL_CW) # sentido horário
glFrontFace(GL_CCW) # sentido anti-horário (padrão)
```

Sumário

1) Introdução
2) Back Face Culling
   - Programação OpenGL
3) Algoritmo Z-Buffer
   - Programação OpenGL

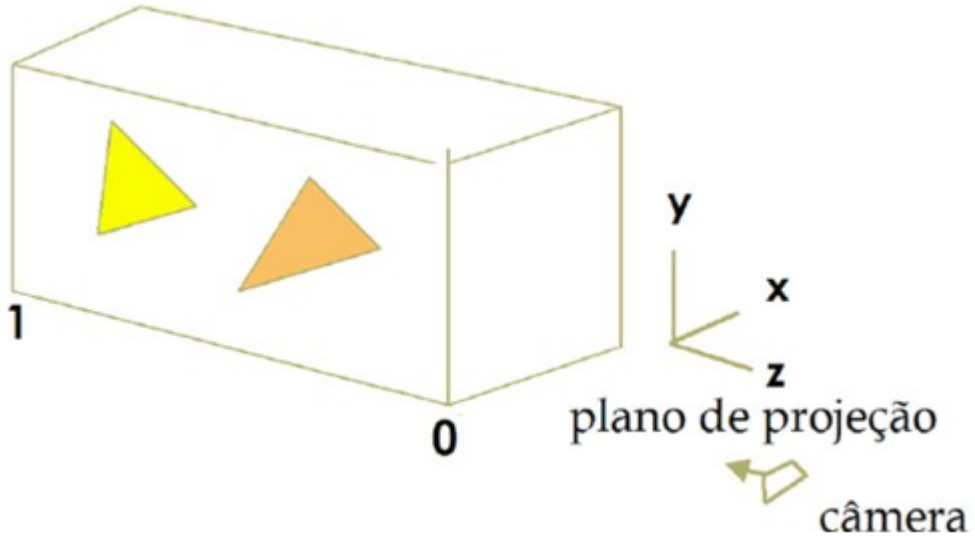
Outro Problema: Faces Ocultas

- Algumas faces da cena ficam ocultas atrás de outras: só queremos desenhar (renderizar) as faces (ou partes delas) realmente visíveis



Solução: Algoritmo Z-Buffer (ou Depth-Buffer)

- Considere que as faces passaram pela transformação de projeção, e tiveram suas coordenadas z armazenadas – suponha valores de z normalizados no intervalo 0 a 1 (0 plano *near* e 1 plano *far*)
- Para cada pixel (x, y), queremos traçar a **face mais próxima da câmera**, ou seja, com menor valor de z



Algoritmo Z-Buffer

- Algoritmo usa dois buffers:
  - **Frame Buffer:** armazena os valores RGB que definem a cor de cada pixel
  - **Z-Buffer:** mantém informação de profundidade associada a cada pixel
- Inicialização
  - Todas posições do **z-buffer** são inicializadas com a maior profundidade:

```
depth_buffer(x,y) = 1.0
```

  - Todas as posições do frame buffer são inicializadas com a cor de fundo da cena:

```
frame_buffer(x,y) = cor_de_fundo
```

Algoritmo Z-Buffer

- À medida em que cada face é renderizada, ou seja, os pixels são determinadas através do algoritmo de scanline:

```
/* calcula (se necessário) a profundidade z para cada
pixel projetado (xx,y) da face */

if (z < depth_buffer(x,y)){
   depth_buffer(x,y) = z;
   frame_buffer(x,y) = cor_do_pixel;
}
```

- Note que os valores de profundidade estão normalizados entre 0.0 e 1.0 com 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 usando o *valor do pixel precedente* usando uma única adição
- 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 pronta para ser exibida está no *frame-buffer*

Algoritmo Z-Buffer

- **Vantagens**
  - Simples e comumente implementado em hardware
  - Objetos podem ser desenhados em qualquer ordem
  - Placas gráficas otimizam operações no Z-Buffer
- **Desvantagens**
  - Quantidade de memória necessária (em um sistema 1280×1024 precisa 1.3 milhões de posições)
  - Alguns cálculos desnecessários, devido a ordem arbitrária do processamento dos objetos

Sumário

1) Introdução
2) Back Face Culling
   - Programação OpenGL
3) Algoritmo Z-Buffer
   - Programação OpenGL

Algoritmo Z-Buffer

- Na função inicialização do GLUT, incluir uma requisição para criar um depth buffer. Por exemplo:

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
```

- Habilitar z-buffer:

```
glEnable(GL_DEPTH_TEST);
```

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

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Exemplo 2 – Algoritmo Z-Buffer

```
#include <GL/glut.h>

void init(){
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo

    //habilita remoção de faces ocultas
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity(); //carrega a matriz de identidade
    glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0); //projeção ortogonal
}
```

Exemplo 2 – Algoritmo Z-Buffer

```
void display(void){
    //limpa o buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //define que a matriz é a de modelo
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    //triângulo na frente
    glColor3f(1.0f, 0.0f, 0.0f); //N . V < 0 (visível)
    glBegin(GL_TRIANGLES);
        glVertex3f(0, 2, 0);
        glVertex3f(-2, -2, 0);
        glVertex3f(2, -2, 0);
    glEnd();

    //triângulo atrás
    glColor3f(0.0f, 0.0f, 1.0f); //N . V < 0  (visível)
    glBegin(GL_TRIANGLES);
        glVertex3f(2, 4, -2);
        glVertex3f(0, 0, -2);
        glVertex3f(4, 0, -2);
    glEnd();

    //força o desenho das primitivas
    glFlush();
}
```

Exemplo 2 – Algoritmo Z-Buffer

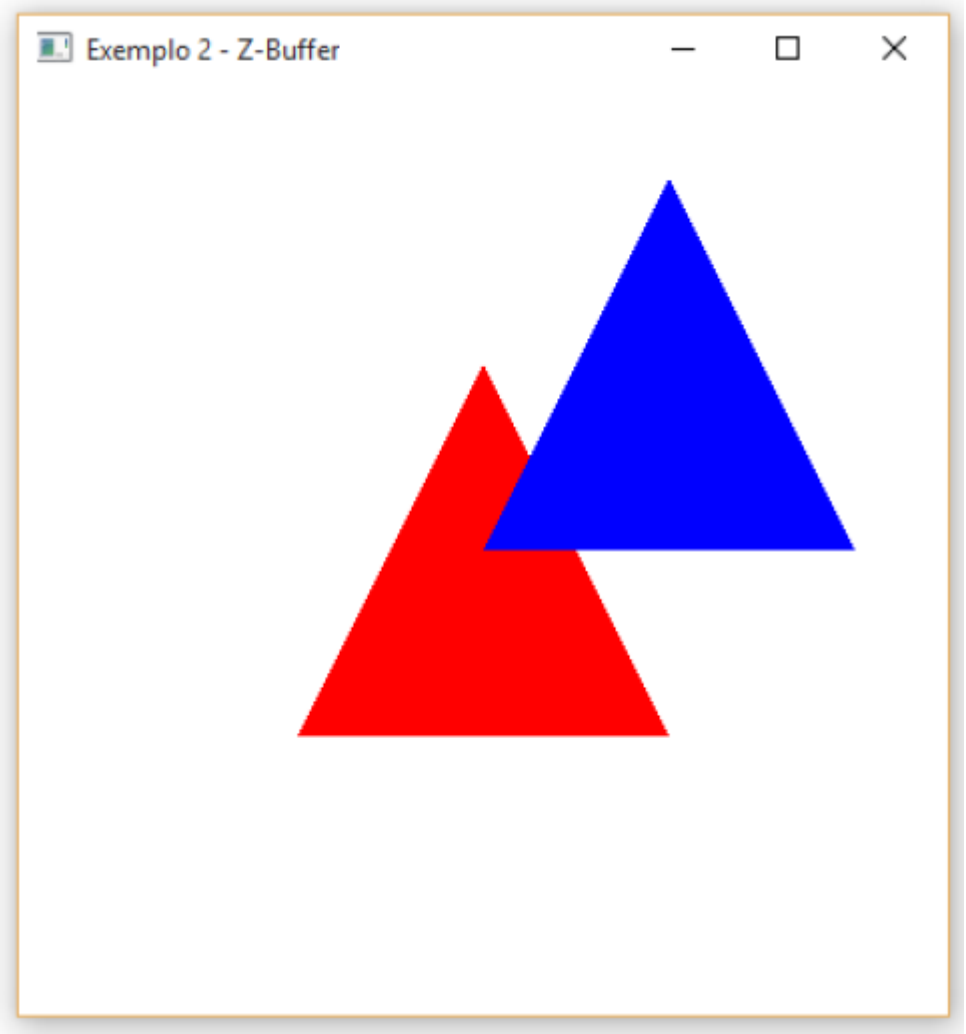
```
int main(int argc, char **argv){
  /* cria uma janela */
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH); /* single buffering */
  glutInitWindowSize(400, 400);
  glutCreateWindow("Exemplo 2 - Z-Buffer");

  glutDisplayFunc(display);

  init(); //executa funcao de inicializacao

  glutMainLoop(); //mostre tudo e espere
  return(0);
}
```

Exemplo 2 – Algoritmo Z-Buffer



Exemplo 2 – Algoritmo Z-Buffer

```
#include <GL/glut.h>

void init(){
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo

    //habilita o teste de profundidade
    glEnable(GL_DEPTH_TEST);

    //habilita remoção de faces ocultas
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity(); //carrega a matriz de identidade
    glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0); //projeção ortogonal
}
```

Exemplo 2 – Algoritmo Z-Buffer