OpenGL

Ricardo Dutra da Silva
Elisa de Cássia Silva Rodrigues

Universidade Tecnológica Federal do Paraná
Universidade Federal de Itajubá

2020

Primitivas OpenGL

- Além da `GL_TRIANGLES`, o OpenGL fornece outras primitivas.
- Em especial, algumas importantes são:
  - `GL_POINTS`,
  - `GL_LINES`.
- Veremos estas e outras no arquivo `primitives.ccp`.

Primitivas OpenGL

- A escolha do tipo de primitiva é feita no programa usando os caracteres '1' a '7' do teclado.

Exemplo

```
void keyboard(unsigned char key, int x, int y)
   ...
   case '1':
          type_primitive = GL_POINTS;
   ...
   case '7':
          type_primitive = GL_TRIANGLE_FAN;
```

Primitivas OpenGL

- Ao final da função é chamamos `glutPostRedisplay` para sinalizar redesenho.
- Função `display` deve ser executada.

Exemplo

```
void keyboard(unsigned char key, int x, int y)
    ...
    glutPostRedisplay();
}
```

Primitivas OpenGL

- Vamos analisar como cada primitiva é desenhada usando umas sequência de vértices $v_1, v_2, \ldots, v_n$.
- Definimos os seguintes vértices para o programa.

Exemplo

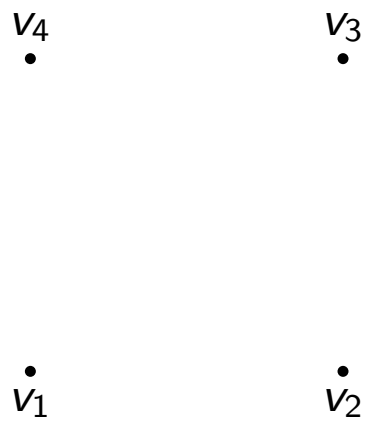
```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```

Primitivas OpenGL

- A primitiva `GL_POINTS` desenha cada vértice $v_1, v_2, \ldots, v_n$ como um ponto.

Exemplo

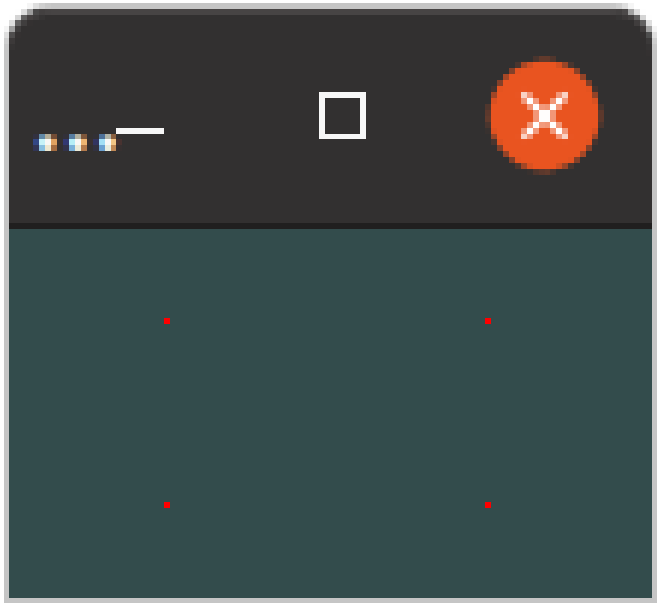
```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```



Primitivas OpenGL

Exemplo

```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```

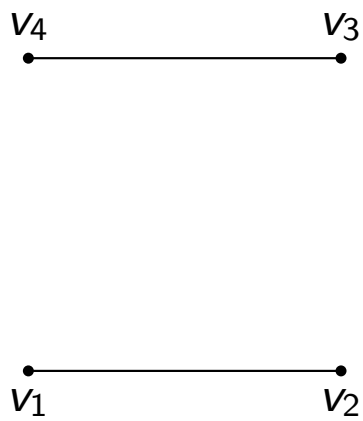


Primitivas OpenGL

- A primitiva `GL_LINES` desenha linhas para os pares: $(v_1, v_2), (v_3, v_4), \ldots, (v_{n-1}, v_n)$.
- Se n é ímpar um par não é formado, logo não há linha que o contenha.

Exemplo

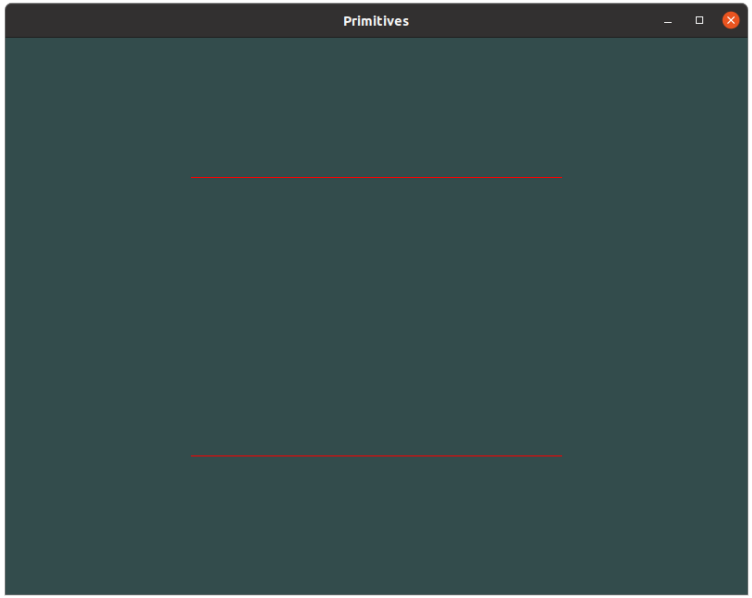
```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```



Primitivas OpenGL

Exemplo

```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```

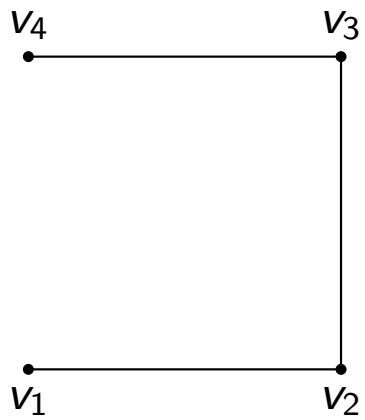


Primitivas OpenGL

- A primitiva `GL_LINE_STRIP`desenha linhas para os pares: $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$.

Exemplo

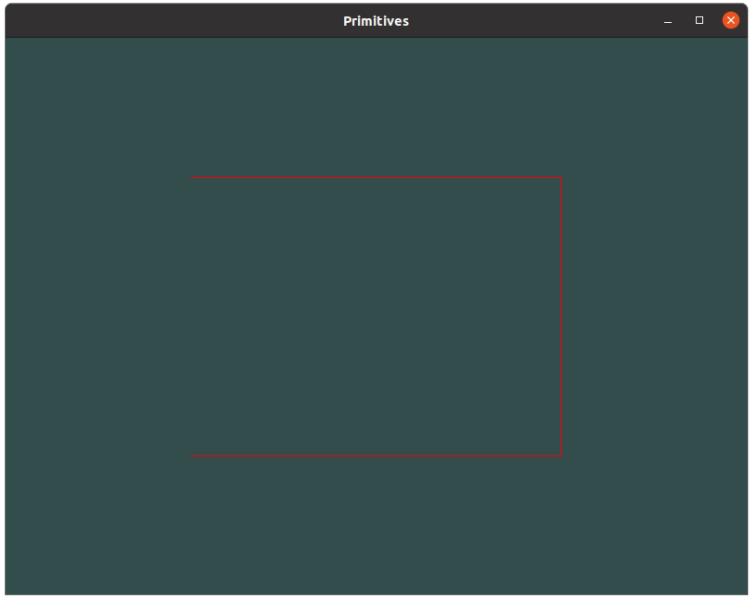
```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```



Primitivas OpenGL

Exemplo

```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```

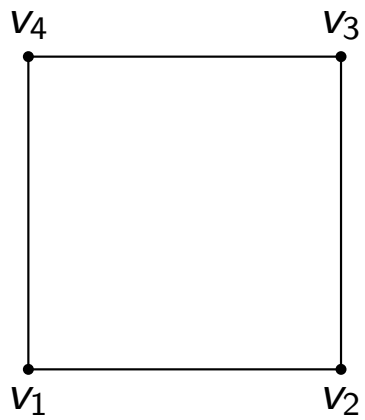


Primitivas OpenGL

- A primitiva `GL_LINE_LOOP`desenha linhas para os pares: $(v_1, v_2), (v_2, v_3), \ldots, (v_n, v_1)$.

Exemplo

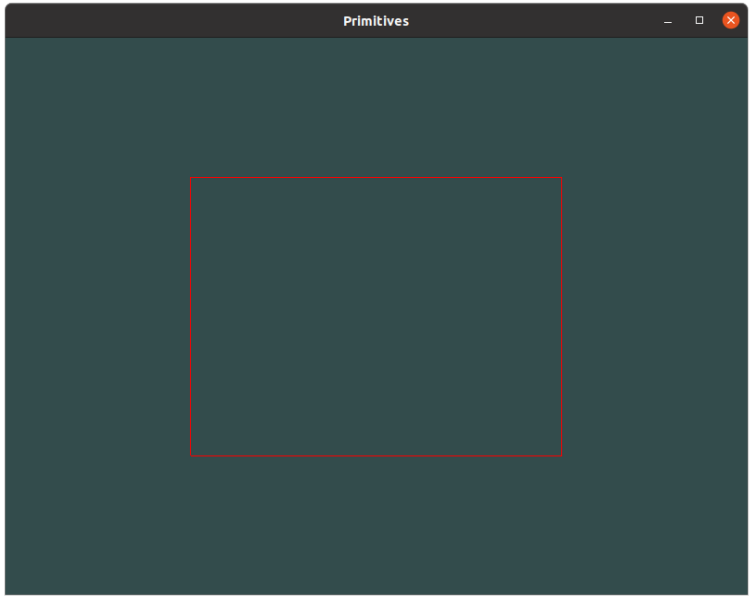
```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```



Primitivas OpenGL

Exemplo

```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```

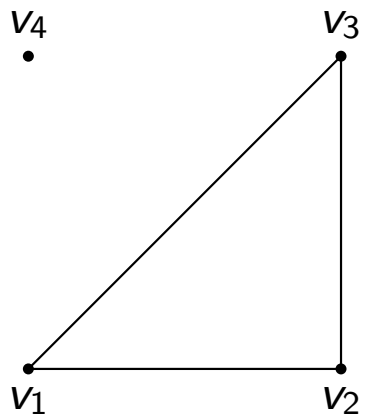


Primitivas OpenGL

- A primitiva `GL_TRIANGLES` desenha triângulos para as triplas: $(v_1, v_2, v_3), (v_3, v_4, v_5), \ldots, (v_{n-2}, v_{n-1}, v_n)$.
- Se n não é múltiplo de três, os pontos que não formam uma tripla são ignorados.

Exemplo

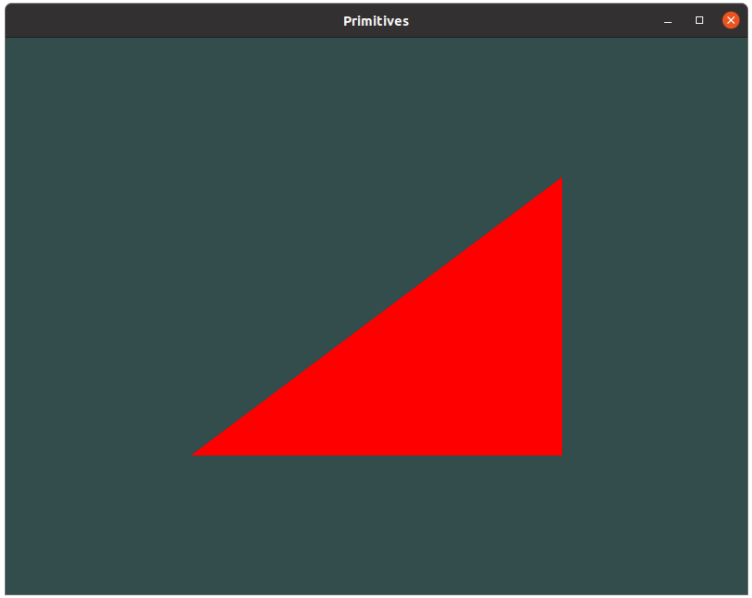
```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```



Primitivas OpenGL

Exemplo

```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```

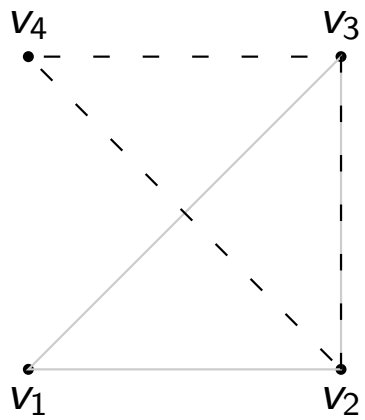


Primitivas OpenGL

- A primitiva `GL_TRIANGLE_STRIP` desenha triângulos para as triplas: $(v_1, v_2, v_3), (v_2, v_3, v_4), \ldots, (v_{n-2}, v_{n-1}, v_n)$.

Exemplo

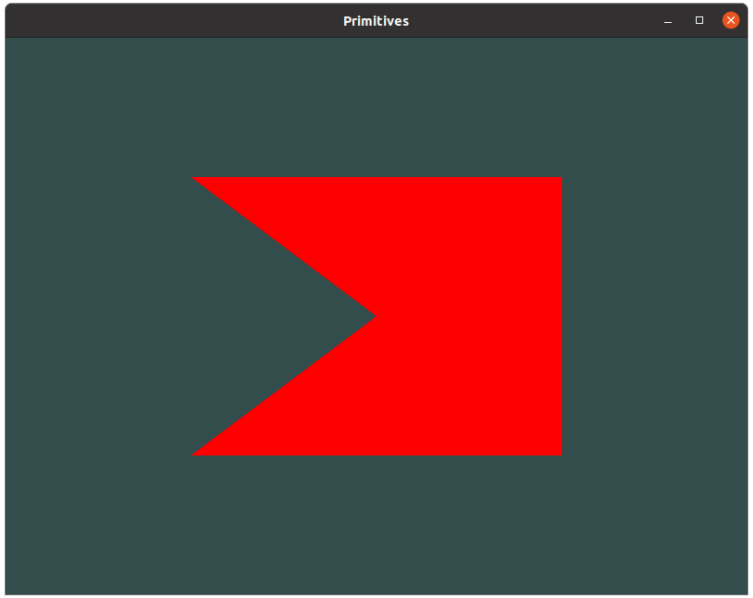
```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```



Primitivas OpenGL

Exemplo

```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```

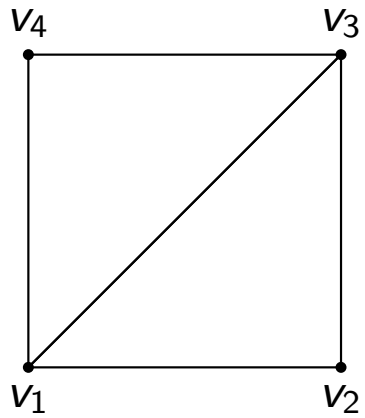


Primitivas OpenGL

- A primitiva `GL_TRIANGLE_FAN` desenha triângulos para as triplas: $(v_1, v_2, v_3), (v_1, v_3, v_4), \ldots, (v_1, v_{n-1}, v_n)$.

Exemplo

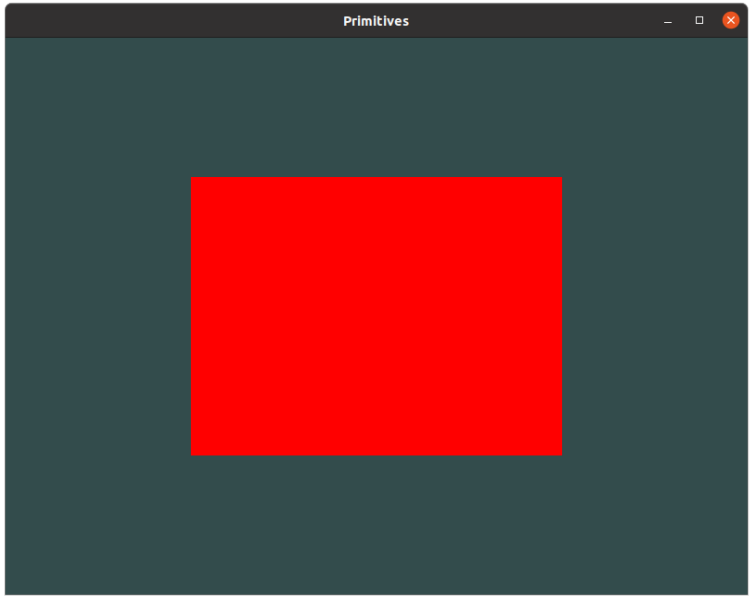
```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```



Primitivas OpenGL

Exemplo

```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
     0.5f,  0.5f, 0.0f, //v3
    -0.5f,  0.5f, 0.0f  //v4
};
```



Objetos complexos

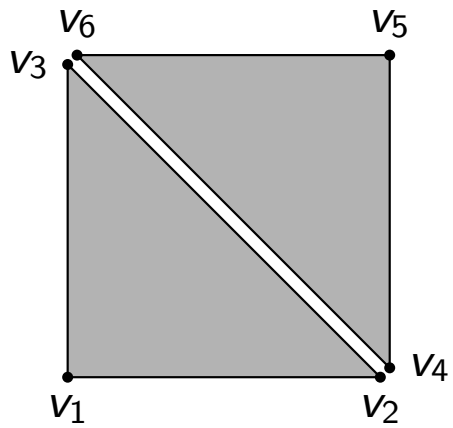
- A OpenGL tem como primitivas básicas pontos, linhas e triângulos.
- Modelos 3D mais complexos devem ser formados por conjuntos de triângulos.
- Uma malha de triângulos.
- Projetamos os objetos como esses conjuntos de triângulos e carregamos cada triângulo nos buffers do OpenGL.

Objetos complexos

- Um simples retângulo deve, por exemplo, ser decomposto em dois triângulos.
- No código `rectangles.cpp` definimos um retângulo com os seguintes triângulos.

Exemplo

```
float vertices[] = {
    // First triangle
    -0.5f, -0.5f, 0.0f, //v1
     0.5f, -0.5f, 0.0f, //v2
    -0.5f,  0.5f, 0.0f, //v3
    // Second triangle
     0.5f, -0.5f, 0.0f, //v4
     0.5f,  0.5f, 0.0f, //v5
    -0.5f,  0.5f, 0.0f  //v6
};
```



- Espaço entre triângulos é ilustrativo: $v_2 = v_4$ e $v_3 = v_6$.

Objetos complexos

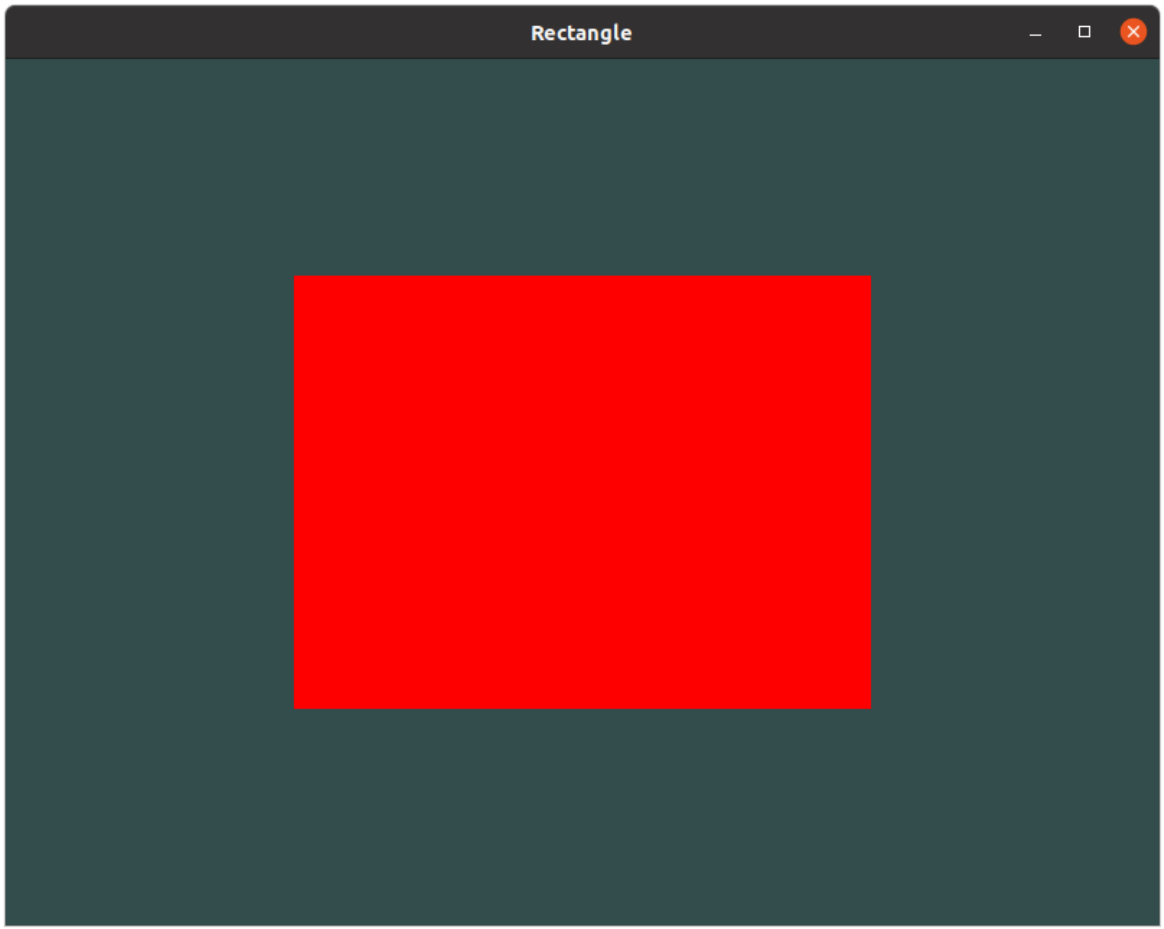
- Ao desenhar informamos ao OpenGL para:
  - usar a primitiva `GL_TRIANGLES`;
  - usar 6 vértices.

Exemplo

```
void display()
{
    ...
    glDrawArrays(GL_TRIANGLES, 0, 6);
    ...
}
```

Objetos complexos

- O retângulo.



Objetos complexos

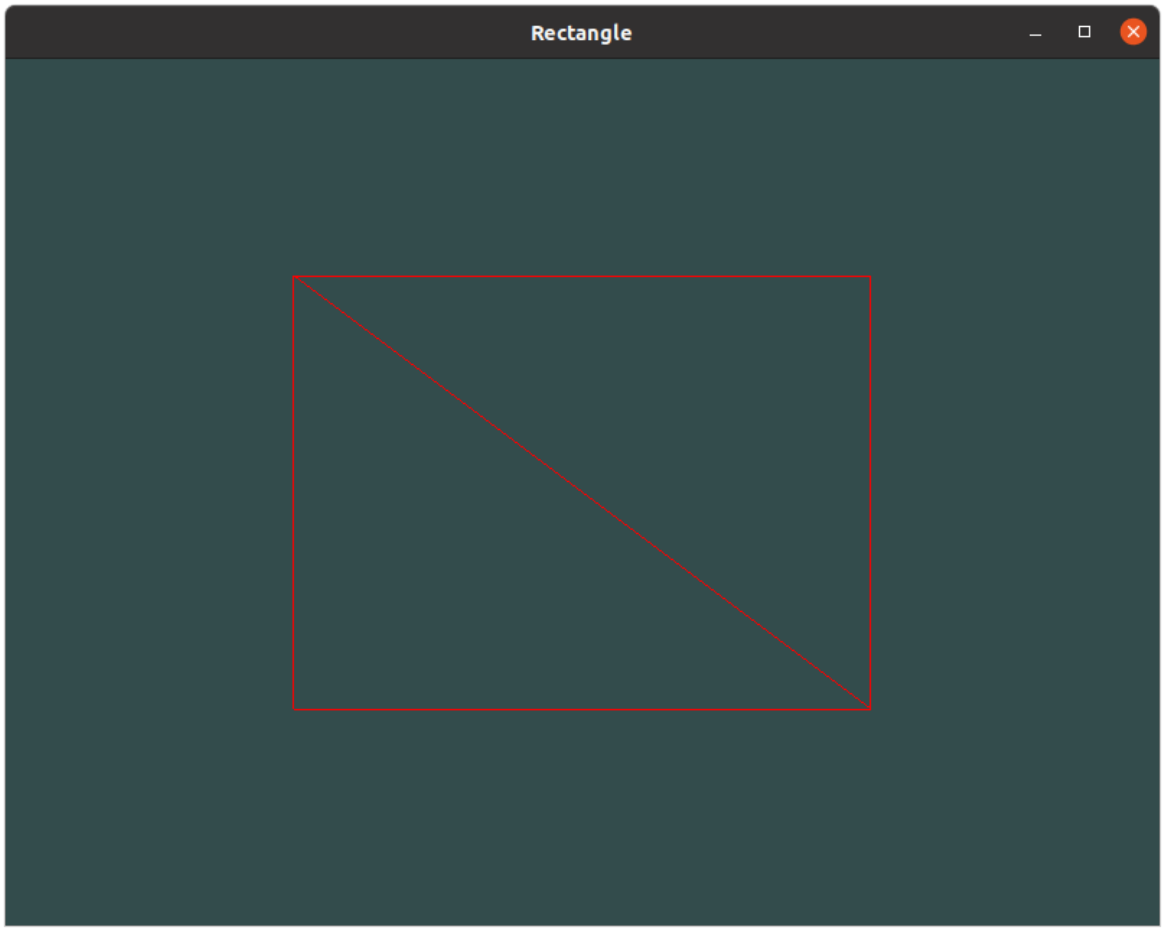
- Objetos também podem ser desenhados sem preenchimento (wireframe).
- No programa selecionamos o modo usando 1 e 2 no teclado.

Exemplo

```
void keyboard(unsigned char key, int x, int y)
        ...
        case '1':
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
                break;
        case '2':
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
                break;
        ...
```

Objetos complexos

- Wireframe: triângulos que compõem o retângulo.



Cores

- Podemos definir cores como atributo de cada vértice, como no código `rectangle2.cpp`.
- Cada vértice formado por dois atributos:
  - tripla para (x, y, z);
  - tripla para (R, G, B).

Exemplo

```
float vertices[] = {
    // First triangle
    // coordinate       color
    -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, //v1
     0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, //v2
    -0.5f,  0.5f, 0.0f, 1.0f, 0.0f, 0.0f, //v3
    // Second triangle
    // coordinate       color
     0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, //v4
     0.5f,  0.5f, 0.0f, 0.0f, 1.0f, 0.0f, //v5
    -0.5f,  0.5f, 0.0f, 1.0f, 0.0f, 0.0f  //v6
    };
```

Cores

- Incrementamos os shaders para manipular o atributo de cor.

Vertex shader

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 color;

out vec3 vColor;

void main()
{
    gl_Position = vec4(position, 1.0);
    vColor = color;
}
```

Cores

- Para cada vértice o vertex shader recebe:
  - na localização 0 um vetor `vec3` com (x, y, z),
  - na localização 1 um vetor `vec3` com (R, G, B).

Vertex shader

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 color;

out vec3 vColor;

void main()
{
    gl_Position = vec4(position, 1.0);
    vColor = color;
}
```

Cores

- Definimos uma variável de saída (out) `vColor` para enviar a cor para as etapas seguintes do pipeline.

Vertex shader

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 color;

out vec3 vColor;

void main()
{
    gl_Position = vec4(position, 1.0);
    vColor = color;
}
```

Cores

- O fragment shader recebe a cor e copia ela para a saída em um vetor `vec4` (R, G, B, A).
- $A = 1.0$ é a componente de opacidade.

Fragment shader

```
#version 330 core

in vec3 vColor;
out vec4 FragColor;

void main()
{
    FragColor = vec4(vColor, 1.0f);
}
```

Cores

- Copiamos os atributos intercalados para o buffer.
- Precisamos configurar os atributos para informar ao vertex shader:
  - a localização do atributo (location usado no shader) (parâmetro 1);
  - deslocamento entre atributos no buffer (parâmetro 5);
  - início dos atributos no buffer (parâmetro 6).

Cores

- Atributo coordenada:
  - location: 0,
  - deslocamento: próxima coordenada após 6 posições, 3 para a coordenada + 3 para a cor;
  - início: primeira posição do buffer.

Exemplo

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6*sizeof(float),
                      (void*)0);
glEnableVertexAttribArray(0);
```

Cores

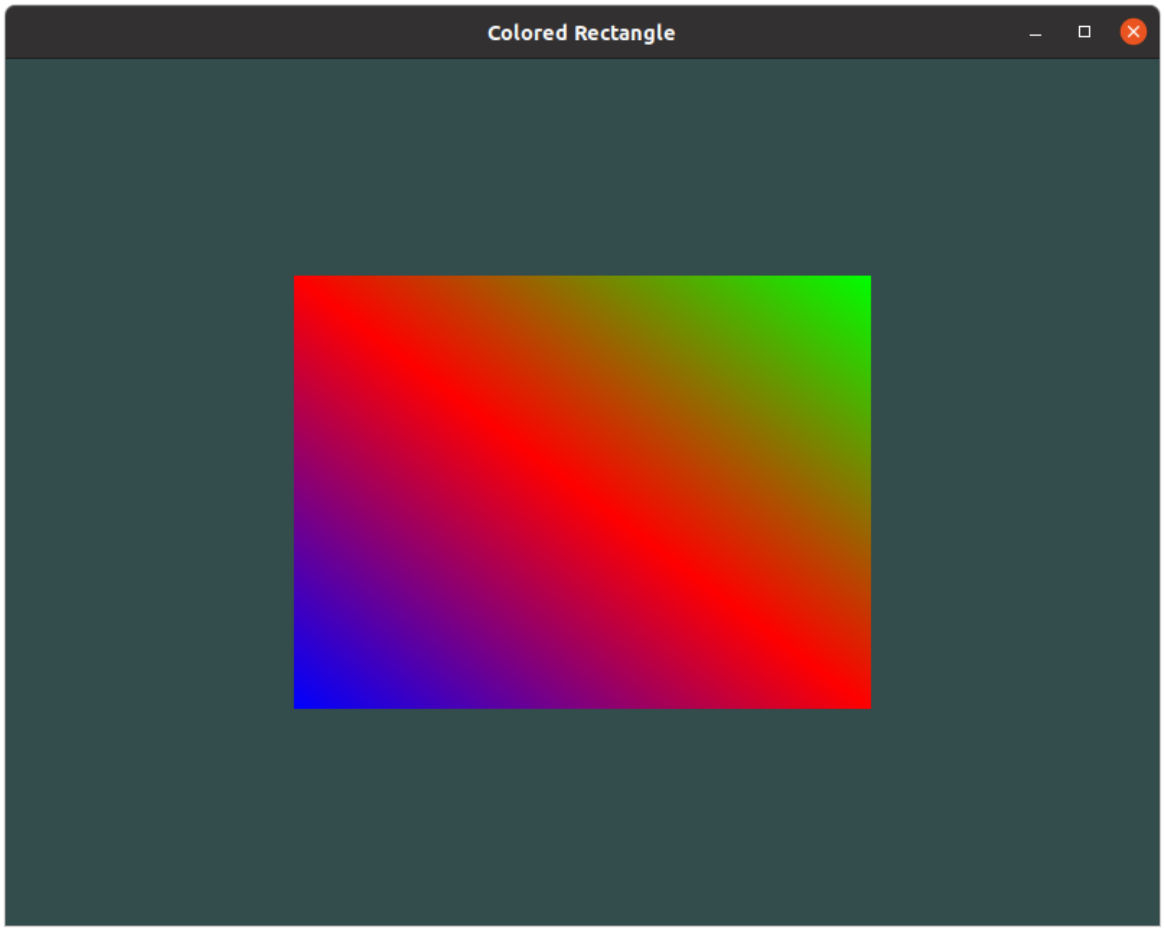
- Atributo cor:
  - location: 1,
  - deslocamento: próxima cor após 6 posições, 3 a cor + 3 para a coordenada;
  - início: após as 3 primeiras posições do buffer (que contêm a primeira coordenada).

Exemplo

```
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6*sizeof(float),
                      (void*)(3*sizeof(float)));
glEnableVertexAttribArray(1);
```

Objetos complexos

- Retângulo.



Objetos complexos

- Wireframe.