Introdução ao OpenGL

Prof. Mauro Miazaki

UNICENTRO
PARANÁ

Guarapuava, 2025

O que é o OpenGL?

- Uma API (*Application Programming Interface*) para geração de gráficos:
  - 2D e 3D;
  - Primitivas vetoriais e matriciais (imagens);
  - Capaz de gerar imagens de alta qualidade;
  - Implementado de forma a tirar vantagem da aceleração gráfica (se disponível);
  - Independente de plataforma;
  - Independente de sistema de janelas.

O que é o OpenGL?

- A API tem rotinas para:
  - Desenhar primitivas geométricas e imagens;
  - Alterar variáveis de estado;
    - Ex.: cor, material, fontes de iluminação, etc.
  - Consultar variáveis de estado.

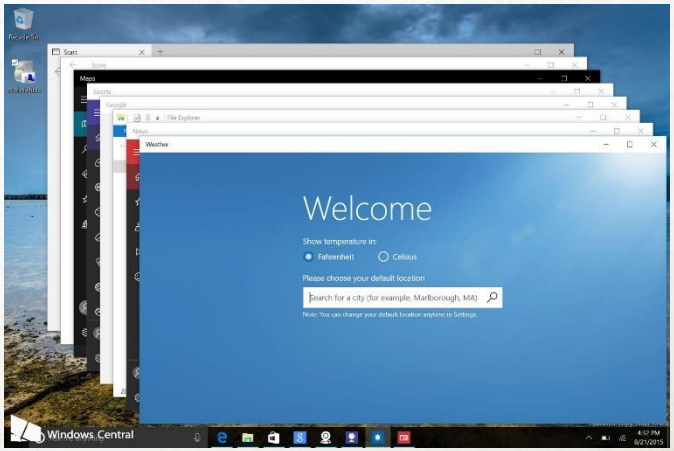
O que é o OpenGL?

- OpenGL é um padrão em evolução.
  - Mecanismo padronizado de extensões;
  - Novas versões são estabelecidas por Khronos Group.
    - Um consórcio sem fins lucrativos focado em criar padrões abertos;
    - Possui cerca de 120 membros, como: AMD, Apple, Google, Intel, Nvidia, Samsung, Sony, IBM, Microsoft, LG, etc.

KHRONOS™ GROUP

Sistema de Janelas

- Principal meio de interação homem/máquina em ambientes de computação modernos.
- Tela é dividida em janelas (eventualmente superpostas).
- Janelas são controladas por aplicações que têm a incumbência de mantê-las sempre atualizadas.
- Interação do usuário e do próprio sistema de janelas são comunicados à aplicação através de *eventos*, ex.:
  - Mouse foi apertado;
  - Janela foi redimensionada.



Sistema de Janelas

- Eventos são tratados por rotinas *callback* da aplicação. Ex.:
  - Redesenhar o conteúdo da janela;
  - Mover um objeto de um lado para outro da janela.
- *Callback* é uma função passada como argumento para outra função e chamada posteriormente, geralmente quando um evento específico ocorre.
- *Callbacks* são amplamente usados em programação assíncrona, manipulação de eventos e interfaces gráficas.

Sistema de Janelas

- Cada Sistema de Janelas possui uma API distinta.
  - MS Windows: Windows Manager;
  - macOS: WindowServer;
  - Android: Window Manager;
  - Linux: X11, Wayland.
    - Compositores Wayland:
      - Weston, Mutter (Gnome), Kwin (KDE), Xfwm (Xfce), entre outros.
- Portabilidade: camada de interface para diversos Sistemas de Janela, mas com API única.
  - Ex.: FreeGLUT, PySDL2, Pygame, entre outros.

APIs Relacionadas

- GLU (*OpenGL Utility Library*):
  - Parte do padrão OpenGL;
  - Curvas, trianguladores, quádricas, etc.
- GLUT (*OpenGL Utility Toolkit*):
  - API portátil de acesso aos sistemas de janelas;
  - Encapsula e esconde as camadas proprietárias;
  - Não é parte oficial do OpenGL;
  - Não é mais atualizado.
- FreeGLUT:
  - Alternativa ao GLUT;
  - Muito parecido com GLUT e bastante estável;
  - Frequentemente atualizado;
  - O PyOpenGL já inclui esta API.

Instalação no Python

- pip install PyOpenGL PyOpenGL_accelerate

Anatomia de um programa OpenGL/FreeGLUT

```
from OpenGL.GL import *
from OpenGL.GLUT import *

def display():
    # ...

# Outras rotinas callback

def main():
    glutInit()
    glutInitDisplayMode( modo )
    glutCreateWindow( nome da janela )
    glutDisplayFunc( display )
    glutReshapeFunc( reshapeCallback )
    # Registro de outras rotinas callback
    glutMainLoop()

if __name__ == "__main__":
    main()
```

- Importação de módulos
- Rotinas *Callback*
- Inicialização do GLUT
- Inicialização da janela
- Registro de *callbacks*
- Laço principal

Exemplo (Parte 1)

```
from OpenGL.GL import *
from OpenGL.GLUT import *

def display():
    # Limpa todos os pixels da tela
    glClear(GL_COLOR_BUFFER_BIT)

    # Desenha um polígono branco (retângulo)
    glBegin(GL_POLYGON)
    glVertex3f(0.25, 0.25, 0.00)
    glVertex3f(0.75, 0.25, 0.00)
    glVertex3f(0.75, 0.75, 0.00)
    glVertex3f(0.25, 0.75, 0.00)
    glEnd()

    # Troca buffers - double-buffering
    glutSwapBuffers()
```

Exemplo (Parte 2)

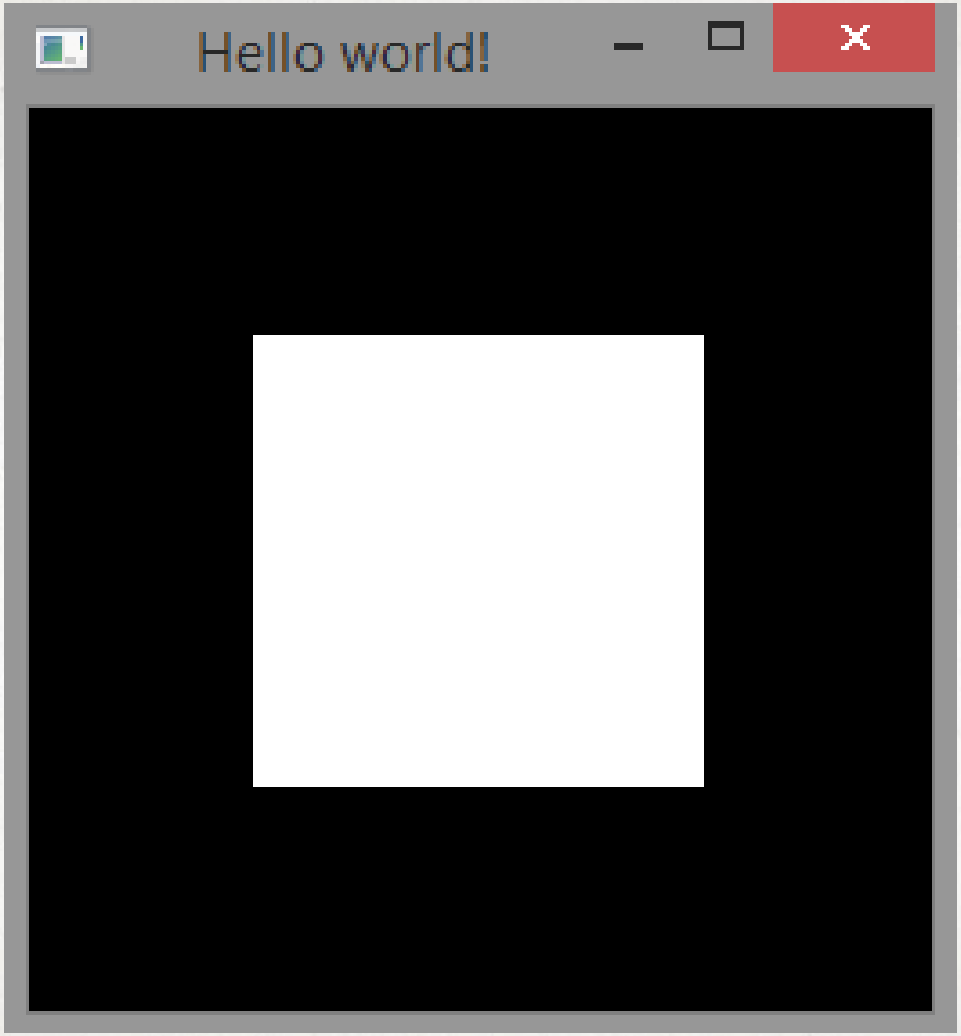
```
def init():
    # Define a cor de fundo preta
    glClearColor(0.0, 0.0, 0.0, 1.0)

    # Inicializa as matrizes do OpenGL
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0)
    glMatrixMode(GL_MODELVIEW)

def main():
    glutInit()
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB)
    glutInitWindowSize(250, 250)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"Hello world!")
    init()
    glutDisplayFunc(display)
    glutMainLoop()
```

```
if __name__ == "__main__":
    main()
```

Resultado do Exemplo



Buffer simples x *Buffer* duplo

- *Buffer* simples: todos os comandos de desenho são executados diretamente no buffer da tela (desenhados imediatamente).
  - Habilitado com:
    - glutInitDisplayMode(GLUT_SINGLE | ...)
  - Na função *display*, deve ser chamado no fim:
    - glFlush()

Buffer simples x *Buffer* duplo

- *Buffer* duplo: habilita o uso de dois *buffers*, o que ajuda a reduzir o efeito *flickering* (piscadas) na tela.
  - Todos os comandos de desenho são executados em um *buffer* não exibido na tela, que é enviado à tela quando todo o desenho a ser exibido estiver pronto.
  - Habilita com:
    - glutInitDisplayMode(GLUT_DOUBLE | ...)
  - Na função *display*, deve ser chamado no fim:
    - glutSwapBuffers()

FreeGLUT – Registrando *Callbacks*

- *Callbacks* são rotinas que serão chamadas para tratar eventos.
- Para uma rotina *callback* ser efetivamente chamada ela precisa ser registrada através da função:

  `glutXxxFunc(`*`callback`*`)`

  - Onde `Xxx` designa uma classe de eventos e *`callback`* é o nome da rotina.
- Por exemplo, para registrar uma *callback* de desenho chamada `desenho`, usa-se:

  `glutDisplayFunc(desenho)`

FreeGLUT – *Callback* de desenho

- É a rotina chamada automaticamente sempre que a janela ou parte dela precisa ser redesenhada.
  - Ex.: uma janela obscurecida por outra que foi fechada.
- Todo programa GLUT precisa ter uma!
- Exemplo:

```
def display ( ):
  glClear( GL_COLOR_BUFFER_BIT ) # Limpa a tela
  glBegin( GL_POLYGON )          # Desenha um retângulo
  glVertex3f( 0.25, 0.25, 0.0 )
  glVertex3f( 0.75, 0.25, 0.0 )
  glVertex3f( 0.75, 0.75, 0.0 )
  glVertex3f( 0.25, 0.75, 0.0 )
  glEnd()
  glutSwapBuffers() # Uso de double-buffering
```

FreeGLUT - Outras *Callbacks*

- Outras *callbacks* comumente usadas:
  - Eventos de teclado:

```
def keyboard(key, x, y)
```

  - Eventos de mouse:
    - Clique em um botão do mouse:

```
def mouse(button, state, x, y)
```

    - Movimento do mouse (**sem** um botão pressionado):

```
def passiveMotion(x, y)
```

    - Movimento do mouse **com** um botão pressionado:

```
def motion(x, y)
```

  - Chamada continuamente quando nenhum outro evento ocorre:

```
def idle()
```

- Várias outras.

FreeGLUT – Inicialização do Programa

- Inicialização do FreeGLUT:

```
glutInit()
glutInit(argc, argv)
```

  - Estabelece contato com sistema de janelas
  - Opções argv:
    - https://www.opengl.org/resources/libraries/glut/spec3/node10.html

FreeGLUT – Inicialização do Programa

- Inicialização da(s) janela(s):

`glutInitDisplayMode(`*`modo`*`)`

  - Estabelece os tipos de recursos necessários para as janelas que serão criadas. *Modo* é um "ou" bit-a-bit de constantes:
    - GLUT_RGB cores dos pixels serão expressos em RGB;
    - GLUT_DOUBLE *bufferização* dupla (ao invés de simples);
    - GLUT_DEPTH *buffer* de profundidade (*z-buffer*);
    - GLUT_ACCUM *buffer* de acumulação;
    - GLUT_ALPHA *buffer* de cores terá componente alfa;
    - Outros: https://www.opengl.org/resources/libraries/glut/spec3/node12.html

`glutInitWindowPosition(x, y)`

  - Estabelece a posição inicial do canto superior esquerdo da janela a ser criada.

`glutInitWindowSize(width, height)`

  - Estabelece o tamanho (em pixels) da janela a ser criada.

FreeGLUT – Inicialização do Programa

- Criação da(s) janela(s):

  `glutCreateWindow(`*`nome`*`)`

  - Cria uma nova janela;
  - *Nome* é usado para rotular a janela;
  - Retorna um número inteiro, usado pelo GLUT para identificar a janela.
- Outras inicializações:
  - Após a criação da janela, pode-se configurar variáveis de estado do OpenGL que não mudarão no decorrer do programa. Por exemplo:
    - Cor do fundo;
    - Tipo de sombreamento desejado.

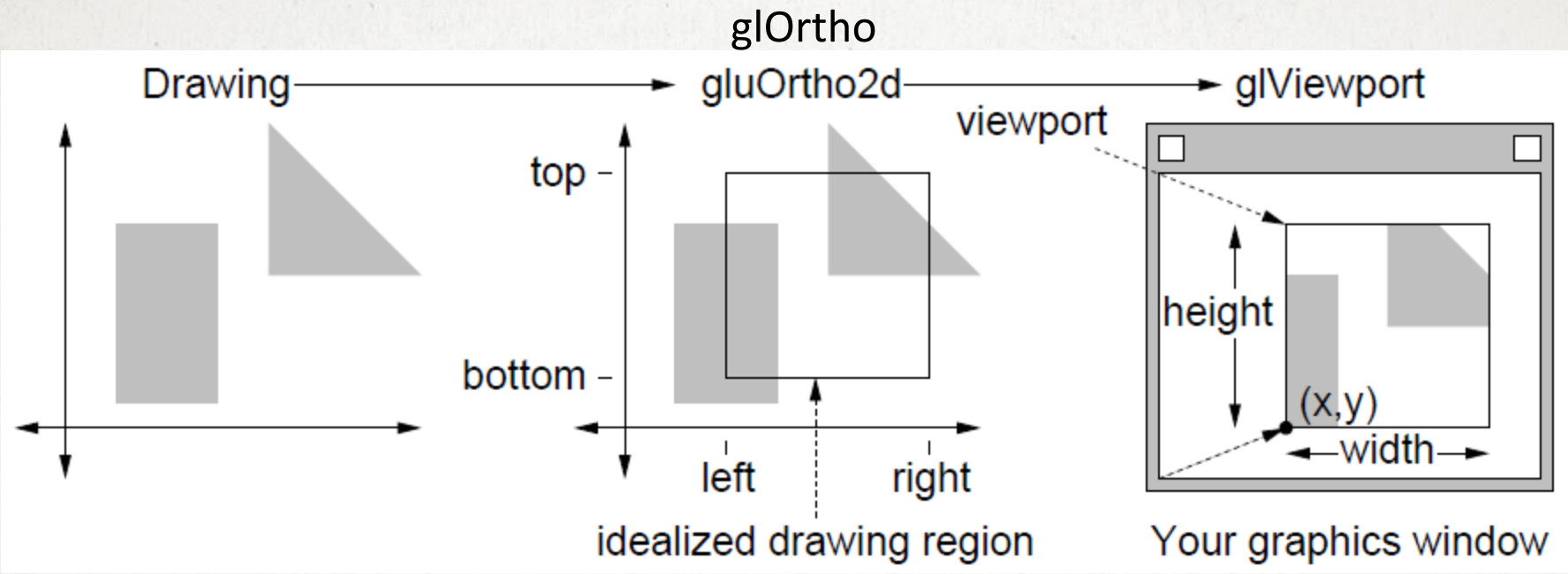
FreeGLUT – Laço Principal do Programa

- Depois de registradas as *callbacks*, o controle é entregue ao sistema de janelas:

```
glutMainDisplayLoop()
```

- Esta rotina é o “despachante” de eventos.
- Nunca retorna, a não ser que a janela seja fechada.

OpenGL – Window e Viewport



Janela e Projeção Ortogonal

`glOrtho(x1, x2, y1, y2, z1, z2)`

- Esta função de projeção define um retângulo no sistema de coordenadas do mundo, no qual serão especificadas as coordenadas dos objetos.
- Ex.:

```
glMatrixMode(GL_PROJECTION)      # set projection matrix
glLoadIdentity()                 # initialize to identity
glOrtho(0.0, 1.0, 0.0, 1.0 , -1.0, 1.0) # map unit cube to viewport
glMatrixMode(GL_MODELVIEW)       # set modelview matrix
```

Viewport

```
glViewport(x, y, width, height)
```

- Viewport é um retângulo dentro da janela gráfica na tela, na qual os elementos gráficos aparecerão.

FreeGLUT – *Callback* de redimensionamento

- *Callback*: glutReshapeFunc(reshape)
- A função tem a forma:

```
def reshape (width, height): ...
```

  - width/height são a nova largura/altura da janela (em pixels) após o redimensionamento.
- Chamada sempre que a janela é redimensionada, isto é, teve seu tamanho alterado.
- Se uma rotina de redimensionamento não for especificada, o GLUT usa uma rotina de redimensionamento "*default*" que simplesmente ajusta o *viewport* para usar toda a área da janela.

FreeGLUT – *Callback* de redimensionamento

- Exemplos:

```
def reshape(width, height):
    glViewport(0, 0, width, height)
```

```
def reshape(width, height):
    size = width if width < height else height
    glViewport(0, 0, size, size)
```

OpenGL – Primitivas de desenho

```
glBegin( PRIMITIVA )
```

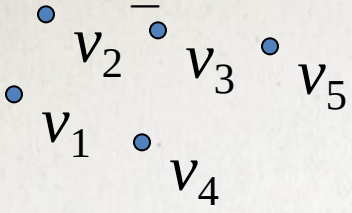
especificação de vértices, cores, coordenadas de textura, propriedades de material.

```
glEnd()
```

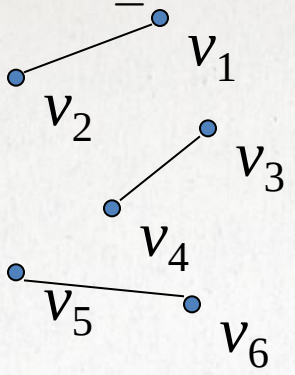
- Entre `glBegin()` e `glEnd()` apenas alguns comandos podem ser usados. Ex.:
  - `glMaterial`
  - `glNormal`
  - `glTexCoord`
- Uma vez emitido um vértice (`glVertex`), este é desenhado com as propriedades (cor, material, normal, coordenadas de textura, etc.) registradas nas variáveis de estado correspondentes.
- Antes de emitir um vértice, assegurar-se que cor, material, normal, etc. têm o valor certo.

OpenGL – Primitivas de desenho

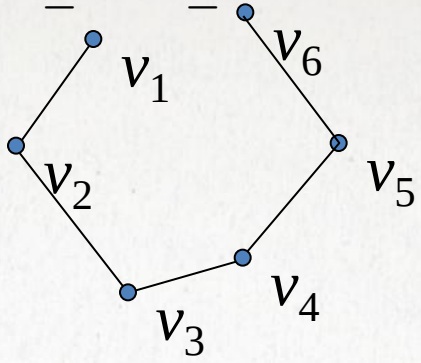
GL_POINTS



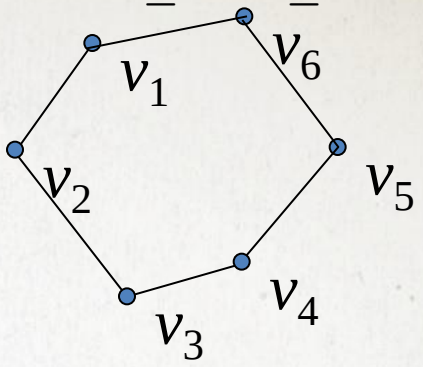
GL_LINES



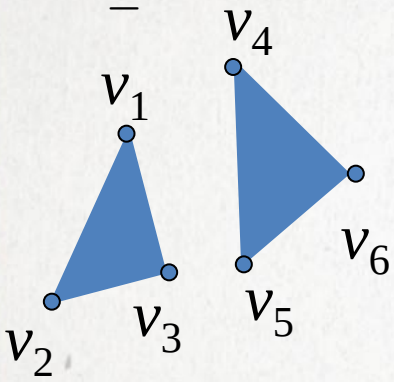
GL_LINE_STRIP



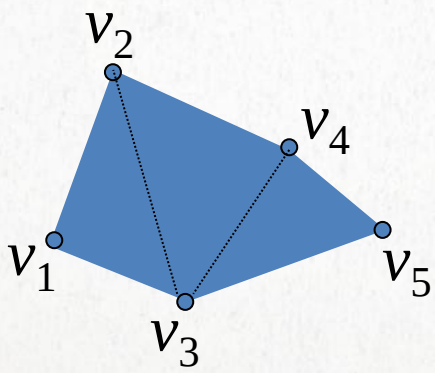
GL_LINE_LOOP



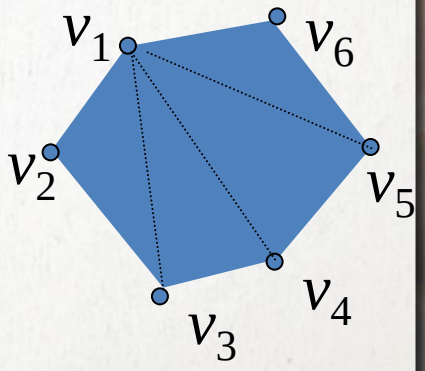
GL_TRIANGLES



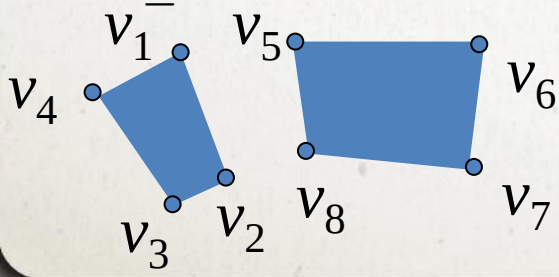
GL_TRIANGLE_STRIP



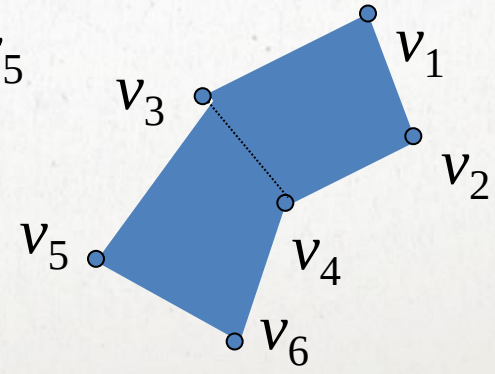
GL_POLYGON
GL_TRIANGLE_FAN



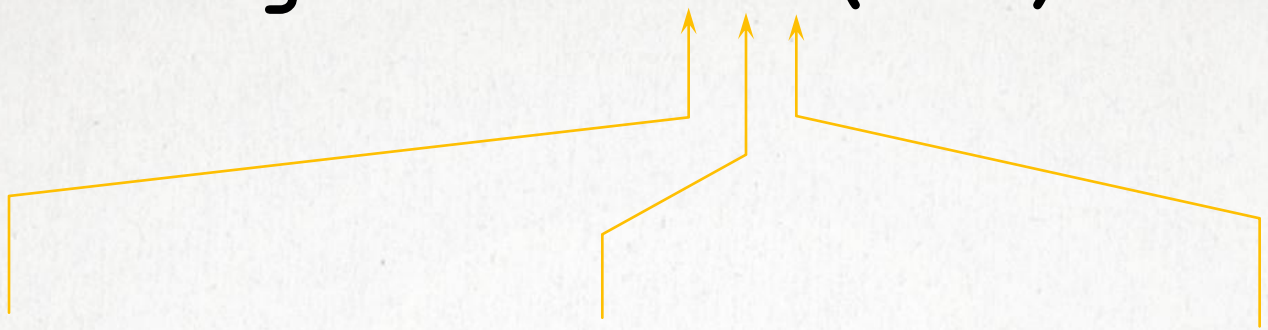
GL_QUADS



GL_QUAD_STRIP



OpenGL – Convenções de Nome



Número de componentes

```
2 - (x,y)
3 - (x,y,z)
4 - (x,y,z,w)
```

Tipo de dado

```
b  - byte
ub - unsigned byte
s  - short
us - unsigned short
i  - int
ui - unsigned int
f  - float
d  - double
```

vetor

omita o "v" quando coords dadas uma a uma

```
glVertex2f( x, y )
```

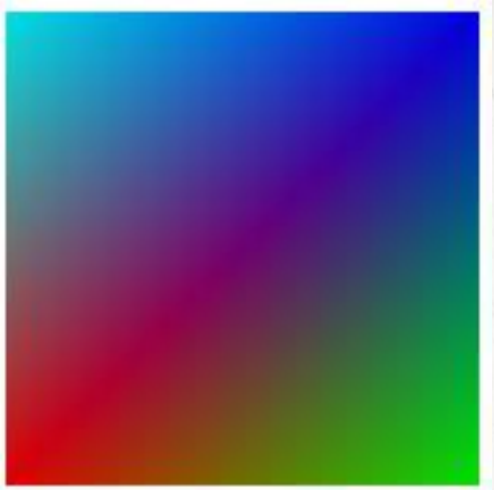
OpenGL – Controlando as cores

- Pode-se especificar diretamente as cores (*default* – modo habilitado por padrão).
  - `glColor**()`
- Pode-se também computar as cores a partir de um modelo de iluminação.
  - Será visto futuramente.

Cores

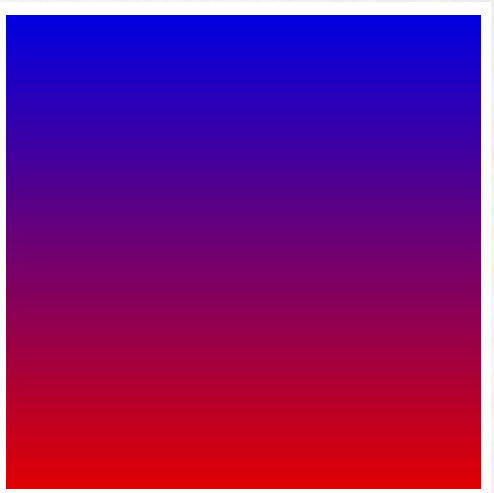
- A atribuição de cores pode ser feita por vértice:

```
glBegin(GL_POLYGON)
glColor3f(1,0,0)
glVertex3f(0.25, 0.25, 0.0)
glColor3f(0,1,0)
glVertex3f(0.75, 0.25, 0.0)
glColor3f(0,0,1)
glVertex3f(0.75, 0.75, 0.0)
glColor3f(0,1,1)
glVertex3f(0.25, 0.75, 0.0)
glEnd()
```



Cores

```
glBegin(GL_POLYGON)
glColor3f(1,0,0)
glVertex3f(0.25, 0.25, 0.0)
//glColor3f(0,1,0)
glVertex3f(0.75, 0.25, 0.0)
glColor3f(0,0,1)
glVertex3f(0.75, 0.75, 0.0)
//glColor3f(0,1,1)
glVertex3f(0.25, 0.75, 0.0)
glEnd()
```



Cores

glBegin(GL_POLYGON)

glColor3f(0,0,0)

glVertex3f(0.25, 0.25, 0.0)

glVertex3f(0.75, 0.25, 0.0)

glColor3f(1,0,0)

glVertex3f(0.75, 0.75, 0.0)

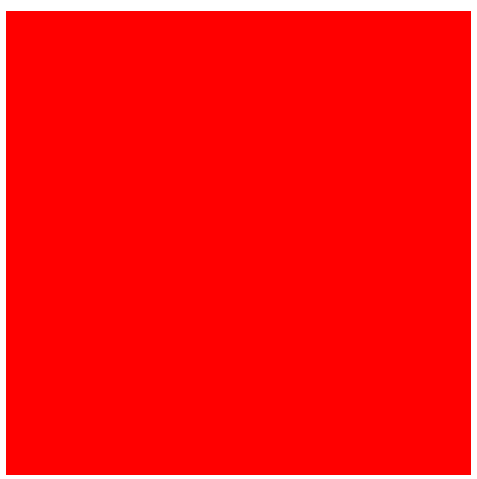
glVertex3f(0.25, 0.75, 0.0)

glEnd()

Cores

 # Desabilita shading
glShadeModel(GL_FLAT)

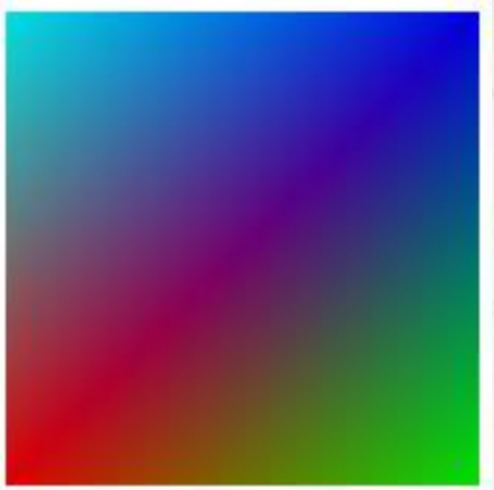
glBegin(GL_POLYGON)
glColor3f(1,0,0)
glVertex3f(0.25, 0.25, 0.0)
glColor3f(0,1,0)
glVertex3f(0.75, 0.25, 0.0)
glColor3f(0,0,1)
glVertex3f(0.75, 0.75, 0.0)
glColor3f(0,1,1)
glVertex3f(0.25, 0.75, 0.0)
glEnd()



Cores

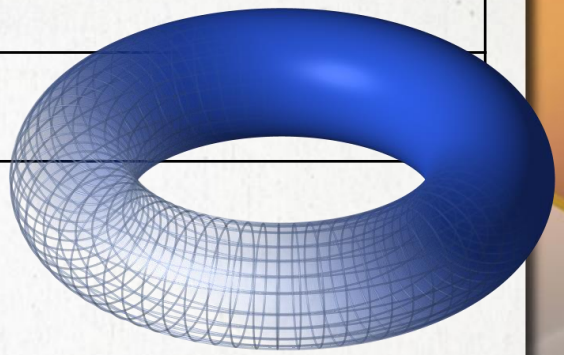
 # Habilita shading (*default*)
glShadeModel(GL_SMOOTH)

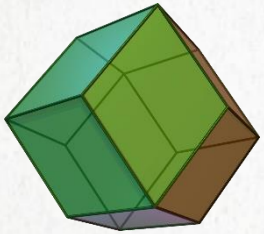
glBegin(GL_POLYGON)
glColor3f(1,0,0)
glVertex3f(0.25, 0.25, 0.0)
glColor3f(0,1,0)
glVertex3f(0.75, 0.25, 0.0)
glColor3f(0,0,1)
glVertex3f(0.75, 0.75, 0.0)
glColor3f(0,1,1)
glVertex3f(0.25, 0.75, 0.0)
glEnd()

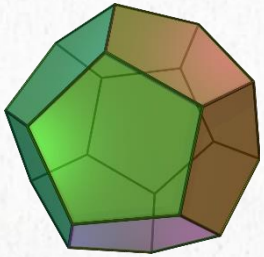


GLUT – Objetos 3D pré-definidos

Cone	Octahedron (8 faces)	Teapot
Cube	Tetrahedron (10 faces)	Torus
Sphere	Dodecahedron (12 faces)	
	Icosahedron (20 faces)	





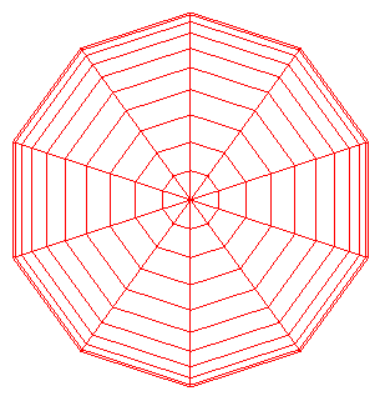


Estes modelos são desenhados com o centro na origem do sistema de coordenadas do mundo.

GLUT – Objetos 3D pré-definidos

- **glutWireCube**(*size*)
- **glutSolidCube**(*size*)

- **glutWireSphere**(*radius*, *slices, stacks*)
- **glutSolidSphere**(*radius*, *slices, stacks*)



glutWireSphere(1,10,20)

GLUT – Objetos 3D pré-definidos

- **glutSolidCone**(base, height, slices, stacks)
- **glutSolidTorus**(innerRadius, outerRadius, nsides, rings)
- **glutSolidTeapot**(size)
- **glutSolidOctahedron**()
- **glutSolidTetrahedron**()
- **glutSolidDodecahedron**()
- **glutSolidIcosahedron**()

Teclado

- Chamado se uma tecla é pressionada no teclado:

```
def keyboard(key, x, y):
    print(key, x, y)
    match key: # key é a tecla pressionada
        case b'q':
            glutDestroyWindow(glutGetWindow())
```

Callback: glutKeyboardFunc(keyboard)

Teclado – Teclas Especiais

```
def keySpecial(key, x, y):
    if key == GLUT_KEY_LEFT:
        print("Seta para a esquerda pressionada!")
    elif key == GLUT_KEY_RIGHT:
        print("Seta para a direita pressionada!")
```

- *Callback*:

`glutSpecialFunc(keySpecial)`

- Mais teclas:

http://www.opengl.org/resources/libraries/glut/spec3/node54.html

Teclas <SHIFT> <ALT> <CTRL>

```
if glutGetModifiers() == GLUT_ACTIVE_SHIFT:
    print("SHIFT pressionado")

if glutGetModifiers() == GLUT_ACTIVE_ALT:
    print("ALT pressionado")

if glutGetModifiers() == GLUT_ACTIVE_CTRL:
    print("CTRL pressionado")
```

Mouse

- Chamado quando o botão do mouse é apertado:

```
def mouse(b, s, x, y):
    match b: # b indica qual botão
        case GLUT_LEFT_BUTTON:
            if s == GLUT_DOWN:
                print("Botao pressionado ", x, y)
            elif s == GLUT_UP:
                print("Botao liberado ", x, y)
```

Callback: glutMouseFunc(mouse)

Nome do parâmetro GLUT	Significado
GLUT_LEFT_BUTTON	Botão esquerdo do mouse
GLUT_MIDDLE_BUTTON	Botão do meio do mouse
GLUT_RIGHT_BUTTON	Botão direito do mouse
GLUT_DOWN	Botão do mouse pressionado
GLUT_UP	Botão do mouse liberado

Movimento do Mouse

- *Callback* chamado quando o mouse se move dentro da janela, enquanto um ou mais botões do mouse estão pressionados.
  - `glutMotionFunc(motion)`
    - `motion(x, y)`
- *Callback* chamado quando o mouse se move dentro da janela sem *nenhum* botão pressionado.
  - `glutPassiveMotionFunc(passiveMotion)`
    - `passiveMotion(x, y)`

Referências

- Manual GLUT: https://www.opengl.org/resources/libraries/glut/spec3/spec3.html
- Manual FreeGLUT: http://freeglut.sourceforge.net/docs/api.php
- OpenGL Programming Guide – Version 1.1 (Redbook): http://glprogramming.com/red/