PROGRAMAÇÃO OPENGL

EDITORA CAMPUS

PROGRAMAÇÃO OPENGL

- **Componentes Oficiais**
  GL
  GLU

- **Não oficiais GLUT**

PROGRAMAÇÃO OPENGL

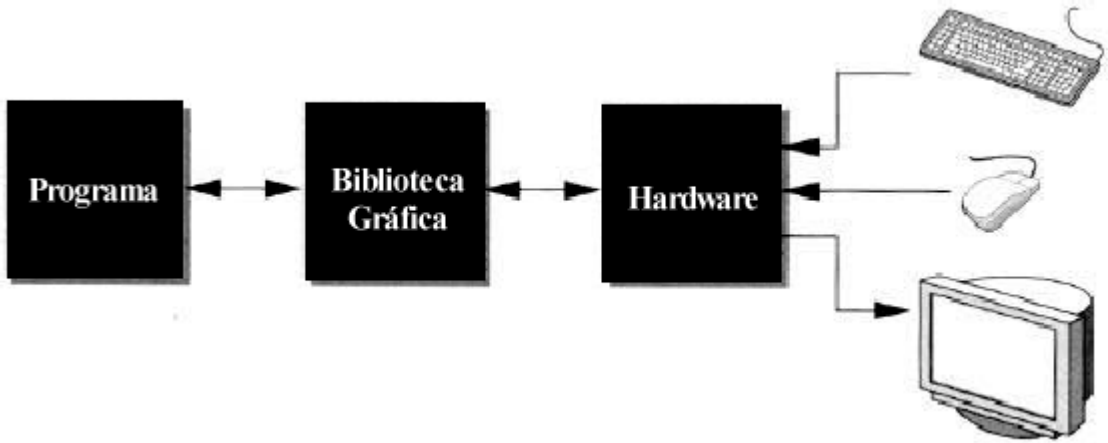
- **Disponível em várias linguagens**

Delph
Visual Basic
Java
C/C++

- **GLUT**

http://www.opengl.org/developers/documentation/glut/index.html

API GRÁFICA
(API - application programmer's interface)



• Uma API atua como interface entre o código da aplicação e o hardware.

• As APIs mais populares (OpenGL, DirectX, Java3D, PHIGS, GKS-3D) são baseadas em modelos de câmeras sintéticas.

• Possuem funções que especificam e simulam:
  • objetos, observadores, fontes de luz e propriedades dos materiais

OpenGL

- **Low-level graphic API**
  - "immediate mode" render para o frame buffer

- **Ordem da Programação:**
  - comandos ou display list
  - ajuste do estado(transformações, cores, etc.)
  - per vertex operations & primitive assembly
    - aplica as transformações, sombreamentos
    - cria triângulos/linhas/pontos
  - rasterização
    - de primitivas para pixel
  - operações per pixel
    - blending, texturas, etc.
  - frame buffer

OpenGL

- **Independente do sistema de Janela**
  - sem facilitações para eventos de janela ou entrada de usuários
  - necessita bibliotecas adicionais

- **Fácil programação em C/C++**

- **Disponível em quase todas as plataformas**
  - Windows
  - Unix
  - MacOS
  - Silicon
  - Linux
  - etc

Glut

- Ferramenta utilitário OpenGL
- Provê facilitações para eventos de janela e entrada de usuários baseadas em eventos
- Uso restrito para grandes aplicações
- Uso ideal para estudo de OpenGL

COMANDOS

- GL

glColor3f

Biblioteca gl — comando — número de argumentos — tipo dos argumentos

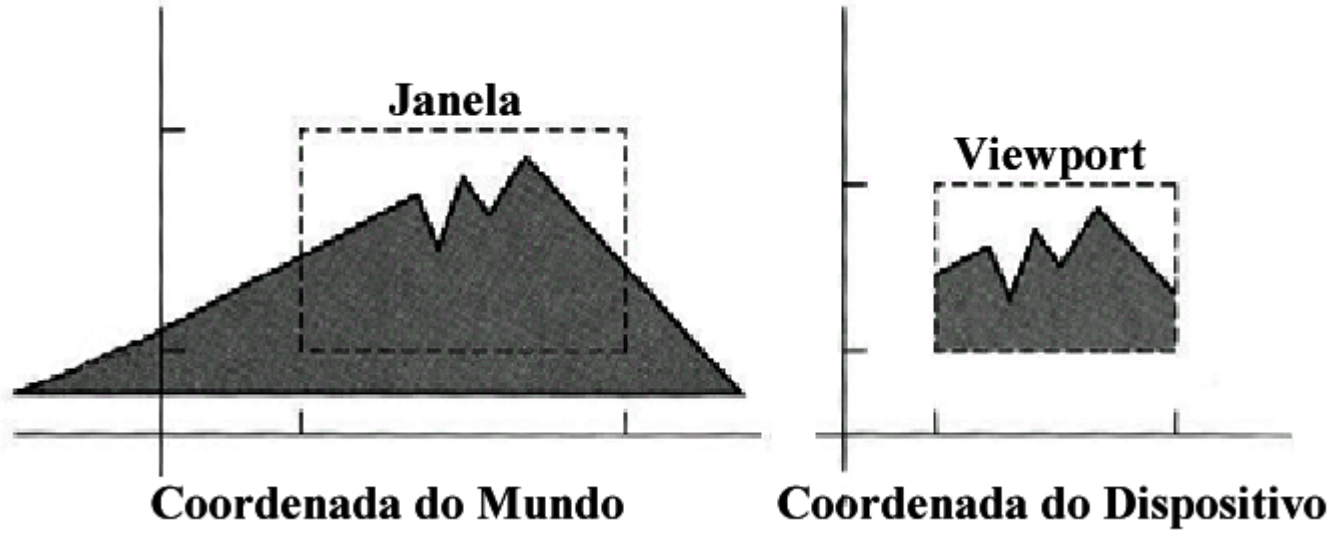
- GLU

Prefixo glu

- GLUT

Prefixo glut

Especificando um observador em 2D



- **A maioria do sistemas gráficos permitem especificar:**
  - a parte da figura para mostrar (Janela)
  - o lugar para mostrar a figura na tela (viewport)

- **A janela padrão do OpenGL é [-1,1]x[-1,1]**

- **A viewport padrão do OpenGL é full-window**

Especificando um observador em 2D

- **Define a viewport**

  glViewport(left, bottom, width, height)

- **Define o modo da matriz e inicializa:**

  glMatrixMode(GL_PROJECTION)

  glLoadIdentity()

- **Define a projeção ortográfica**

  gluOrtho2D(left, right, bottom, top)

Limpando o Buffer

- **A OpenGL possui diversos buffers que devem ser limpos antes do seu uso**
  - color buffer (para desenhar as imagens)
  - depth buffer (para determinar o que é ou não visível)
  - accumulation buffer (para motion blur, anti-alising, ...)
  - stencil buffer (para restringir a certas porções da tela)

- **Clear values**
  - glClearColor(r,g,b,a)
  - glClearDepth(depth)
  - glClearAccum(r,g,b,a)
  - glClearStencil(s)

- **Perform the clear**
  - glclear(mask)
  - mask
    - gl_color_buffer_bit
    - gl_depth_buffer_bit
    - gl_accum_buffer_bit
    - gl_stencil_buffer_bit

Especificando a Cor

- **Em OpenGL, a descrição da forma do objeto é independente de sua cor**

- **Especificando a cor**
  - glColor3f(r,g,b)

```
glColor3f( 0.0, 0.0, 0.0 ) # black
glColor3f( 1.0, 0.0, 0.0 ) # red
glColor3f( 0.0, 1.0, 0.0 ) # green
glColor3f( 1.0, 1.0, 0.0 ) # yellow
glColor3f( 0.0, 0.0, 1.0 ) # blue
glColor3f( 1.0, 0.0, 1.0 ) # magenta
glColor3f( 0.0, 1.0, 1.0 ) # cyan
glColor3f( 1.0, 1.0, 1.0 ) # white
```

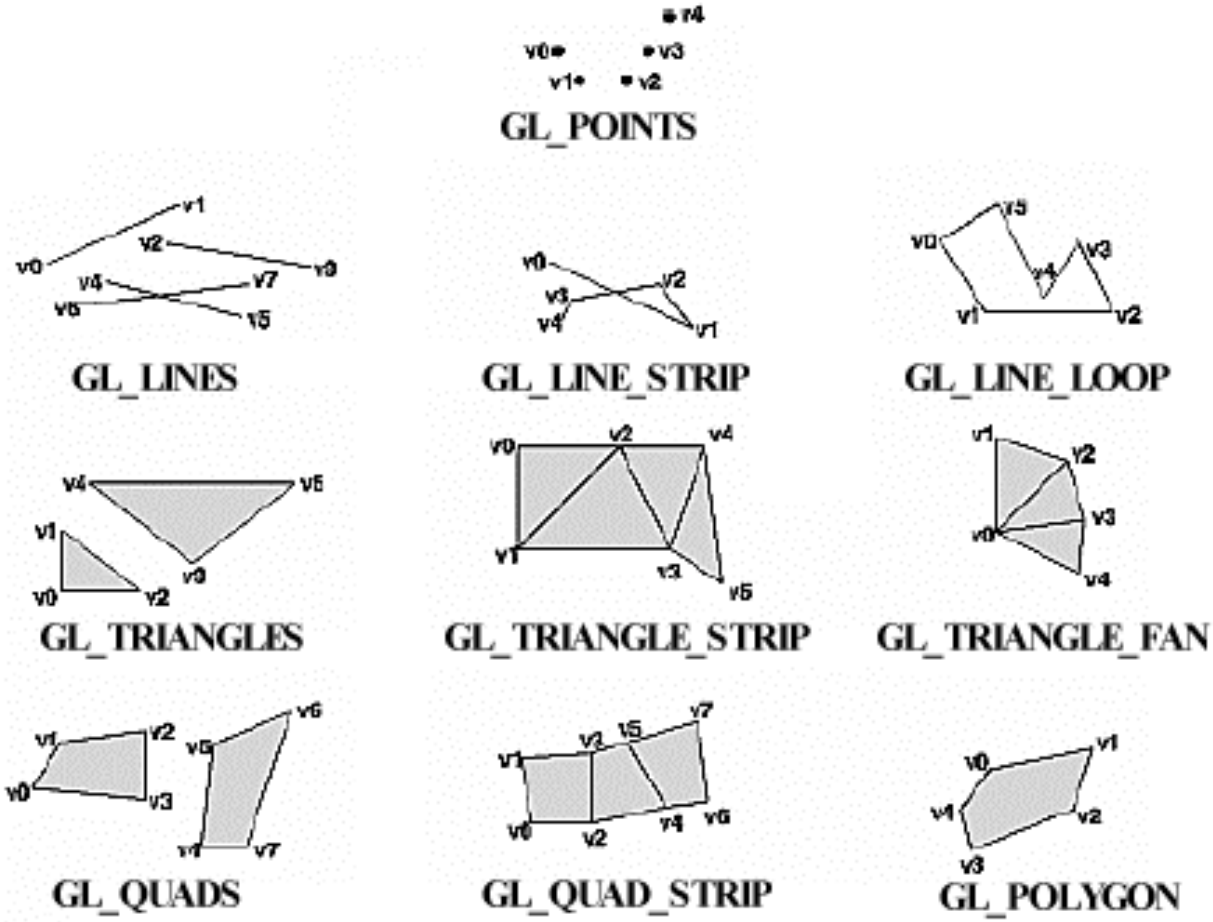
- **Exemplo**
  - desenhe um objeto A e B em vermelho
  - desenhe um objeto C em azul

```
set_current_color( red )
draw_object( A )
draw_object( B )
set_current_color( green )
set_current_color( blue )
draw_object( C )
```

Especificando as primitivas

- Ordenação dos vértices deve ser consistente
- Padrão: Sentido anti-horário
- É possível especificar a ordem
  - glFrontFace (ordem)
    - GL_CW: Sentido horário
    - GL_CCW: Sentido anti-horário

Primitivas OpenGL



Especificando as primitivas

- **Especifica os valores x,y e z dos pontos**

```
glBegin()
glVertex3f( x,y,z )
glEnd()
```

- **Desenhando um quadrado**

```
glBegin( GL_POLYGON )
glVertex3f(-.5, .5, .5)
glVertex3f( .5, .5, .5)
glVertex3f( .5,-.5, .5)
glVertex3f(-.5,-.5, .5)
glEnd()
```

- **Adicionando Cores**

```
glBegin(GL_POLYGON)
glColor3f (1, 0, 0)
glVertex3f(-.5, .5, .5)
glVertex3f( .5, .5, .5)
glColor3f (0, 1, 0)
glVertex3f( .5,-.5, .5)
glVertex3f(-.5,-.5, .5)
glEnd()
```

Propriedades de linhas e pontos

- **Tamanho (default = 1)**

  glPointSize(size)

  glLineWidth(width **)**

- **Ligar Suavização (padrão = OFF)**

  glEnable(GL_POINT_SMOOTH)

  glEnable(GL_LINE_SMOOTH)

- **Desligar Suavização**

  glDisable(GL_POINT_SMOOTH)

  glDisable(GL_LINE_SMOOTH)

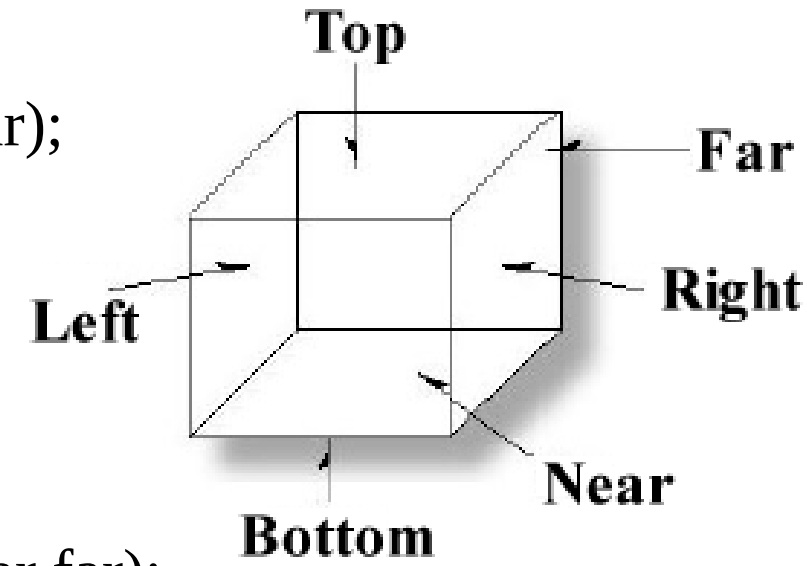
Modos de Renderização

- **Wireframe: glPolygonMode (GL_FRONT, GL_LINE);**
- **Sólido: glPolygonMode (GL_FRONT, GL_FILL);**
- **Pontos: glPolygonMode (GL_FRONT, GL_POINT);**
- **Backface Culling**
  - **glEnable (GL_CULL_FACE);**
- **Z-Buffer**
  - **glEnable (GL_DEPTH_TEST);**

Projeções

- **Projeção Ortogonal**

glOrtho(left,right,bottom,top,near,far);



- **Projeção Perspectiva**

glFrustrum(left,right,bottom,top,near,far);

Da biblioteca de utilitários:

gluPerspective(angle,aspect,near,far)

Posição da câmera

void gluLookAt (GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz);

onde eyex, eyey e eyez são as coordenadas da posição da câmera

centerx, centery e centerz são as coordenadas da posição do alvo

upx, upy e upz indica o lado de cima da cena 3D

Transformações Geométricas

- **Matrizes de modelagem e projeção são independentes**

  glMatrixMode (GL_PROJECTION);
  glMatrixMode (GL_MODELVIEW);

- **Translação**

  glTranslate(TYPE x, TYPE y, TYPE z)

- **Rotação**

  glRotate(TYPE angle, TYPE x, TYPE y, TYPE z)

- **Escala**

  glScale(TYPE x, TYPE y, TYPE z)

Objetos Sólidos

- **Utilizando biblioteca de utilitário Glut**

```
glutSolidTeapot(GLdouble size);  // Desenha uma Chaleira
glutSolidCube(GLdouble size);     // Desenha um Cubo
glutSolidSphere(GLdouble radius, GLint slices, GLint stacks); // Desenha uma Esfera
glutSolidCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks);
glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);
glutSolidIcosahedron(void);
glutSolidOctahedron(void);
glutSolidTetrahedron(void);
glutSolidDodecahedron(GLdouble radius);
```

Fonte de Luz

```
glLightf(GLenum light, GLenum pname, GLfloat param);
glLighti(GLenum light, GLenum pname, GLint param);
glLightfv(GLenum light, GLenum pname, const GLfloat *params);
glLightiv(GLenum light, GLenum pname, const GLint *params);
```

onde *light* especifica qual fonte de luz está sendo alterada

pname especifica qual parâmetro de luz está sendo determinado

param (GLfloat ou GLint) para parâmetros como concentração, atenuação e tamanho de cones.

- É possível utilizar pelo menos 8 fontes de luz

Iluminação

Passos necessários

- Habilitar iluminação
- glEnable (GL_LIGHTING);
- Configurar material dos objetos
- Especificar vetores normais
- Configurar modelo de iluminação
- Configurar fontes de luz

Iluminação

Vetores normais

- Para realizar os cálculos de iluminação, OpenGL espera que os vetores normais sejam unitários.
- Transformações como glScale alteram o comprimento dos vetores normais
- É possível solicitar ao OpenGL que converta todos os vetores normais para unitários
    - glEnable (GL_NORMALIZE);
    - Entretanto, isso representa um custo adicional que pode ser evitado.
- Vetores normais podem ser associados a cada vértice
    - glNormal3f (0.0f, 1.0f, 0.0f);
    - glVertex3f (a, b, c);
    - glNormal3f (0.0f, 0.0f, 1.0f);
    - glVertex3f (d, e, f);

Iluminação

- **Modelo de Tonalização**

glShadeModel(GLenum mode);

Flat shading: GL_FLAT
Shading suave: GL_SMOOTH

- **Propriedades de Iluminação do Material**

glMaterialf(Glenum face, GLenum pname, GLfloat param);
glMateriali(GLenum face, GLenum pname, GLint param);
glMaterialfv(GLenum face, GLenum pname, const GLfloat *params);
glMaterialiv(GLenum face, GLenum pname, const GLint *params);

Iluminação

- **Modelo de Iluminação**

GL_LIGHT_MODEL_AMBIENT: é usado para especificar a luz ambiente.
GL_LIGHT_MODEL_TWO_SIDE: é usado para indicar se ambos os lados de um polígono são iluminados.
GL_LIGHT_MODEL_LOCAL_VIEWER: modifica o cálculo dos ângulos de reflexão especular;

```
glLightModelf(GLenum pname, GLfloat param);
glLightModeli(GLenum pname, GLint param);
glLightModelfv(GLenum pname, const GLfloat *params);
glLightModeliv(GLenum pname, const GLint *params);
```

Texturas

- **Habilitar o uso de texturas**

```
glEnable (GL_TEXTURE_2D);
```

- **Definir a forma de armazenamento dos pixels na textura**

```
glPixelStorei (GL_UNPACK_ALIGNMENT, 1);
```

- **Definir quantas texturas serão usadas**

```
GLuint texture_id[MAX_NO_TEXTURES];
glGenTextures (1, texture_id);
```

- **Definr a identidade da textura**

```
texture_id[0] = 1001;
```

Texturas

- **Define a textura corrente**

```
glBindTexture (GL_TEXTURE_2D, texture_id[0]);
```

- **Carrega uma imagem TGA**

```
image_t temp_image;
tgaLoad  ( "TGAImage.tga", &temp_image, TGA_FREE | TGA_LOW_QUALITY );
```

Links

Delphi
http://www.glscene.org
http://www.delphi3d.net
http://www.delphi3d.net/articles/viewarticle.php?article=c_tutor.htm
http://delphigl.cfxweb.net/

Visual Basic
http://is6.pacific.net.hk/~edx/

Java

http://www.jausoft.com/gl4java.html

OpenGL

http://www.opengl.org
http://nehe.gamedev.net
http://www.gametutorials.com
http://www.opengl.org/developers/documentation/glut/index.html
http://www.opengl.org/developers/documentation/

FIM

www.campus.com.br