

André Tavares da Silva

andre.silva@udesc.br

The Red Book e <http://openglbook.com/the-book.html>

O que é OpenGL?

- **Open Graphics Library**
- “*Uma interface de software com o hardware gráfico*”;
- Atualmente na versão 4.6
- Aproximadamente 600 comandos;
- Uma camada de abstração entre o programa de aplicação e o hardware gráfico;
- Construída na forma de API;

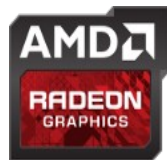
O que é OpenGL?

- Independente do sistema operacional e do hardware;
- Projetada para aplicações gráficas interativas 2D e 3D;
- Controlada pelo ARB (Architecture Review Board):



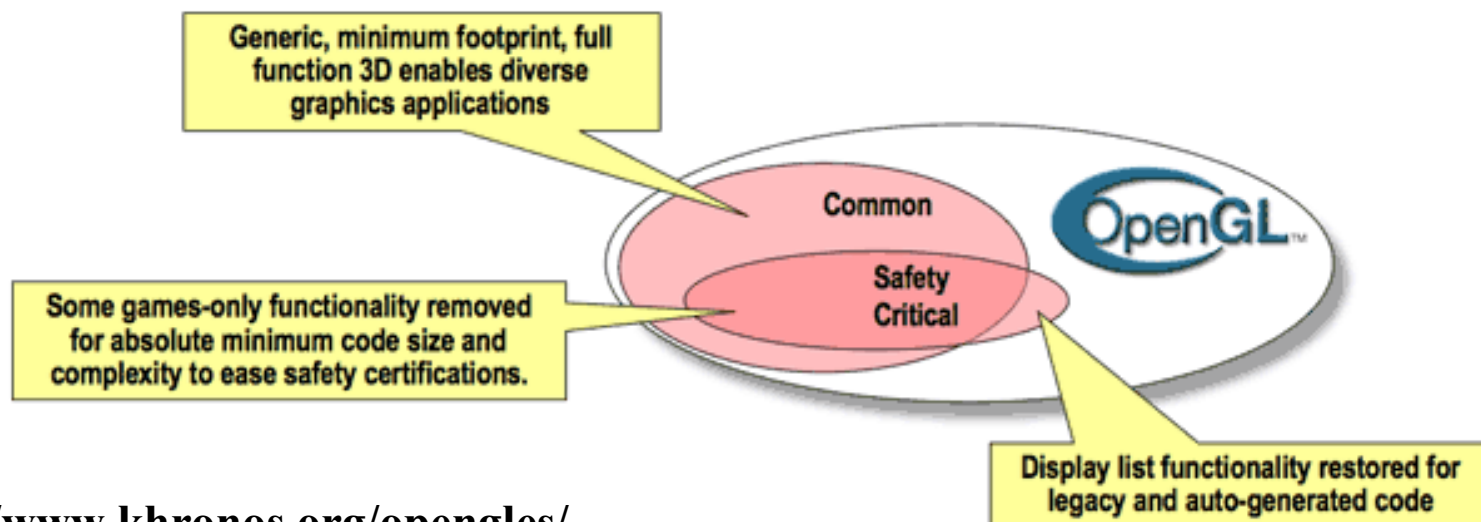
O que é OpenGL?

- Independente do sistema operacional e do hardware;
- Projetada para aplicações gráficas interativas 2D e 3D;
- Controlada pelo ARB (Architecture Review Board):
- www.khronos.org





- Subconjunto da OpenGL com algumas extensões.
- OpenGL ES-SC: para aplicações críticas (ex.: aviação e automotiva)



Características

- Rapidez;
- Portabilidade;
- Estabilidade;
- Excelente qualidade visual;
- Padrão de indústria;
- Suporte para:
 - Mapeamento de texturas;
 - Iluminação;
 - Transparência;
 - Animação;
 - ...

O que OpenGL faz?

- Constrói formas a partir de objetos primitivos como pontos, linhas, polígonos, imagens e *bitmaps*;
- Permite visualizar essas primitivas de qualquer ponto de um espaço tridimensional;
- Calcula as cores dos objetos, que podem ser pré-definidas, determinadas por condições de iluminação, texturas, ou uma combinação dessas três;
- Converte as primitivas para *pixels* (rasterização);

O que OpenGL não faz?

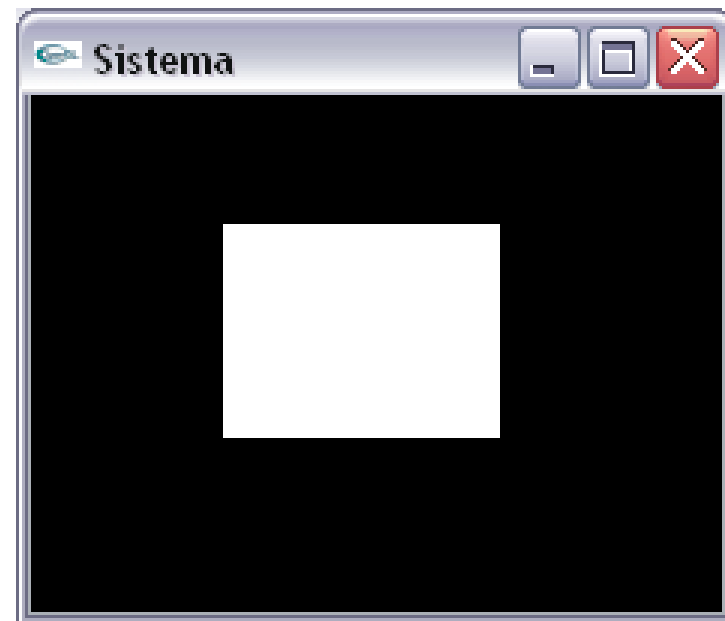
- Tarefas relacionadas a manipulação de janelas;
- Não processa nenhuma entrada de usuário;
- Não possui comandos de alto nível para construir objetos de 3 dimensões;
- Não armazena estrutura de dados;

E como fazer?

- Manipulação de janelas e entrada de usuário:
 - **Glut/FreeGlut, GLui, SDL, GLFW, GLX, GLAux, QT, wxWidgets, GTK, FLTK, GLShell,...**
- Comandos de alto nível para construção de objetos de 3 dimensões: **GLU**

Exemplo de código em OpenGL

```
glClearColor(0.0, 0.0, 0.0, 0.0);  
glClear(GL_COLOR_BUFFER_BIT);  
glColor3f(1.0,1.0,1.0);  
glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);  
glBegin(GL_POLYGON);  
    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();  
glFlush();
```



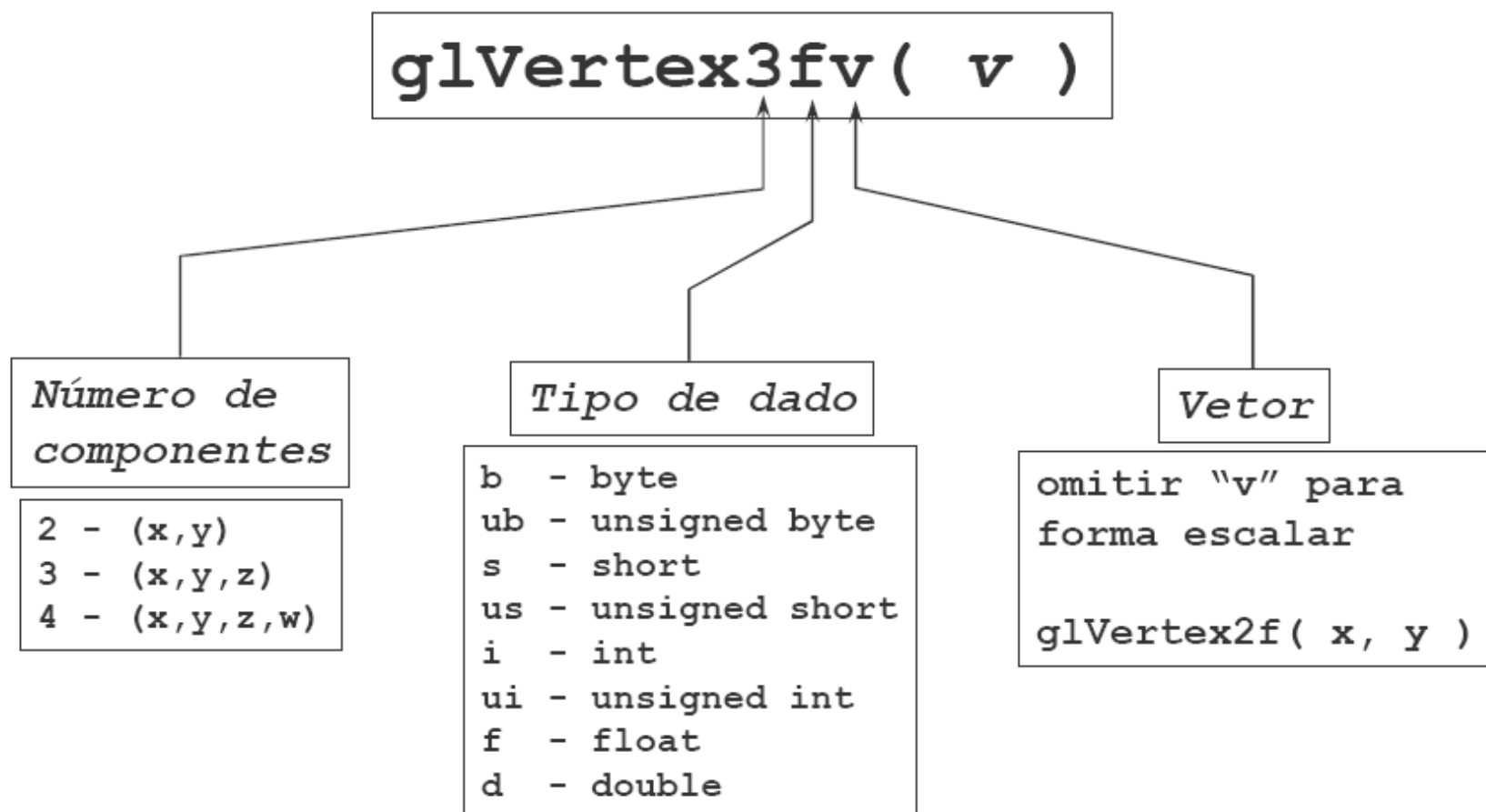
Aspectos Técnicos

- Primitivas são pontos, linhas, polígonos, imagens e *bitmaps*;
- O desenho das primitivas é afetado pelo estado atual das variáveis de controle:
 - escala, rotação, translação, cor, iluminação, textura, largura de linhas, etc.
- As variáveis de controle têm valores *default*;

Sintaxe dos Comandos

- Todos os comandos OpenGL seguem um padrão. As funções sempre começam com **gl**.
- Indica:
 - Função
 - Quantidade de parâmetros
 - Tipo de dados dos parâmetros

Sintaxe dos Comandos



Sintaxe dos Comandos

- As constantes começam com GL_ e todas as letras são maiúsculas.

Exemplos:

- GL_COLOR_BUFFER_BIT
- glEnable()
- glVertex2f()
- GLfloat vertex_array[] = {0.0, 1.0, 0.0};
- glVertex3fv(vertex_array);

Tipos de Dados

- Problema: Cada compilador ou ambiente possui regras diferenciadas para o tamanho das variáveis.
- Solução: Tipos de dados próprios do OpenGL.
- Portabilidade garantida.

Tipos de Dados

Tipo de dado OpenGL	Representação interna	Tipo de dado C equivalente	Sufixo
GLbyte	8-bit integer	signed char	b
GLshort	16-bit integer	short	s
GLint, GLsizei	32-bit integer	int ou long	i
GLfloat, GLclampf	32-bit floating-point	float	f
GLdouble, GLclampd	64-bit floating-point	double	d
GLubyte, GLboolean	8-bit unsigned integer	unsigned char	ub
GLushort	16-bit unsigned integer	unsigned short	us
GLuint, GLenum, GLbitfield	32-bit unsigned integer	unsigned long ou unsigned int	ui

Máquina de Estados

- Cada estado *OpenGL* vale até o programa explicitamente trocá-lo;
- Ao definir uma cor, ela valerá para qualquer objeto na sequência até um novo comando de cor;
- Outros exemplos de estado: padrão de preenchimento de linhas e polígonos, posições e características de luzes, propriedades dos materiais, transformações...

Máquina de Estados

- Algumas variáveis de estado são habilitadas ou desabilitadas usando `glEnable` / `glDisable`;
- `glEnable(GL_LIGHTING)` habilita o cálculo da iluminação pelas fontes de luz;
- **DICA:** a troca frequente de estados afeta a *performance*. O melhor é “setar” os estados uma vez só, quando possível.

Máquina de Estados

- Os valores das variáveis podem ser recuperados através de funções:
 - glGetBooleanV(), glGetDoubleV(), glEnable ()
- É possível salvar e restaurar um conjunto de variáveis em uma pilha, para depois restaurá-las:
 - void glPushAttrib(GLbitfield mask)
 - void glPopAttrib(void)

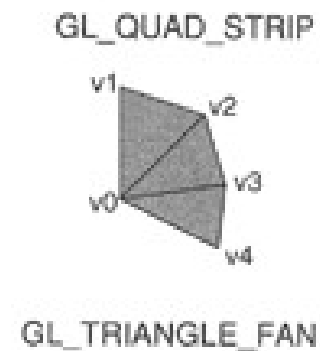
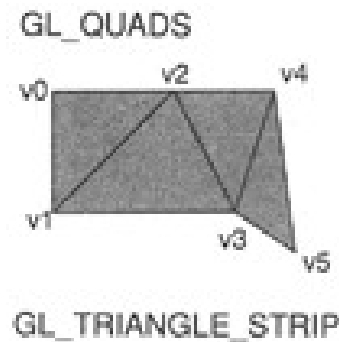
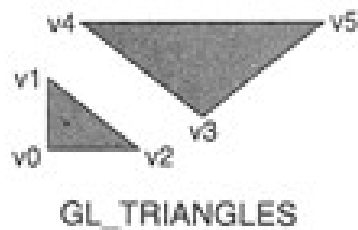
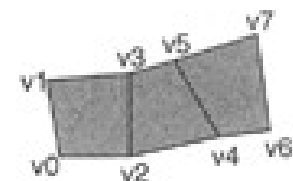
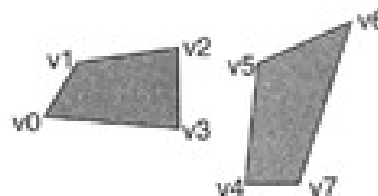
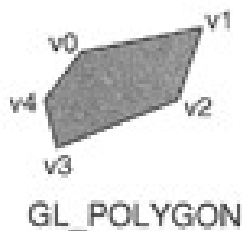
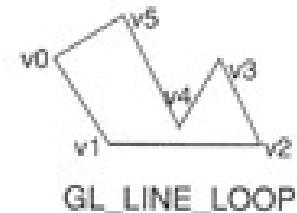
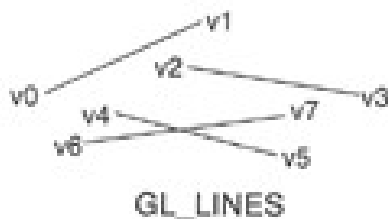
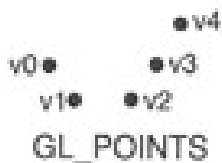
Desenhando

- Primitivas são delimitadas por `glBegin()` e `glEnd()`;
- `glVertex*()` especifica o vértice;
- Atributos de um vértice:
 - `glColor*()`
 - `glNormal*()`
 - `glMaterial*()`

Desenhando

Valor	Descrição
GL_POINTS	Pontos individuais
GL_LINES	Par de vértices = 1 linha
GL_LINE_STRIP	Pontos conectados por linhas
GL_LINE_LOOP	= strip + 1 linha para fechar
GL_TRIANGLES	3 pontos = 1 triângulo
GL_TRIANGLE_STRIP	Série de triângulos conectados
GL_TRIANGLE_FAN	Todos tri compartilham ponto zero
GL_QUADS	4 pontos = 1 quadrilátero
GL_QUAD_STRIP	Série de quads. conectados
GL_POLYGON	Todos pontos = 1 polígono

Desenhando



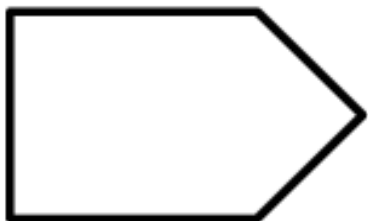
Desenhando

Desenhar um triângulo vermelho no plano $z=0$

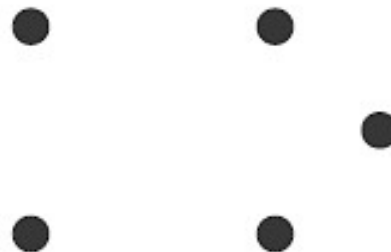
```
glBegin(GL_TRIANGLE);  
    glColor3f(1,0,0); // seta cor para vermelho  
    glNormal3f(0,0,1); // seta normal para (0,0,1)  
    glVertex3f(0,0,0); // primeiro vértice  
    glVertex3f(1,0,0); // segundo vértice  
    glVertex3f(0,1,0); // terceiro vértice  
glEnd();
```

Desenhando

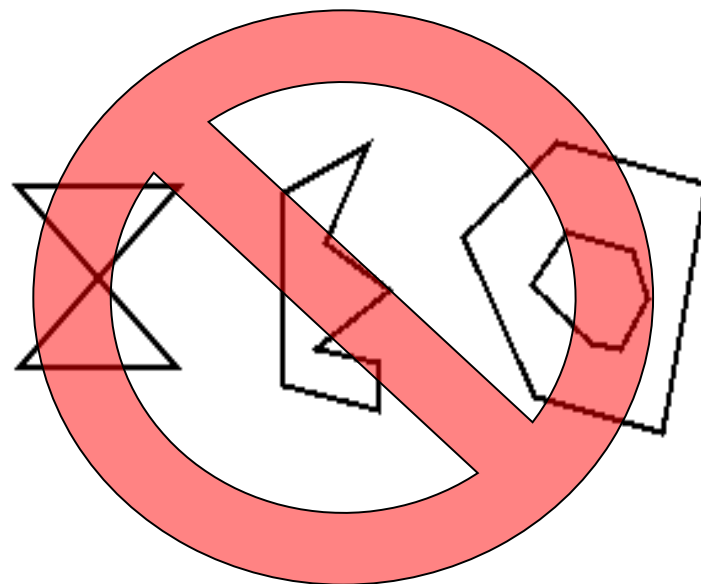
```
glBegin(GL_POLYGON);  
    glVertex2f(0.0, 0.0);  
    glVertex2f(0.0, 3.0);  
    glVertex2f(4.0, 3.0);  
    glVertex2f(6.0, 1.5);  
    glVertex2f(4.0, 0.0);  
glEnd();
```



```
glBegin(GL_POINTS);  
    glVertex2f(0.0, 0.0);  
    glVertex2f(0.0, 3.0);  
    glVertex2f(4.0, 3.0);  
    glVertex2f(6.0, 1.5);  
    glVertex2f(4.0, 0.0);  
glEnd();
```

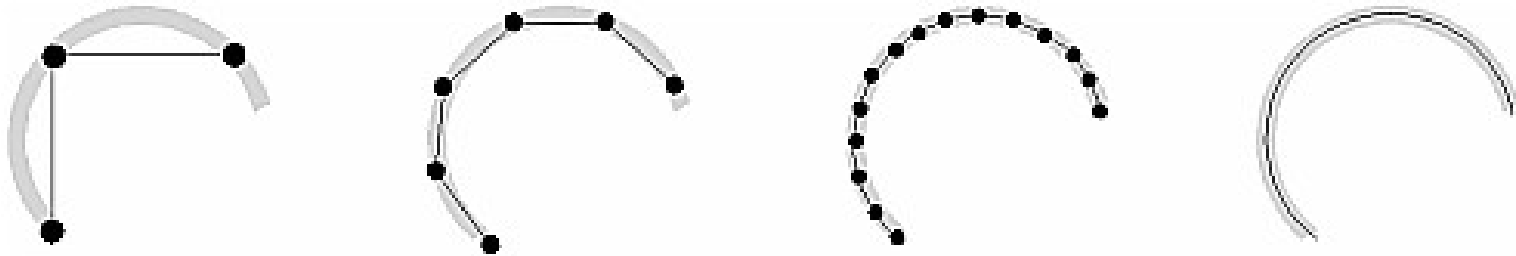


Polígonos Válidos e Inválidos



Apenas são válidos polígonos simples e convexos.

Curvas



Devem ser aproximadas por retas.

Curvas

```
#define PI 3.1415926535898
GLfloat circle_points = 100;
glBegin(GL_LINE_LOOP);
    for (i = 0; i < circle_points; i++)
    {
        angle = 2*PI*i/circle_points;
        glVertex2f(cos(angle), sin(angle));
    }
glEnd();
```

Erros

- Quando ocorre algum erro em OpenGL ou GLU o programa não para; apenas não faz o esperado, o que dificulta a depuração!
- Quando um erro ocorre um *flag* é setado com um código de erro. Para acessar esse código deve-se usar `glGetError()`;
- Por outro lado... cada chamada *OpenGL* a mais afetam a *performance*... Em alguns casos quando queremos renderização em tempo real temos que eliminar o maior número possível dessas chamadas!

Transformações

- Matrizes de Projeção
 - `glFrustum(left,right,bottom,top,near,far)`
 - `glOrtho(left,right,bottom,top,near,far)`
 - `gluPerspective(fovy,aspect,zNear,zFar)`
 - `gluOrtho2D(left,right,bottom,top)`
 - `gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)`
- Coordenadas de Tela
 - `glViewport(x, y, width, height)`

Transformações

- Modelagem
 - `glTranslate{fd}(x, y, z)`
 - `glRotate{fd}(angle, x, y, z)`
 - `glScale{fd}(x,y,z)`
- Propriedades Materiais `glMaterial*()`
 - ambiente, difuso, especular
- Fontes de Luz `glLight*()`
 - cor, posição, atenuação, ...

Matrizes

- Três tipos de matrizes:
 - **GL_MODELVIEW**
 - **GL_PROJECTION**
 - **GL_TEXTURE**
- A matriz *modelview* controla as transformações dos vértices dos objetos da cena
- A matriz de projeção controla como a cena 3-D é projetada em 2-D
- A matriz de texturas (geralmente pouco conhecida e utilizada) transforma as coordenadas das texturas para obter efeitos como projetar e deslocar texturas

Exemplo

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
glFrustum(-1, 1, -1, 1, 0.0, 40.0);  
glMatrixMode(GL_MODELVIEW);  
glPushMatrix();  
    glRotatef(45.0, 1.0, 1.0, 1.0);  
    render();  
glPopMatrix();
```

Dicas

- A origem do *OpenGL* é no canto esquerdo (*left-hand*);
- Lembre-se de:
 - ativar o modo de preenchimento desejado:
`glEnable(GL_DEPTH_TEST);`
 - Limpar os respectivos *buffers*:
`glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);`

Dicas

- Evitar mudanças constantes de estado;
- Ao manipular as matrizes *OpenGL* lembre-se de verificar se o tipo de matriz correto está ativado;

GLU

OpenGL Utility Library

GLU - *OpenGL Utility Library*

- Conjunto de rotinas utilizadas frequentemente
- Acompanha todas as distribuições OpenGL
- Construídas a partir de comandos OpenGL
- Criada para prover rotinas de desenho de alto nível à OpenGL e é normalmente distribuída junto com o pacote básico da OpenGL

GLU - *OpenGL Utility Library*

- Rotinas para:
 - mapeamento entre coordenadas de tela e de mundo
 - geração de mipmaps de texturas
 - manipulação (*tessellation*) de primitivas poligonais
 - manipulação de projeções
 - desenho de superfícies quádricas de objetos tridimensionais como esferas, cilindros e discos
 - NURBS

GLUT

OpenGL Utility Toolkit

GLUT

- Graphics Library Utility Toolkit
- Biblioteca de software que se integra com OpenGL (autor: Mark J. Kilgard - SGI)
- Habilita acesso aos eventos do sistema operacional necessários em programas interativos
 - mouse, teclado, display, etc.

GLUT

- Até onde possível, independente do sistema de janelas nativo;
- Programas de tamanho pequeno e médio;
- Não apresenta muitos recursos de interface (apenas menus);
- Programação orientada a eventos;

Programação Interativa

- Programas gráficos são na maioria dos casos Interativos
 - Usuário controla a execução do programa
- Controle é executado via dispositivos de entrada (mouse, teclado, etc.)

Eventos

- Um gerenciador monitora em *background* a ocorrência de *eventos* e os coloca numa fila de eventos.
- *Evento*: mudança no estado do dispositivo causada pela ação do usuário
- Programa principal, à sua conveniência, verifica a fila de eventos

Eventos

- Primeiro evento da fila é removido
- Controle da execução é transferido para execução de uma rotina associada ao evento
- Se a fila estiver vazia, programa principal segue execução (rotina *idle*)
- Exemplos de eventos: click do mouse, tecla pressionada, janela redimensionada, etc.

Rotinas GLUT

- Iniciam com prefixo glut
glutInitDisplayMode
- (0,0) canto superior esquerdo da janela
- Incluir diretiva: **#include <GL/glut.h>**
- Incluir a lib na compilação: **glut32.lib**

Rotinas GLUT

- **glutInit(int *argc, char **argv);**
 - Inicializa o GLUT; Primeira função GLUT a ser chamada;
- **void glutInitDisplayMode (unsigned int mode);**
 - Define uma série de propriedades associadas às janelas que venham a ser criadas;
- **void glutInitWindowPosition(int x, int y);**
 - Define a posição da janela na tela;
- **void glutInitWindowSize(int largura, int altura);**
 - Define o tamanho da janela;

Rotinas GLUT

- **int glutCreateWindow(char* name);**
 - Abre uma janela com as características previamente definidas. Retorna o identificador da janela;
- **void glutDisplayFunc (void (*func)(void));**
 - Especifica a função a ser chamada sempre que o conteúdo de uma janela tem que ser redesenhado (abertura, pop-up, objetos sobrepostos...);
- **void glutMainLoop (void);**
 - Função para ativação do ciclo infinito de processamento de eventos.

Rotinas GLUT

- **void glutPostRedisplay(void);**
 - Força o redesenho da janela;
- **void glutReshapeFunc(void (*func)(int x, int y));**
 - Especifica a função a ser chamada sempre que a janela é redimensionada;
- **void glutKeyboardFunc(void (*func)(char key, int x, int y));**
- **void glutMouseFunc(void (*func)(int b, int e, int x, int y));**
 - Especificam quais funções iram tratar dos eventos do teclado e do mouse.

Rotinas GLUT

- **void glutMotionFunc(void (*func)(int x, int y));**
 - Especifica a função a ser chamada sempre que o mouse é movido com o botão pressionado;
- **void glutIdleFunc(void (*func)(void));**
 - Especifica qual função irá ser executada enquanto não existem eventos a serem tratados.

Objetos 3D Pré-definidos

- `void glutWire*(parâmetros)`
- `void glutSolid*(parâmetros)`

* Objetos Possíveis

- cylinder cube cone dodecahedron icosahedron octahedron sphere teapot torus
- Exemplos:
 - `glutWireCube(GLdouble size);`
 - `glutSolidCube(GLdouble size);`