Introdução ao OpenGL (continuação)

Background

- Necessidade de encontrar uma API uniforme para a programação de sistemas gráficos
  - esforços anteriores de normalização: GKS, PHIGS, ...
- Sistemas proprietários
  - HP (Starbase), SGI (**G**raphics **L**ibrary)
- OpenGL (1992)
- Concorrência apenas por parte da Microsoft com o Direct3D

Características

- API com cerca de 250 comandos para o hardware
  - independente de hardware específico
- Open
  - política de licenciamento aberta
  - gerido pela **A**rquitecture **R**eview **B**oard, formada por representantes da nVIDIA, AMD, IBM, Intel, SGI, ...
  - Extensões específicas dos fabricantes (normalmente para funcionalidades experimentais)
- Não inclui:
  - gestão das janelas
  - o input do utilizador (rato/teclado/touch)

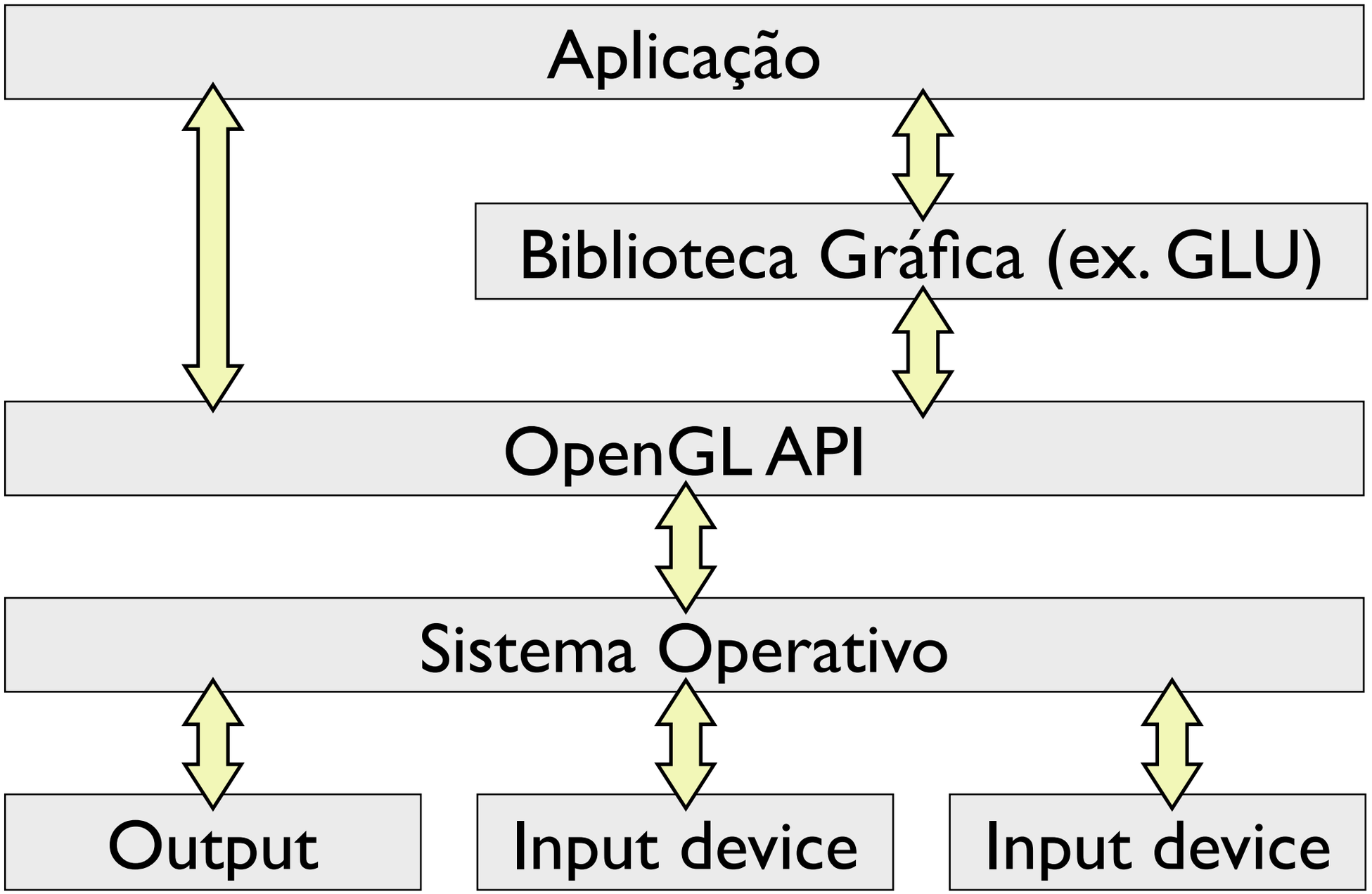
Vantagens

- "norma" para a produção de imagens 3D;
  - biblioteca C/C++ disponibilizada por vários fabricantes: nVIDIA, AMD, Microsoft, Qualcomm, (Mesa), etc;
  - incluída em qualquer sistema Windows, Mac, Linux, Android ou iOS;
- independente da plataforma
- suportada pelo hardware
- independente do gestor de janelas

Aspetos Relevantes

- Core: primitivas básicas (pontos, linhas, polígonos,...)
- Bibliotecas construídas sobre o OpenGL:
  - GLU (Open**GL U**tility Library) - oferece funções de conveniência e alguns objetos 3D típicos (Cubos, Quadrics, NURBS, ...)
- Baseia-se numa máquina de estados
  - o estado mantém-se até ser alterado (cor, coordenadas de textura, vectores normais, transformações, ...)
  - permite a redução do número de parâmetros a usar nas chamadas das funções da API

Organização em camadas



Estrutura básica

- API de baixo nível
  - perto do hardware mas independente do mesmo
- Funcões principalmente de 2 tipos:
  - Primitivas gráficas
  - Alteração do estado do sistema
- Um sistema imediato puro (sem retenção):
  - comandos simples
  - passagem direta para o hardware
  - sem representação interna da cena
- Namespace coerente:
  - comandos iniciados por gl_, constantes por GL_

Primitivas Geométricas

- Todas as primitivas geométricas são definidas pelos seus vértices

Comando OpenGL: `glVertex*()`

- Significado do sufixo:

```
glVertex2f(float v1, float v2)

glVertex3i(int v1, int v2, int v3)

glVertex4dv(double p[4])
```

número de argumentos

tipo dos argumentos (int, float, double,...)

usa-se o sufixo "v" para representar argumentos do tipo vetor (array)

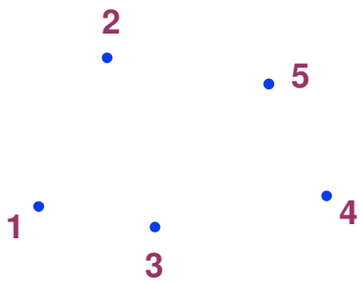
- Exemplos:

```
glVertex2s( 2, 3 );
glVertex3d( 0.0, 0.0, 3.1415926535898 );
glVertex4f( 2.3, 1.0, -2.2, 2.0 );
Gldouble ad_vect[3] = {5.0, 9.0, 1992.0};
glVertex3dv( ad_vect );
```

Procedimento base para o desenho

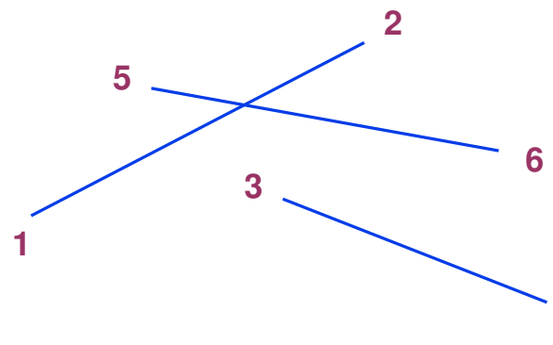
- Delimitar a primitiva com `glBegin()` ... `glEnd()`:
  - pelo meio todo o código válido na linguagem é permitido
  - colocar, para além dos vértices `glVertex*()`, outros comandos, tais como: `glColor*()`, `glNormal*()`, os quais permitem mudar os valores dos atributos dos vértices
  - não usar outros comandos
  - os atributos devem ser modificados antes de se enviarem as coordenadas do vértice
- Modelo cliente-servidor:
  - cliente (aplicação) produz vértices e valores dos atributos
  - servidor (driver+hardware) regista pedidos num *buffer*
  - `glFinish()` e `glFlush()` permitem indicar o fim dum *frame* e esvaziar o *buffer*.

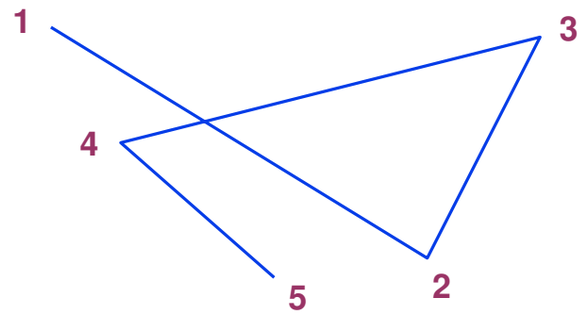
Primitivas



GL_POINTS

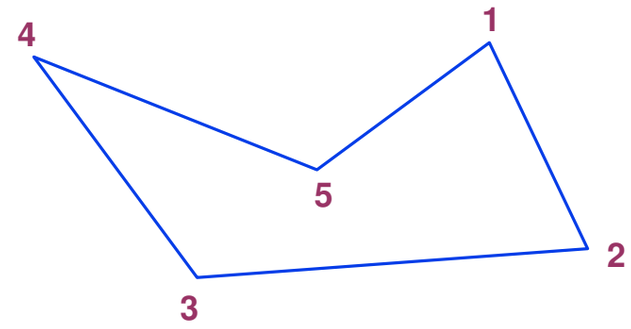
GL_LINES



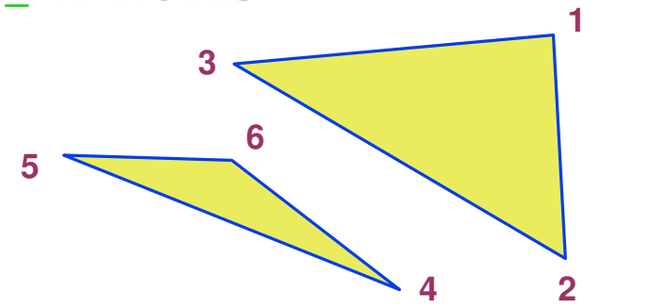


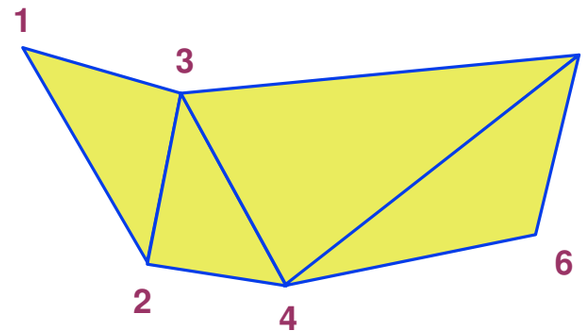
GL_LINE_STRIP

GL_LINE_LOOP



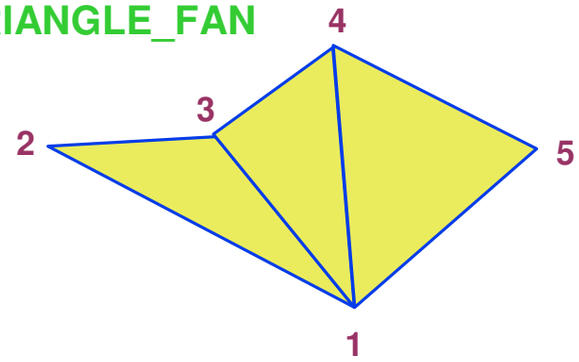
GL_TRIANGLES



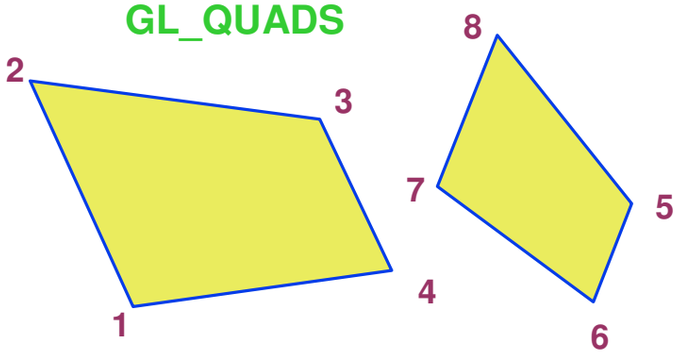


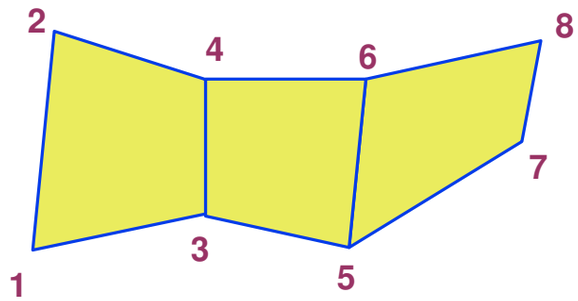
GL_TRIANGLE_STRIP

GL_TRIANGLE_FAN



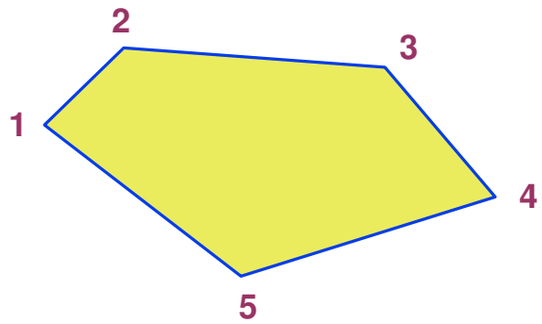
GL_QUADS





GL_QUAD_STRIP

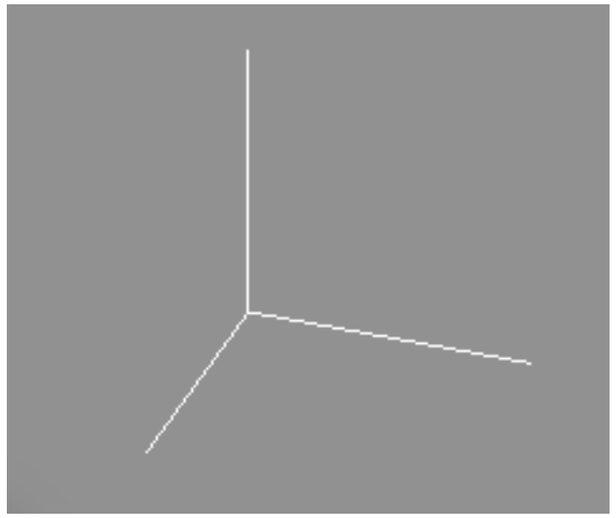
GL_POLYGON



M.Próspero

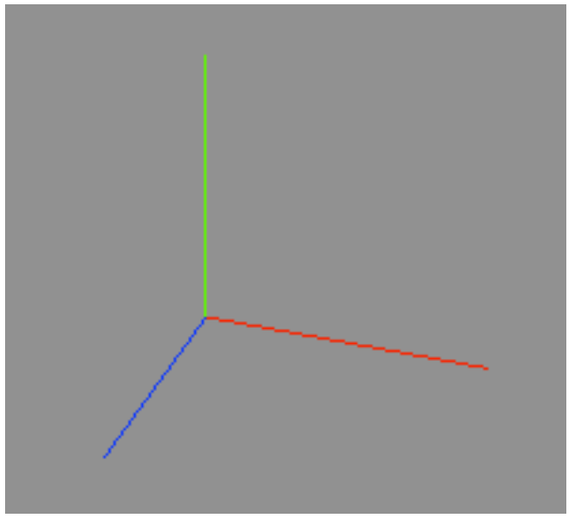
Exemplos

```
glBegin( GL_LINES );
   glVertex3f( 0.0, 0.0, 0.0 );
   glVertex3f( 1.0, 0.0, 0.0 );
   glVertex3f( 0.0, 0.0, 0.0 );
   glVertex3f( 0.0, 1.0, 0.0 );
   glVertex3f( 0.0, 0.0, 0.0 );
   glVertex3f( 0.0, 0.0, 1.0 );
glEnd( );
```



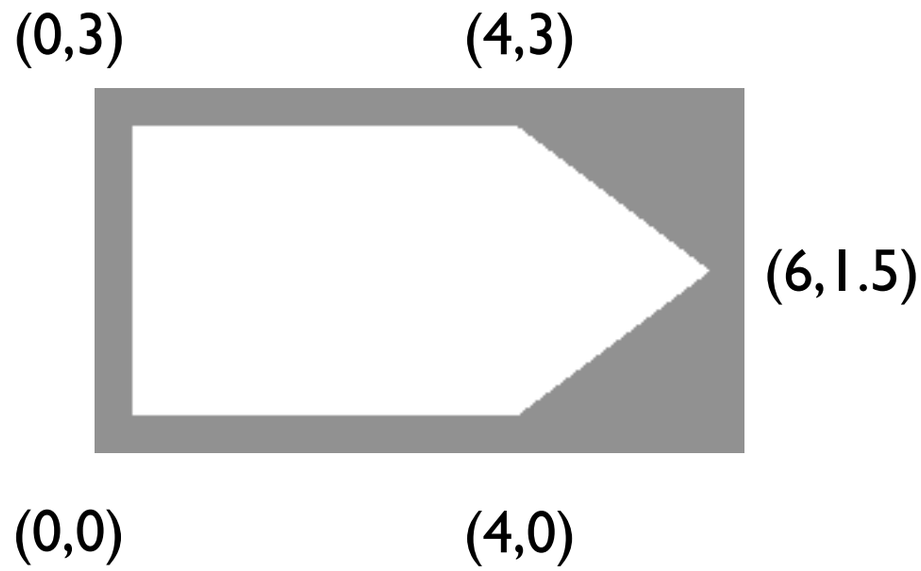
Exemplos

```
glBegin( GL_LINES );
   glColor3f (1.0, 0.0, 0.0);
   glVertex3f( 0.0, 0.0, 0.0 );
   glVertex3f( 1.0, 0.0, 0.0 );
   glColor3f (0.0, 1.0, 0.0);
   glVertex3f( 0.0, 0.0, 0.0 );
   glVertex3f( 0.0, 1.0, 0.0 );
   glColor3f (0.0, 0.0, 1.0);
   glVertex3f( 0.0, 0.0, 0.0 );
   glVertex3f( 0.0, 0.0, 1.0 );
glEnd( );
```



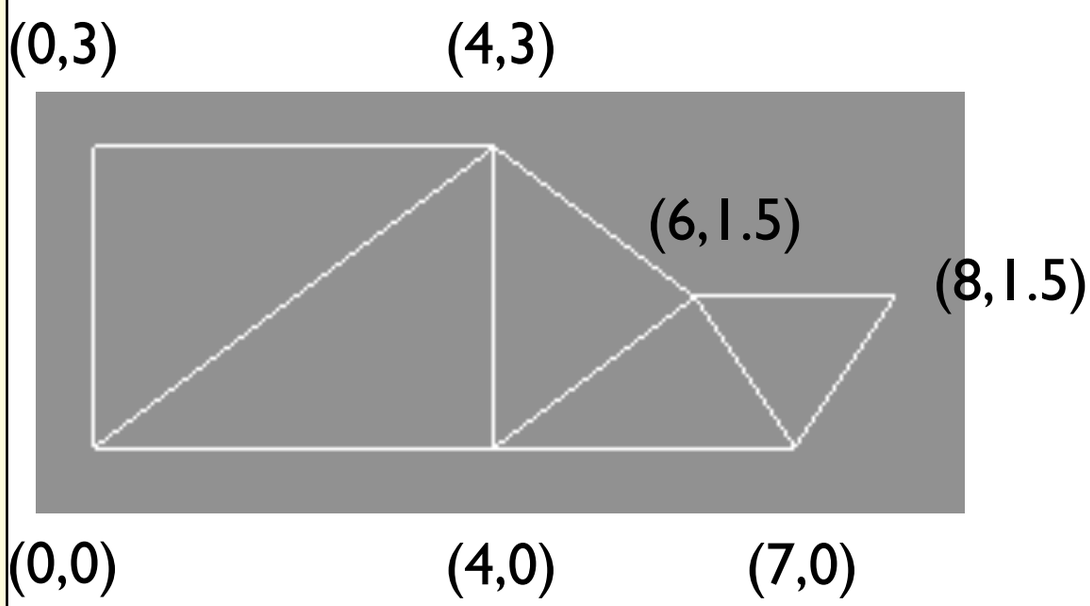
Exemplos

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



Exemplos

```
glBegin( GL_TRIANGLE_STRIP );
   glVertex2f( 0.0, 3.0 );
   glVertex2f( 0.0, 0.0 );
   glVertex2f( 4.0, 3.0 );
   glVertex2f( 4.0, 0.0 );
   glVertex2f( 6.0, 1.5 );
   glVertex2f( 7.0, 0.0 );
   glVertex2f( 8.0, 1.5 );
glEnd( );
```



glFinish()

Usa-se para terminar um frame

glFinish

NAME

glFinish - block until all GL execution is complete

C SPECIFICATION

void **glFinish**(void)

DESCRIPTION

glFinish does not return until the effects of all previously called GL commands are complete. Such effects include all changes to GL state, all changes to connection state, and all changes to the frame buffer contents.

NOTES

glFinish requires a round trip to the server.

ERRORS

GL_INVALID_OPERATION is generated if **glFinish** is called between a call to **glBegin** and the corresponding call to **glEnd**.

From OpenGL reference manual
at http://www.glprogramming.com/blue/

glFlush()

Usa-se para sincronizar input com output

glFlush

NAME

glFlush - force execution of GL commands in finite time

C SPECIFICATION

void **glFlush**(void)

DESCRIPTION

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. **glFlush** empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

Because any GL program might be executed over a network, or on an accelerator that buffers commands, all programs should call **glFlush** whenever they count on having all of their previously issued commands completed. For example, call **glFlush** before waiting for user input that depends on the generated image.

NOTES

glFlush can return at any time. It does not wait until the execution of all previously issued OpenGL commands is complete.

ERRORS

GL_INVALID_OPERATION is generated if **glFlush** is called between a call to **glBegin** and the corresponding call to **glEnd**.

Immediate vs Retained

- Modo imediato
  - modo direto
  - as primitivas não são mantidas em memória
- Modo retido
  - as primitivas são armazenadas pelo sistema gráfico em listas
  - as listas podem ser armazenadas na placa gráfica
  - as listas podem ser reutilizadas com diferentes atributos/transformações
  - potencial aumento do desempenho

Estados/Propriedades

- O OpenGL é uma grande máquina de estados
- Os estados permitem controlar
  - atributos das primitivas
  - iluminação
  - sombreamento
  - mapeamento de texturas
  - modo de visualização: wireframe/preenchido, testes de visibilidade, ...
- Os estados podem ser modificados com `glEnable()`/ `glDisable()`
- `glGet()` permite obter informação sobre um estado/valor de uma propriedade

Suporte a operações matriciais

- O OpenGL oferece 4 matrizes (de dimensão 4x4)*:
  - GL_MODELVIEW — usada para guardar as transformações geométricas dos objetos
  - GL_PROJECTION — usada para a matriz de projeção
  - GL_TEXTURE — usada para transformar coordenadas de textura
  - GL_COLOR — usada para efetuar conversão de espaços de cor

* na verdade são 4 pilhas de matrizes, com suporte para operações Push e Pop.

Operações relacionadas com matrizes

- Definição da pilha de matrizes corrente (sujeita a manipulação através de comandos):

glMatrixMode(mode)

mode: GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE or GL_COLOR

- A matriz corrente é a matriz que se encontra no topo da pilha corrente

- Comandos para manipular a matriz corrente

glLoadIdentity() - replaces the topmost matrix with the identity matrix
glLoadMatrix*(T) - replaces the topmost matrix with T
glMultMatrix*(T) - multiplica a matriz corrente pela matriz dada: M'=MT
glRotate*(), **glTranslate***(), **glScale***() - semelhante a glMultMatrix, substituindo T por uma matriz duma transformação geométrica elementar.

- Comandos para manipular a pilha

glPushMatrix() - pushes a copy of the topmost matrix
glPopMatrix() - pops the topmost matrix

M'=M.T

- As operações que acumulam transformações na matriz corrente fazem-no à direita
- Os pontos são vetores coluna
- logo...

A última transformação é a primeira a ser aplicada!

```
glScalef(2,2,2);
glTranslatef(2,0,3);
glVertex3d(1,1,1);
```

Quais as coordenadas finais deste vértice?