Rápida Introdução ao *GLUT*

Gubi

10 de junho de 2010

Este documento lista as chamadas de função para a construção de um programa que usa a biblioteca *Glut*. A *Glut* coloca uma camada de funções sobre a bliblioteca mais fundamental *OpenGL*, que é bastante veloz e portátil.

O texto não é nem pretende ser completo, mas fornece o necessário para um programa inteiro e com alguma sofisticação. Ele foi baseado em tutoriais presentes na *internet*.

As funções estão descritas na ordem natural de sua utilização.

Para declarar e poder as funções da *Glut*, não se esqueça de incluir:

```
#include <GL/glut.h>
```

no início do programa.

1 Inicialização

A biblioteca gráfica precisa ser inicializada, antes de mais nada. Como o ela pode pegar opções de linha de comando, é necessário passar os ponteiros para os argumentos.

Para usar as chamadas do *Glut*

1.1 Primeira coisa

A primeira chamada deve ser:

```
void glutInit(int *argc, char **argv);
```

1.2 Posicionamento

Se você quiser posicionar a janela na tela (opcional), use a chamada:

```
void glutInitWindowPosition(int x, int y);
```

O valor -1 para x ou y (ou ambos) indica que o sistema escolhe a valor da coordenada. Se você não chamar esta função, a escolha fica para o sistema.

Note que o sistema não vai necessariamente obedecer sua sugestão...

1.3 Tamanho

Para definir o tamanho da janela, basta chamar:

```
void glutInitWindowSize(int width, int height);
```

1.4 Modo de Apresentação

O *Glut* e o *OpenGL* permitem uma grande variedade de formas de apresentar a janela.

O modo mais simples esta chamada pode ser pulada.

Na sua chamada mais geral, esta função define 3 características: cor, tipo dos *buffer*s[1] e quais *buffer*s usar.

A forma de indicação de cor pode ser feita por um destes valores:

- **GLUT_RGB** — 3 valores por pixel, para as componentes vermelha, verde e azul.
- **GLUT_RGBA** — idem, mas com um componente a mais, *alpha*, que indica o nível de transparência.
- **GLUT_INDEX** — para usar uma tabela de cores. Hoje em dia tem menos utilidade.

O tipo dos *buffers* pode ser simples ou duplo. O *buffer* único é mais rápido, o duplo permite animações suaves.

- **GLUT_SINGLE** — simples.

[1] Áreas de desenho

- **GLUT_DOUBLE** — duplo.

O mais simples é **GLUT_SINGLE**.
Os *buffers* que podem ser usados são os seguintes:

- Color — cada pixel contém a cor a ser representada. É o normal.
- Depth — cada pixel contém a distância do ponto à vista, assim é possível indicar se um ponto não deve aparecer por estar coberto.
- Stencil — serve para definir máscaras de desenho. Apenas algumas regiões da imagem são redesenhadas.
- Accumulation — usado para sobrepor desenhos e causar efeitos especiais.

1.5 Criação da Janela

Depois de definidas as inicializações, a janela pode ser criada.
A chamada é bem simples:

```
glutCreateWindow(char *nome);
```

O valor de retorno, inteiro, pode ser usado para referência em outras chamadas.

2 Desenhando

Como a janela pode ser encoberta, minimizada ou redimensionada, possivelmente ela terá que ser redesenhada em momentos imprevisíveis. A solução é fornecer uma função para ser chamada pelo sistema sempre que preciso.

A função é registrada pela chamada:

```
glutDisplayFunc(redesenha);
```

A função de desenho não recebe parâmetros e nem retorna valor: `void f()`.

2.1 Limpando

Para desenhar, existem diversas chamadas, dependendo do que se pretende. O primeiro passo é limpar o desenho anterior:

```
glClear(GL_COLOR_BUFFER_BIT);
```

A constante `GL_COLOR_BUFFER_BIT` indica que o *buffer* de cor deve ser limpo. Outras constantes existem para outros *buffer*s.

2.2 Objetos geométricos

É possível desenhar elementos nas 3 dimensões, que serão projetados automaticamente. O *default* é desenhar no plano XY, com $Z = 0$.

Existe uma convenção interessante na maioria das funções do *OpenGL*: o final do nome da função indica o número e tipo dos argumentos. Assim, por exemplo, `glVertex2f` recebe 2 *floats* e `glVertex3f` recebe 3. No primeiro caso, o ponto (*vertex*) será desenhado no plano XY.

Os elementos são desenhados em blocos, demarcados por `glBegin(`*`tipo do objeto`*`)`.

A cor de cada elemento é definida por

```
glColor3f(red,green,blue);
```

onde os parâmetros indicam a quantidade de vermelho, verde e azul, com valores entre 0 e 1.

O processo todo é melhor explicado através de exemplos.

2.2.1 Pontos ou vértices

Pontos azuis.

```
glBegin(GL_POINTS);
      glColor3f(0., 0., 1.);
      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();
```

Para definir o tamanho do ponto, use:

```
glPointSize(tamanho);
```

antes do `glBegin`.

2.2.2 Linhas

Linhas de várias cores.

```
glBegin(GL_LINES);
      glColor3f(1.0, 1.0, 0.0);
      glVertex2(-1.0, 1,0);
      glVertex2f(2.0, 2.0);
      glColor3f(1.0, 0.0, 0.0);
      glVertex2f(0.0, 0.0);
      glColor3f(1.0, 1.0, 1.0);
      glVertex2f(1.0, -1.0);
      glVertex2f(-2.0, -2.0);
glEnd();
```

Pares pontos são ligados e as cores vão mudando de uma extremidade a outra, se for o caso. Se o número de vértices for ímpar, o último é ignorado.

Para fazer linhas conectadas, basta usar `GL_LINE_STRIP`. Para fechar o ciclo, use `GL_LINE_LOOP`.

De forma parecida com os pontos, a largura da linha pode ser definida com

```
glLineWidth(largura);
```

2.2.3 Polígonos

O mecanismo geral é o mesmo, indique os vértices dentro de um bloco `glBegin(GL_POLIGON);` — `glEnd();`. A *restrição* é que o polígono deve obrigatoriamente ser *convexo*.

Existem algumas variações:

- `GL_TRIANGLES` — um triângulo é desenhado a cada 3 vértices.
- `GL_TRIANGLE_STRIP` — um triângulo é desenhado a cada vértice a partir do terceiro. Isto é, triângulos consecutivos compartilham um lado.

- `GL_TRIANGLE_FAN` — um triângulo é desenhado a cada vértice a partir do terceiro, mas todos compartilham o primeiro vértice, como em uma rosácea.
- `GL_QUADS` — desenha quadriláteros a cada 4 vértices.
- `GL_QUAD_STRIP` — desenha quadriláteros a cada 2 vértices a partir o quarto, compartilhando um lado.

Existem algumas outras formas para desenhar, procure