INTRODUÇÃO A OPENGL

Isabel Harb Manssour (manssour@inf.pucrs.br)
Professora da Faculdade de Informática da PUCRS
Porto Alegre, RS

01. INTRODUÇÃO

OpenGL é definida como "um programa de interface para *hardware* gráfico". Na verdade, OpenGL é uma biblioteca de rotinas gráficas e de modelagem, bi (2D) e tridimensional (3D), extremamente portável e rápida. OpenGL permite a criação de aplicações gráficas interativas 2D e 3D com uma grande qualidade visual, tendo como principal vantagem a rapidez, uma vez que usa algoritmos desenvolvidos e otimizados pela *Silicon Graphics, Inc.*, líder mundial em Computação Gráfica e Animação [3].

OpenGL não é uma linguagem de programação, é uma poderosa e sofisticada API (*Application Programming Interface*) para criação de aplicações gráficas. Seu funcionamento é semelhante ao de uma biblioteca C, uma vez que fornece uma série de funcionalidades. Normalmente se diz que um programa é baseado em OpenGL ou é uma aplicação OpenGL, o que significa que ele é escrito em alguma linguagem de programação que faz chamada a uma ou mais bibliotecas OpenGL.

As aplicações OpenGL variam de ferramentas CAD a programas de modelagem usados para criar personagens para o cinema, tal como um dinossauro. Além do desenho de primitivas gráficas, como por exemplo, linhas e polígonos, OpenGL dá suporte a iluminação, colorização, mapeamento de textura, transparência, animação, entre outros efeitos especiais. Atualmente, OpenGL é reconhecida e aceita como um padrão API para desenvolvimento de aplicações gráficas 3D em tempo real.

Ao invés de descrever a cena e sua aparência, quando se está utilizando OpenGL é preciso apenas determinar os passos necessários para alcançar o efeito desejado. Estes passos envolvem chamadas a esta API, que inclui aproximadamente 250 comandos e funções (200 comandos da OpenGL e 50 comandos da GLU - *OpenGL Utility Library*). Por ser portável, OpenGL não possui funções para gerenciamento de janelas, interação com o usuário ou arquivos de entrada/saída. Cada ambiente, como por exemplo o *Microsoft Windows*, possui suas próprias funções para estes propósitos. Não existe um formato de arquivo OpenGL para modelos ou ambientes virtuais. OpenGL fornece um pequeno conjunto de primitivas gráficas para construção de modelos, tais como pontos, linhas e polígonos. A biblioteca GLU (que faz parte da implementação OpenGL) é que fornece várias funções para modelagem [2, 3].

02. UTILIZAÇÃO

Como uma API, OpenGL segue a convenção de chamada da linguagem C. Isto significa que programas escritos em C podem facilmente chamar funções desta API, tanto porque estas foram escritas em C, como porque é fornecido um conjunto de funções C intermediárias que chamam funções escritas em *assembler* ou outra linguagem [3].

Apesar de OpenGL ser uma biblioteca de programação padrão, existem muitas implementações desta biblioteca, como por exemplo para *Windows* e para *Linux*. A implementação utilizada no ambiente *Linux* é a biblioteca Mesa, que está disponível em http://www.mesa3d.org/. Também existem implementações para os compiladores *Borland C++*, *Dev-C++*, *Delphi* e *Visual Basic*.

Para obter as bibliotecas e a documentação de cada implementação acesse http://www.opengl.org/.

No caso da implementação da *Microsoft*, o sistema operacional fornece os arquivos e *opengl32.dll* e *glu32.dll*, necessários para execução de programas OpenGL. Além disso, são fornecidas com suas ferramentas de programação, como por exemplo com o *Microsoft Visual C++*, as bibliotecas *opengl32.lib* (OpenGL) e *glu32.lib* (GLU - biblioteca de utilitários OpenGL). Assim, para criar programas que usem OpenGL é necessário adicionar estas duas bibliotecas à lista de bibliotecas importadas. Protótipos para todas as funções, tipos e macros OpenGL estão (por convenção) no *header gl.h*. Os protótipos da biblioteca de funções utilitárias estão no arquivo *glu.h*. Estes arquivos normalmente estão localizados em uma pasta especial no diretório do *include*. O código abaixo mostra as inclusões típicas para um programa *Windows* que usa OpenGL [3]:

```
#include <gl/gl.h>
#include <gl/glu.h>
```

Para utilizar a biblioteca GLUT (ver a seção 5), é necessário copiar os arquivos *glut.dll* e *glut32.dll* para a pasta *System* do *Windows* (ou para a pasta *System32* do *WindowsNT*/*Windows2000*); os arquivos *glut.lib* e *glut32.lib* para a mesma pasta do ambiente de programação onde estão as outras bibliotecas (*glu32.lib* e *opengl32.lib*); e o arquivo *glut.h* para a mesma pasta onde estão os arquivos *gl.h* e *glu.h* [3]. Estes arquivos normalmente estão comprimidos em um único arquivo que pode ser obtido em http://www.opengl.org/developers/documentation/glut/index.html.

Também é necessário criar um projeto para fazer o *link* das bibliotecas com a aplicação que será desenvolvida. No *Microsoft Visual C++* um projeto é criado da seguinte maneira: no menu *File* selecione a opção *New*; na janela aberta, selecione a opção *Win32ConsoleApplication* e coloque o nome do projeto e o local onde ele será gravado; no menu *Project* selecione a opção *Settings*; na janela aberta, selecione a guia *Link* e acrescente na caixa de texto *Object/library modules* a(s) biblioteca(s) *opengl32.lib*, *glu32.lib*, ou *glut32.lib*; no final, adicione um arquivo fonte (.cpp) ao projeto, selecionando a opção *Add to Project*/*Files* do menu *Project*.

03. TIPOS DE DADOS

Para tornar o código portável, foram definidos tipos de dados próprios para OpenGL. Estes tipos de dados são mapeados os tipos de dados C comuns, que também podem ser utilizados. Como os vários compiladores e ambientes possuem regras diferentes para determinar o tamanho das variáveis C, usando os tipos OpenGL é possível “isolar” o código das aplicações destas alterações.

Na tabela 3.1, definida por Woo et al. [2] e Wright e Sweet [3], são apresentados os tipos de dados OpenGL, os tipos de dados C correspondentes e o sufixo apropriado. Estes sufixos são usados para especificar os tipos de dados para as implementações ISO C de OpenGL. Pode-se observar que todos os tipos começam "GL", e a maioria é seguido pelo tipo de dado C correspondente.

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

Tabela 3.1 - Tipos de dados OpenGL

04. CONVENÇÕES PARA OS NOMES DAS FUNÇÕES

Todos os nomes das funções OpenGL seguem uma convenção que indica de qual biblioteca a função faz parte e, freqüentemente, quantos e que tipos de argumentos a função tem. Todas as funções possuem uma raiz que representa os comandos OpenGL que correspondem às funções. Por exemplo, a função *glColor3f* possui *Color* como raiz. O prefixo *gl* representa a biblioteca *gl*, e o sufixo *3f* significa que a função possui três valores de ponto flutuante como parâmetro. Variações desta função podem receber três valores inteiros como parâmetro (*glColor3i*), três *doubles* (*glColor3d*) e assim por diante. Resumindo, todas as funções OpenGL possuem o seguinte formato: <PrefixoBiblioteca> <ComandoRaiz> <ContadorArgsOpcional> <TipoArgsOpcional>.

05. BIBLIOTECAS

Segundo Woo et al. [2], OpenGL fornece um conjunto de comandos poderosos, mas primitivos. Portanto, todas as rotinas de desenho de alto nível devem ser elaboradas em função destes comandos. Sendo assim, foram desenvolvidas algumas bibliotecas para simplificar a tarefa de programação. Estas bibliotecas são apresentadas a seguir.

- **GLU - *OpenGL Utility Library***: contém várias rotinas que utilizam os comandos OpenGL de baixo nível para executar tarefas como, por exemplo, definir as matrizes para projeção e orientação da visualização. Esta biblioteca é fornecida como parte de cada implementação de OpenGL, e suas funções usam o prefixo **glu** [2].
- **GLUT - *OpenGL Utility Toolkit***: é um *toolkit* independente de plataforma, que inclui alguns elementos GUI (*Graphical User Interface*), tais como menus *pop-up* e suporte para *joystick*. O seu principal objetivo é esconder a complexidade das APIs dos diferentes sistemas de janelas. As funções desta biblioteca usam o prefixo **glut**. É interessante comentar que a GLUT substituiu a GLAUX, uma biblioteca auxiliar OpenGL que havia sido criada para facilitar o aprendizado e a elaboração de programas OpenGL independente do ambiente de programação [2, 3]. Informações sobre a GLUT podem ser obtidas em http://www.opengl.org/developers/documentation/glut.html.
- **GLX - *OpenGL Extension to the X Window System***: fornecido como um "anexo" de OpenGL para máquinas que usam o *X Window System*. Funções GLX usam o prefixo **glX**. Para *Microsoft Windows* 95/98/NT, as funções WGL fornecem as janelas para a interface OpenGL. Todas as funções WGL usam o prefixo **wgl**. Para IBM/OS2, a PGL é a *Presentation Manager* para a interface OpenGL, e suas funções usam o prefixo **pgl**. Para *Apple*, a AGL é a interface para sistemas que suportam OpenGL, e as funções AGL usam o prefixo **agl** [2].
- **FSG - *Fahrenheit Scene Graph***: é um *toolkit* orientado à objetos e baseado em OpenGL, que fornece objetos e métodos para a criação de aplicações gráficas 3D interativas. FSG, que foi escrito em C++ e é separado de OpenGL, fornece componentes de alto nível para criação e edição de cenas 3D, e a habilidade de trocar dados em outros formatos gráficos [2].

06. MÁQUINA DE ESTADOS

OpenGL é uma máquina de estados, conforme descrito por Woo et al. [2]. Isto significa que é possível colocá-la em vários estados (ou modos) que não são alterados, a menos que uma função seja chamada para isto. Por exemplo, a cor corrente é uma variável de estado que pode ser definida como branco. Todos os objetos, então, são desenhados com a cor branca, até o momento em que outra cor corrente é definida.

OpenGL mantém uma série de variáveis de estado, tais como estilo de uma linha, posições e características das luzes, e propriedades do material dos objetos que estão sendo desenhados. Muitas delas referem-se a modos que podem ser habilitados ou desabilitados com os comandos *glEnable()* e *glDisable()*.

Cada variável de estado possui um valor inicial que pode ser alterado. As funções que podem ser utilizadas para saber o seu valor são: *glGetBooleanv()*, *glGetDoublev()*, *glGetFloatv*, *glGetIntegerv*, *glGetPointerv()* ou *glIsEnabled()*. Dependendo do tipo de dado, é possível saber qual destas funções deve ser usada [2]. O trecho de programa a seguir mostra um exemplo da utilização destas funções.

```
int luz;
:
glEnable(GL_LIGHTING);    // Habilita luz - GL_LIGHTING é a variável de estado
:
luz = glIsEnabled(GL_LIGHTING);   // retorna 1 (verdadeiro)
:
glDisable(GL_LIGHTING);   // Desabilita luz
:
luz = glIsEnabled(GL_LIGHTING);   // retorna 0 (falso)
:
```

07. PRIMEIRO PROGRAMA

A biblioteca GLUT, descrita na seção 5, é utilizada nos exemplos deste artigo que serão apresentados a partir desta seção. Portanto, estes exemplos podem ser compilados em várias plataformas. Para entender o funcionamento da GLUT, logo abaixo é apresentado o menor programa OpenGL possível, definido por Wright e Sweet [3].

```
// PrimeiroPrograma.c - Isabel H. Manssour
// Um programa OpenGL simples que abre uma janela GLUT
// Este código está baseado no Simple.c, exemplo disponível no livro
// "OpenGL SuperBible", 2nd Edition, de Richard S. e Wright Jr.

#include <gl/glut.h>

// Função callback chamada para fazer o desenho
void Desenha(void)
{
        // Limpa a janela de visualização com a cor de fundo especificada
        glClear(GL_COLOR_BUFFER_BIT);

        // Executa os comandos OpenGL
        glFlush();
}

// Inicializa parâmetros de rendering
void Inicializa (void)
{
       // Define a cor de fundo da janela de visualização como preta
       glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
}

// Programa Principal
void main(void)
{
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutCreateWindow("Primeiro Programa");
        glutDisplayFunc(Desenha);
        Inicializa();
        glutMainLoop();
}
```

Este programa simples contém quatro funções da biblioteca GLUT (prefixo *glut*), e três funções OpenGL (prefixo *gl*). O conteúdo deste programa é descrito detalhadamente a seguir.

- O arquivo ***glut.h*** contém os protótipos das funções utilizadas pelo programa. Ele também inclui os *headers gl.h* e *glu.h* que definem, respectivamente, as bibliotecas de funções OpenGL e GLU. O comando *#include <windows.h>* também pode ser requerido por aplicações *Windows*. Este *header* não foi incluído aqui, porque sua inclusão é opcional com a versão WIN32 da GLUT, uma vez que a *glut.h* já inclui o *windows.h*. Entretanto, se o objetivo é criar um código portável, é um bom hábito incluir este arquivo.
- ***glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);*** avisa a GLUT que tipo de modo de exibição deve ser usado quando a janela é criada. Neste caso os argumentos indicam a criação de uma janela *single-buffered* (GLUT_SINGLE) com o modo de cores RGBA (GLUT_RGB). O primeiro significa que todos os comandos de desenho são feitos na janela de visualização. Uma alternativa é uma janela *double-buffered*, onde os comandos de desenho são executados para criar uma cena fora da tela para depois rapidamente colocá-la na janela de visualização. Este método é geralmente utilizado para produzir efeitos de animação. O modo de cores RGBA significa que as cores são especificadas através do fornecimento de intensidades dos componentes *red*, *green* e *blue* separadas.
- ***glutCreateWindow("Primeiro Programa");*** é o comando da biblioteca GLUT que cria a janela. Neste caso, é criada uma janela com o nome "Primeiro Programa". Este argumento corresponde à legenda para a barra de título da janela.
- ***glutDisplayFunc(Desenha);*** estabelece a função "Desenha" previamente definida como a função *callback* de exibição. Isto significa que a GLUT chama esta função sempre que a janela precisa ser redesenhada. Esta chamada ocorre, por exemplo, quando a janela é redimensionada ou encoberta. É nesta função que devem ser colocadas as chamadas de funções OpenGL, por exemplo, para modelar e exibir um objeto.
- ***Inicializa();*** não é uma função OpenGL nem GLUT, é apenas uma convenção utilizada nos exemplos apresentados por Wright e Sweet [3], nos quais este artigo está baseado. Nesta função são feitas as inicializações OpenGL que devem ser executadas antes do *rendering*. Muitos estados OpenGL devem ser determinados somente uma vez e não a cada vez que a função "Desenha" é chamada.
- ***glutMainLoop();*** é a função que faz com que comece a execução da "máquina de estados" e processa todas as mensagens específicas do sistema operacional, tais como teclas e botões do *mouse* pressionados, até que o programa termine.
- ***glClearColor(0.0f, 0.0f, 0.0f, 1.0f);*** é a função que determina a cor utilizada para limpar a janela. Seu protótipo é: *void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alfa);*. *GLclampf* é definido como um *float* na maioria das implementações de OpenGL. O intervalo para cada componente *red*, *green*, *blue* é de 0 a 1. O componente *alfa* é usado para efeitos especiais, tal como transparência.
- ***glClear(GL_COLOR_BUFFER_BIT);*** "limpa" um *buffer* particular ou combinações de *buffers*, onde *buffer* é uma área de armazenamento para informações da imagem. Os componentes RGB são geralmente referenciados como *color buffer* ou *pixel buffer*. Existem vários tipos de *buffer*, mas por enquanto só é necessário entender que o *color buffer* é onde a imagem é armazenada internamente e limpar o *buffer* com *glClear* remove o desenho da janela.
- ***glFlush();*** faz com que qualquer comando OpenGL não executado seja executado. Neste primeiro exemplo tem apenas a função *glClear* [3].

08. DESENHANDO PRIMITIVAS

Nesta seção é apresentado um exemplo que mostra como fazer um desenho, mover e redimensionar a janela [3].

```
// Quadrado.c - Isabel H. Manssour
// Um programa OpenGL simples que desenha um quadrado em uma janela GLUT.
// Este código está baseado no GLRect.c, exemplo disponível no livro
// "OpenGL SuperBible", 2nd Edition, de Richard S. e Wright Jr.

#include <gl/glut.h>

// Função callback chamada para fazer o desenho
void Desenha(void)
{
     glMatrixMode(GL_MODELVIEW);
     glLoadIdentity();

     // Limpa a janela de visualização com a cor de fundo especificada
     glClear(GL_COLOR_BUFFER_BIT);

     // Especifica que a cor corrente é vermelha
     //         R     G     B
     glColor3f(1.0f, 0.0f, 0.0f);

     // Desenha um quadrado preenchido com a cor corrente
     glBegin(GL_QUADS);
               glVertex2i(100,150);
               glVertex2i(100,100);
               // Especifica que a cor corrente é azul
               glColor3f(0.0f, 0.0f, 1.0f);
               glVertex2i(150,100);
               glVertex2i(150,150);
     glEnd();

     // Executa os comandos OpenGL
     glFlush();
}

// Inicializa parâmetros de rendering
void Inicializa (void)
{
    // Define a cor de fundo da janela de visualização como preta
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
}

// Função callback chamada quando o tamanho da janela é alterado
void AlteraTamanhoJanela(GLsizei w, GLsizei h)
{
    // Evita a divisao por zero
    if(h == 0) h = 1;

    // Especifica as dimensões da Viewport
    glViewport(0, 0, w, h);

    // Inicializa o sistema de coordenadas
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Estabelece a janela de seleção (left, right, bottom, top)
    if (w <= h)
        gluOrtho2D (0.0f, 250.0f, 0.0f, 250.0f*h/w);
    else
        gluOrtho2D (0.0f, 250.0f*w/h, 0.0f, 250.0f);
}

// Programa Principal
void main(void)
{
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
     glutInitWindowSize(400,350);
     glutInitWindowPosition(10,10);
     glutCreateWindow("Quadrado");
```

```
    glutDisplayFunc(Desenha);
    glutReshapeFunc(AlteraTamanhoJanela);
    Inicializa();
    glutMainLoop();
}
```

Este programa, como mostra a figura 8.1, apenas exibe um quadrado azule vermelho numa janela com fundo preto. As novas funções utilizadas neste exemplo são descritas a seguir.

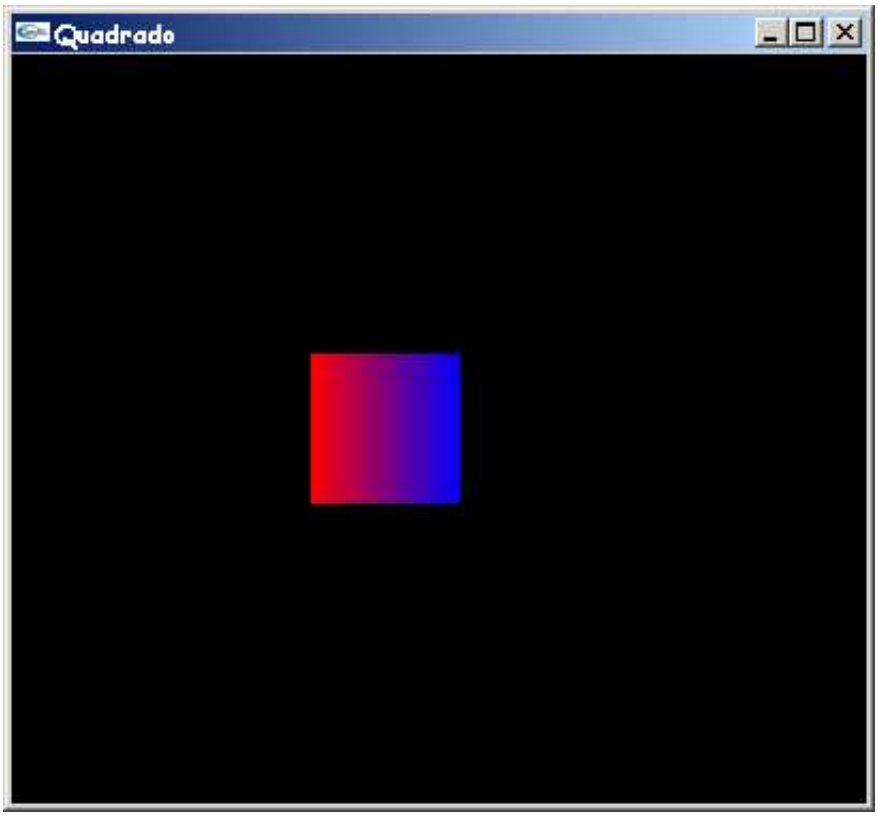

Figura 8.1 - Saída do programa Quadrado.c

- ***glutInitWindowSize(400,350);*** especifica o tamanho em *pixels* da janela GLUT.
- ***glutInitWindowPosition(10,10);*** especifica a localização inicial da janela GLUT, que neste caso é o canto superior esquerdo da tela do computador [2].
- ***glutReshapeFunc(AlteraTamanhoJanela);*** estabelece a função "AlteraTamanhoJanela" previamente definida como a função *callback* de alteração do tamanho da janela. Isto é, sempre que a janela é maximizada, minimizada, etc., a função "AlteraTamanhoJanela" é executada para reinicializar o sistema de coordenadas.
- ***glColor3f(1.0f, 0.0f, 0.0f);*** determina a cor que será usada para o desenho (linhas e preenchimento). A seleção da cor é feita da mesma maneira que na função *glClearColor*, sendo que não é necessário especificar o componente *alfa*, cujo valor *default* é 1.0 (completamente opaco).
- ***glBegin(GL_QUADS);… glEnd();*** usada para desenhar um quadrado preenchido a partir dos vértices especificados entre *glBegin* e *glEnd*. A seção 9 descreve a utilização destas funções. OpenGL mapeia as coordenadas dos vértices para a posição atual da janela de visualização na função *callback* "AlteraTamanhoJanela".

Antes de descrever os parâmetros e comandos da função "AlteraTamanhoJanela", é necessário revisar alguns conceitos e especificações. Em quase todos ambientes de janelas, o usuário pode alterar o tamanho e dimensões da janela em qualquer momento. Quando isto ocorre o conteúdo da janela é redesenhado levando em conta as novas dimensões. Normalmente o esperado é que a escala do desenho seja alterada de maneira que ele fique dentro da janela, independente do

tamanho da janela de visualização ou do desenho [4]. Assim, uma janela pequena terá o desenho completo, mas pequeno, e uma janela grande terá o desenho completo e maior.

No exemplo, o desenho do quadrado 2D é feito no plano xy, com z=0, sendo necessário determinar o tamanho da *viewport* (janela de visualização) e da janela de seleção, pois estes parâmetros influenciam o espaço de coordenadas e a escala dos desenhos 2D e 3D na janela.

Sempre que o tamanho da janela é alterado, a *viewport* e a janela de seleção devem ser redefinidas de acordo com as novas dimensões da janela. Assim, a aparência do desenho não é alterada (por exemplo, um quadrado não vira um retângulo). Como a alteração do tamanho da janela é detectada e gerenciada de maneira diferente em cada ambiente, a biblioteca GLUT fornece a função *glutReshapeFunc*, descrita anteriormente, que registra a função *callback* que a GLUT irá chamar sempre que houver esta alteração. A função passada para a *glutReshapeFunc* deve ter o seguinte protótipo: *void AlteraTamanhoJanela(GLsizei w, GLsizei h);*. O nome "AlteraTamanhoJanela" foi escolhido porque descreve o que a função faz. Os parâmetros recebidos sempre que o tamanho da janela é alterado são a sua nova largura e a sua nova altura, respectivamente. Esta informação é usada para modificar o mapeamento do sistema de coordenadas desejado para o sistema de coordenadas da tela com a ajuda de duas funções, uma OpenGL, *glViewport*, e uma da biblioteca GLU, *gluOrtho2D*. Estas e outras funções chamadas na "AlteraTamanhoJanela", que definem como a *viewport* é especificada, são descritas a seguir.

- ***glViewport(0, 0, w, h);*** recebe como parâmetro a nova largura e altura da janela. O protótipo desta função é: *void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);*. Seus parâmetros especificam o canto inferior esquerdo da *viewport* (x,y) dentro da janela, e a sua largura e altura em *pixels* (*width* e *height*). Geralmente x e y são zero, mas é possível usar a *viewport* para visualizar mais de uma cena em diferentes áreas da janela. Em outras palavras, a *viewport* define a área dentro janela, em coordenadas de tela, que OpenGL pode usar para fazer o desenho. A janela de seleção é, então, mapeada para a nova *viewport*.
- ***gluOrtho2D (0.0f, 250.0f*w/h, 0.0f, 250.0f);*** é usada para projetar na tela uma imagem 2D que está na janela de seleção definida através dos parâmetros passados para esta função. O protótipo desta função é: *void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);*. No sistema de coordenadas cartesianas, os valores *left* e *right* especificam os limites mínimo e máximo no eixo X; analogamente, *bottom* e *top* especificam os limites mínimo e máximo no eixo Y.
- ***glMatrixMode(GL_PROJECTION);*** e ***glLoadIdentity();*** servem, respectivamente, para avisar a OpenGL que todas as futuras alterações, tais como operações de escala, rotação e translação, irão afetar a "câmera" (ou observador), e para inicializar o sistema de coordenadas antes da execução de qualquer operação de manipulação de matrizes. Sem este comando, cada chamada sucessiva de *gluOrtho2D* poderia resultar em uma corrupção da janela de seleção. Em outras palavras, a matriz de projeção é onde o volume de visualização, que neste caso é um plano, é definido; a função *gluOrtho2D* não estabelece realmente o volume de visualização utilizado para fazer o recorte, apenas modifica o volume existente; ela multiplica a matriz que descreve o volume de visualização corrente pela matriz que descreve o novo volume de visualização, cujas coordenadas são recebidas por parâmetro.
- ***glMatrixMode(GL_MODELVIEW);*** avisa a OpenGL que todas as futuras alterações, tais como operações de escala, rotação e translação, irão afetar os modelos da cena, ou em outras palavras, o que é desenhado. A função *glLoadIdentity();* chamada em seguida, faz com que a matriz corrente seja inicializada com a matriz identidade (nenhuma transformação é acumulada) [3].

09. LINHAS, PONTOS E POLÍGONOS

Com apenas algumas primitivas simples, tais como pontos, linhas e polígonos, é possível criar estruturas complexas. Em outras palavras, objetos e cenas criadas com OpenGL consistem em simples primitivas gráficas que podem ser combinadas de várias maneiras [3]. Portanto, OpenGL fornece ferramentas para desenhar pontos, linhas e polígonos, que são formados por um ou mais vértices. Neste caso, é necessário passar uma lista de vértices, o que pode ser feito entre duas chamadas de funções OpenGL: *glBegin()* e *glEnd()*. O argumento passado para *glBegin()* determina qual objeto será desenhado. No exemplo fornecido por Hill [1], para desenhar três pontos pretos foi usada a seguinte seqüência de comandos:

```
glBegin(GL_POINTS);
        glColor3f(0.0f, 0.0f, 0.0f);
        glVertex2i(100, 50);
        glVertex2i(100, 130);
        glVertex2i(150, 130);
glEnd();
```

Para desenhar outras primitivas, basta trocar *GL_POINTS*, que exibe um ponto para cada chamada ao comando *glVertex*, por [1]:

- GL_LINES: exibe uma linha a cada dois comandos *glVertex*;
- GL_LINE_STRIP: exibe uma seqüência de linhas conectando os pontos definidos por *glVertex*;
- GL_LINE_LOOP: exibe uma seqüência de linhas conectando os pontos definidos por *glVertex* e ao final liga o primeiro como último ponto;
- GL_POLYGON: exibe um polígono convexo preenchido, definido por uma seqüência de chamadas a *glVertex*;
- GL_TRIANGLES: exibe um triângulo preenchido a cada três pontos definidos por *glVertex*;
- GL_TRIANGLE_STRIP: exibe uma seqüência de triângulos baseados no trio de vértices v0, v1, v2, depois, v2, v1, v3, depois, v2, v3, v4 e assim por diante;
- GL_TRIANGLE_FAN: exibe uma seqüência de triângulos conectados baseados no trio de vértices v0, v1, v2, depois, v0, v2, v3, depois, v0, v3, v4 e assim por diante;
- GL_QUADS: exibe um quadrado preenchido conectando cada quatro pontos definidos por *glVertex*;
- GL_QUAD_STRIP: exibe uma seqüência de quadriláteros conectados a cada quatro vértices; primeiro v0, v1, v3, v2, depois, v2, v3, v5, v4, depois, v4, v5, v7, v6, e assim por diante.

A função *glVertex2i* pertence à biblioteca GL e possui dois argumentos inteiros. De maneira análoga, também é possível passar valores de ponto flutuante no lugar de inteiros, e três coordenadas (x,y,z) no lugar de duas usando, por exemplo, as seguintes chamadas às funções OpenGL: *glVertex2d(100.0, 50.0);* e *glVertex3f(50.0, 50.0, 50.0);* Além disso, para cada vértice é possível definir uma cor diferente. Neste caso, no desenho final é feita uma "interpolação" das cores, como mostra o exemplo da figura 8.1.

Para ilustrar a diferença na utilização de algumas primitivas gráficas o código apresentado na seção 8 foi alterado da seguinte maneira: inicialmente, os parâmetros passados para a função *glutInitWindowSize* foram alterados para *(200,200)*, para diminuir o tamanho da janela GLUT; depois, antes da função *glBegin(GL_QUADS)* foram chamadas as funções *glPointSize(3)* e *glLineWidth(3)*; finalmente, *GL_QUADS*, foi substituído por *GL_POINTS*, *GL_LINES*, *GL_LINE_STRIP* e *GL_LINE_LOOP*, gerando as imagens apresentadas na figura 9.1.

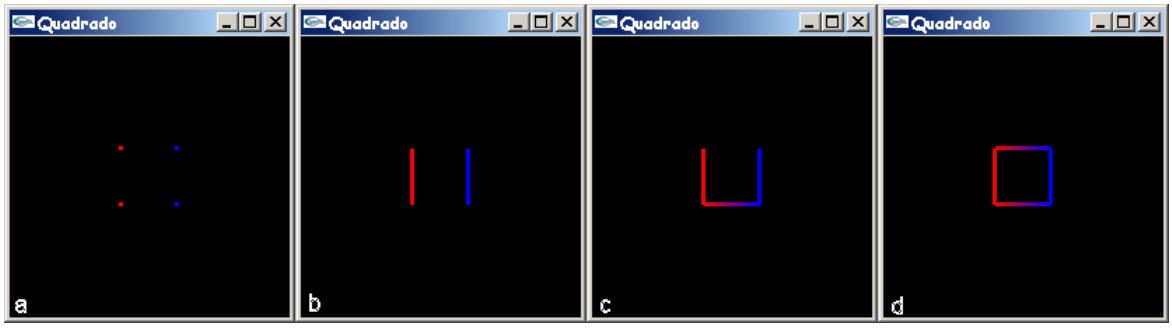

Figura 9.1 – Imagens geradas com a utilização de (a) GL_POINTS, (b) GL_LINES, (c) GL_LINE_STRIP e (d) GL_LINE_LOOP

10. TRANSFORMAÇÕES GEOMÉTRICAS

As transformações geométricas são usadas para manipular um modelo, isto é, através delas é possível mover, rotacionar ou alterar a escala de um objeto. A aparência final da cena ou do objeto depende muito da ordem na qual estas transformações são aplicadas.

A biblioteca gráfica OpenGL é capaz de executar transformações de translação, escala e rotação através de uma multiplicação de matrizes. A idéia central destas transformações em OpenGL é que elas podem ser combinadas em uma única matriz, de tal maneira que várias transformações geométricas possam ser aplicadas através de uma única operação. Isto ocorre porque uma transformação geométrica em OpenGL é armazenada internamente em uma matriz. A cada nova transformação que é aplicada esta matriz é alterada e usada para desenhar os objetos a partir daquele momento. A cada nova alteração é feita uma composição de matrizes. Para evitar este efeito "cumulativo", é necessário utilizar as funções *glPushMatrix()* e *glPopMatrix()*, que salvam e restauram, respectivamente, a matriz atual em uma pilha interna da OpenGL.

A **translação** é feita através da função *glTranslatef(Tx, Ty, Tz)*, que pode receber três números *float* ou *double* (*glTranslated*) como parâmetro. Neste caso, a matriz atual é multiplicada por uma matriz de translação baseada nos valores dados.

A **rotação** é feita através da função *glRotatef(Ângulo, x, y, z)*, que pode receber quatro números *float* ou *double* (*glRotated*) como parâmetro. Neste caso, a matriz atual é multiplicada por uma matriz de rotação de "Ângulo" graus ao redor do eixo definido pelo vetor "x,y,z" no sentido anti-horário.

A **escala** é feita através da função *glScalef(Ex, Ey, Ez)*, que pode receber três números *float* ou *double* (*glScaled*) como parâmetro. Neste caso, a matriz atual é multiplicada por uma matriz de escala baseada nos valores dados.

Logo abaixo a função "Desenha" do exemplo da seção 8 é alterada para incluir as transformações geométricas de translação e escala. A figura 10.1 mostra como o quadrado é exibido após a aplicação das transformações especificadas.

```
void Desenha(void)
{
     glMatrixMode(GL_MODELVIEW);
     glLoadIdentity();
     glClear(GL_COLOR_BUFFER_BIT);
     glColor3f(1.0f, 0.0f, 0.0f);
     glTranslatef(-100.0f, -30.0f, 0.0f);
     glScalef(1.5f, 0.5f, 1.0f);
```

```
    glBegin(GL_QUADS);
              glVertex2i(100,150);
              glVertex2i(100,100);
              // Especifica que a cor corrente é azul
              glColor3f(0.0f, 0.0f, 1.0f);
              glVertex2i(150,100);
              glVertex2i(150,150);
    glEnd();
    glFlush();
}
```

Figura 10.1 – Saída do programa Quadrado.c com a utilização de transformações geométricas

11. GERENCIAMENTO DE EVENTOS

A biblioteca GLUT também contém funções para gerenciar eventos de entrada de teclado e *mouse*. De acordo com Woo et al. [2] e Wright e Sweet [3], estas funções foram descritas a seguir.

- ***glutKeyboardFunc*** - Estabelece a função *callback* que é chamada pela GLUT cada vez que uma tecla que gera código ASCII é pressionada (por exemplo: a, b, A, b, 1, 2). Além do valor ASCII da tecla, a posição (x,y) do *mouse* quando a tecla foi pressionada também é retornada. Parâmetros de entrada da função *callback*: ***(unsigned char key, int x, int y)***
- ***glutSpecialFunc*** - Estabelece a função *callback* que é chamada pela GLUT cada vez que uma tecla que gera código não-ASCII é pressionada, tais como *Home*, *End*, *PgUp*, *PgDn*, *F1* e *F2*. Além da constante que identifica a tecla, a posição corrente (x,y) do *mouse* quando a tecla foi pressionada também é retornada. Parâmetros de entrada da função *callback*: ***(unsigned char key, int x, int y)***. Os valores válidos para o primeiro parâmetro são: GLUT_KEY_F1, GLUT_KEY_F2, GLUT_KEY_F3, GLUT_KEY_F4, GLUT_KEY_F5, GLUT_KEY_F6, GLUT_KEY_F7, GLUT_KEY_F8, GLUT_KEY_F9, GLUT_KEY_F10, GLUT_KEY_F11, GLUT_KEY_F12, GLUT_KEY_LEFT,

GLUT_KEY_UP, GLUT_KEY_RIGHT, GLUT_KEY_DOWN, GLUT_KEY_HOME, GLUT_KEY_END, GLUT_KEY_PAGE_UP, GLUT_KEY_PAGE_DOWN, GLUT_KEY_INSERT.

- ***glutMouseFunc*** - Estabelece a função *callback* que é chamada pela GLUT cada vez que ocorre um evento de *mouse*. Parâmetros de entrada da função *callback*: ***(int button, int state, int x, int y)***. Três valores são válidos para o parâmetro *button*: GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON e GLUT_RIGHT_BUTTON. O parâmetro *state* pode ser GLUT_UP ou GLUT_DOWN. Os parâmetros x e y indicam a localização do mouse no momento que o evento ocorreu.

O programa abaixo exemplifica a utilização das funções acima especificadas. Neste exemplo, foram implementadas as seguintes interações: sempre que o usuário pressiona a tecla 'R' ou 'r', o retângulo é exibido com a cor vermelha; ao pressionar a tecla 'G' ou 'g' o retângulo é exibido com a cor verde; ao pressionar a tecla 'B' ou 'b' o retângulo é exibido com a cor azul; cada vez que o usuário clica com o botão esquerdo do mouse, o tamanho do retângulo é alterado (vai do centro da janela até a posição onde houve o clique do mouse); e cada vez que as teclas *KEY_UP* e *KEY_DOWN* são pressionadas ocorre *zoom-in* e *zoom-out*, respectivamente.

```
// Interacao.c - Isabel H. Manssour
// Um programa OpenGL simples que desenha um  quadrado em
// uma janela GLUT de acordo com interações do usuário.
// Este código está baseado nos exemplos disponíveis no livro
// "OpenGL SuperBible", 2nd Edition, de Richard S. e Wright Jr.

#include <gl/glut.h>

GLfloat xf, yf, win;
GLint view_w, view_h;

// Função callback chamada para fazer o desenho
void Desenha(void)
{
     glMatrixMode(GL_MODELVIEW);
     glLoadIdentity();

     glClear(GL_COLOR_BUFFER_BIT);
     // Desenha um retângulo preenchido com a cor corrente
     glBegin(GL_POLYGON);
               glVertex2f(0.0f, 0.0f);
               glVertex2f(xf, 0.0f);
               glVertex2f(xf, yf);
               glVertex2f(0.0f, yf);
     glEnd();
     glFlush();
}

// Inicializa parâmetros de rendering
void Inicializa (void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    xf=50.0f;
    yf=50.0f;
    win=250.0f;
}

// Função callback chamada quando o tamanho da janela é alterado
void AlteraTamanhoJanela(GLsizei w, GLsizei h)
{
    // Especifica as dimensões da Viewport
    glViewport(0, 0, w, h);
    view_w = w;
    view_h = h;
```

```
    // Inicializa o sistema de coordenadas
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D (-win, win, -win, win);
}

// Função callback chamada para gerenciar eventos de teclado
void GerenciaTeclado(unsigned char key, int x, int y)
{
    switch (key) {
            case 'R':
            case 'r':// muda a cor corrente para vermelho
                    glColor3f(1.0f, 0.0f, 0.0f);
                    break;
            case 'G':
            case 'g':// muda a cor corrente para verde
                    glColor3f(0.0f, 1.0f, 0.0f);
                    break;
            case 'B':
            case 'b':// muda a cor corrente para azul
                    glColor3f(0.0f, 0.0f, 1.0f);
                    break;
    }
    glutPostRedisplay();
}

// Função callback chamada para gerenciar eventos do mouse
void GerenciaMouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON)
         if (state == GLUT_DOWN) {
                 // Troca o tamanho do retângulo, que vai do centro da
                 // janela até a posição onde o usuário clicou com o mouse
                 xf = ( (2 * win * x) / view_w) - win;
                 yf = ( ( (2 * win) * (y-view_h) ) / -view_h) - win;
         }
    glutPostRedisplay();
}

// Função callback chamada para gerenciar eventos do teclado
// para teclas especiais, tais como F1, PgDn e Home
void TeclasEspeciais(int key, int x, int y)
{
    if(key == GLUT_KEY_UP) {
           win -= 20;
           glMatrixMode(GL_PROJECTION);
           glLoadIdentity();
           gluOrtho2D (-win, win, -win, win);
    }
    if(key == GLUT_KEY_DOWN) {
           win += 20;
           glMatrixMode(GL_PROJECTION);
           glLoadIdentity();
           gluOrtho2D (-win, win, -win, win);
    }
    glutPostRedisplay();
}

// Programa Principal
void main(void)
{
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
     glutInitWindowSize(350,300);
     glutInitWindowPosition(10,10);
     glutCreateWindow("Exemplo de Interacao");
     glutDisplayFunc(Desenha);
     glutReshapeFunc(AlteraTamanhoJanela);
     glutKeyboardFunc(GerenciaTeclado);
```

```
    glutMouseFunc(GerenciaMouse);
    glutSpecialFunc(TeclasEspeciais);
    Inicializa();
    glutMainLoop();
}
```

12. PROGRAMANDO EM 3D

Os exemplos apresentados até aqui mostraram apenas desenhos 2D. Esta seção descreve como trabalhar em 3D usando OpenGL. As bibliotecas GLU e GLUT possuem uma série de funções para desenhar primitivas 3D, tais como esferas, cones, cilindros e *teapot*. Em 3D se assume que o processo utilizado para visualizar uma determinada cena é análogo a tirar uma fotografia com uma máquina fotográfica, o que inclui, segundo Woo et al. [2], os seguintes passos:

- Arrumar o tripé e posicionar a câmera para fotografar a cena - equivalente a especificar as transformações de visualização (veja a função *gluLookAt* descrita mais abaixo);
- Arrumar a cena para ser fotografada, incluindo ou excluindo objetos/pessoas - equivalente à etapa de modelagem (inclui as transformações geométricas, *glTranslatef*, *glScalef*, *glRotatef*, e o desenho dos objetos);
- Escolher a lente da câmera ou ajustar o *zoom* - equivalente a especificar as transformações de projeção (veja a função *gluPerspective* descrita mais abaixo);
- Determinar o tamanho final da foto (maior ou menor) - equivalente a especificar a *viewport* (funções *glViewport* e *ChangeSize*).

O exemplo abaixo exemplifica a utilização das funções OpenGL para visualização 3D.

```
// TeaPot3D.c - Isabel H. Manssour
// Um programa OpenGL que exemplifica a visualização de objetos 3D.
// Este código está baseado nos exemplos disponíveis no livro
// "OpenGL SuperBible", 2nd Edition, de Richard S. e Wright Jr.

#include <gl/glut.h>

GLfloat angle, fAspect;

// Função callback chamada para fazer o desenho
void Desenha(void)
{
        glClear(GL_COLOR_BUFFER_BIT);

        glColor3f(0.0f, 0.0f, 1.0f);

        // Desenha o teapot com a cor corrente (wire-frame)
        glutWireTeapot(50.0f);

        // Executa os comandos OpenGL
        glutSwapBuffers();
 }

// Inicializa parâmetros de rendering
void Inicializa (void)
{
       glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
       angle=45;
}

// Função usada para especificar o volume de visualização
void EspecificaParametrosVisualizacao(void)
{
        // Especifica sistema de coordenadas de projeção
        glMatrixMode(GL_PROJECTION);
        // Inicializa sistema de coordenadas de projeção
```

```
        glLoadIdentity();

        // Especifica a projeção perspectiva
         gluPerspective(angle,fAspect,0.1,500);

        // Especifica sistema de coordenadas do modelo
        glMatrixMode(GL_MODELVIEW);
        // Inicializa sistema de coordenadas do modelo
        glLoadIdentity();

        // Especifica posição do observador e do alvo
         gluLookAt(0,80,200, 0,0,0, 0,1,0);
}

// Função callback chamada quando o tamanho da janela é alterado
void AlteraTamanhoJanela(GLsizei w, GLsizei h)
{
        // Para previnir uma divisão por zero
        if ( h == 0 ) h = 1;

        // Especifica o tamanho da viewport
        glViewport(0, 0, w, h);

        // Calcula a correção de aspecto
        fAspect = (GLfloat)w/(GLfloat)h;

        EspecificaParametrosVisualizacao();
}

// Função callback chamada para gerenciar eventos do mouse
void GerenciaMouse(int button, int state, int x, int y)
{
        if (button == GLUT_LEFT_BUTTON)
                if (state == GLUT_DOWN) {  // Zoom-in
                        if (angle >= 10) angle -= 5;
                }
        if (button == GLUT_RIGHT_BUTTON)
                if (state == GLUT_DOWN) {  // Zoom-out
                        if (angle <= 130) angle += 5;
                }
        EspecificaParametrosVisualizacao();
        glutPostRedisplay();
}

// Programa Principal
void main(void)
{
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize(350,300);
        glutCreateWindow("Visualizacao 3D");
        glutDisplayFunc(Desenha);
         glutReshapeFunc(AlteraTamanhoJanela);
        glutMouseFunc(GerenciaMouse);
        Inicializa();
        glutMainLoop();
}
```

O programa apresentado acima, como mostra a figura 12.1, apenas exibe um *teapot* (ou bule de chá) azul, no formato *wire-frame*, numa janela com fundo preto. Ao posicionar o *mouse* sobre a janela e clicar com o botão esquerdo e direito, é possível obter *zoom-in* e *zoom-out*, respectivamente. As novas funções utilizadas neste exemplo são descritas a seguir.

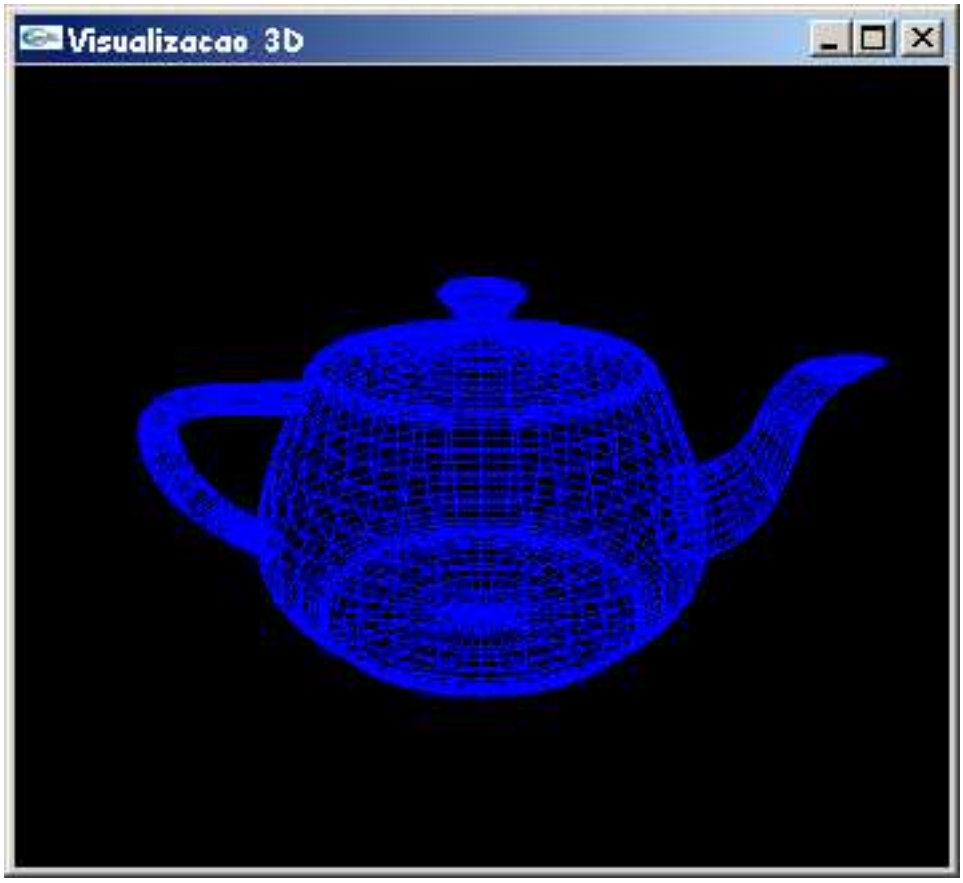

Figura 12.1 – Saída do programa TeaPot3D.c

- ***gluPerspective(angle,fAspect,0.1,500);*** segundo Wright e Sweet [3], esta função estabelece os parâmetros da Projeção Perspectiva, atualizando a matriz de projeção perspectiva. Seu protótipo é: *void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);*. Descrição dos parâmetros: *fovy* é o ângulo, em graus, na direção y (usada para determinar a "altura" do volume de visualização); *aspect* é a razão de aspecto que determina a área de visualização na direção x, e seu valor é a razão em x (largura) e y (altura); *zNear*, que sempre tem um valor positivo maior do que zero, é a distância do observador até o plano de corte mais próximo (em z); *zFar*, que também sempre tem um valor positivo maior do que zero, é a distância do observador até o plano de corte mais afastado (em z). Esta função sempre deve ser definida ANTES da função *gluLookAt*, e no modo GL_PROJECTION.
- ***gluLookAt(0,80,200, 0,0,0, 0,1,0);*** define a transformação de visualização. Através dos seus argumentos é possível indicar a posição da câmera e para onde ela está direcionada. Seu protótipo é: *void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz);*. Descrição dos parâmetros: *eyex*, *eyey* e *eyez* são usados para definir as coordenadas x, y e z, respectivamente, da posição da câmera (ou observador); *centerx*, *centery* e *centerz* são usados para definir, respectivamente, as coordenadas x, y e z da posição do alvo, isto é para onde o observador está olhando (normalmente, o centro da cena); *upx*, *upy* e *upz* são as coordenadas x, y e z, que estabelecem o vetor *up* (indica o "lado de cima" de uma cena 3D) [3].
- ***glutWireTeapot(50.0f);*** é usada para desenhar o *wire-frame* de um *teapot* (=bule de chá). Seu protótipo é: *glutWireTeapot(GLdouble size);*, onde o parâmetro *size* indica um raio aproximado do *teapot*. Uma esfera com este raio “envolve” totalmente o modelo [3].

Assim como o *teapot*, a biblioteca GLUT também possui as seguintes funções para desenhar outros objetos 3D [2]:

- *void glutWireCube(GLdouble size);*
- *void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);*
- *void glutWireCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks);*
- *void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);*
- *void glutWireIcosahedron(void);*
- *void glutWireOctahedron(void);*
- *void glutWireTetrahedron(void);*
- *void glutWireDodecahedron(GLdouble radius);*

Os parâmetros *slices* e *stacks* que aparecem no protótipo de algumas funções, significam, respectivamente, o número de subdivisões **em torno do eixo z** (como se fossem linhas longitudinais) e o número de subdivisões **ao longo do eixo z** (como se fossem linhas latitudinais). Já *rings* e *nsides* correspondem, respectivamente, ao número de seções que serão usadas para formar o *torus*, e ao número de subdivisões para cada seção. A figura 12.2 exibe um exemplo de um *torus* com *rings*=6 e *nsides*=20 (*glutWireTorus(10.0, 25, 20, 6);*), e a figura 12.3 exibe um exemplo com *rings*=20 e *nsides*=20 (*glutWireTorus(10.0, 25, 20, 20);*).

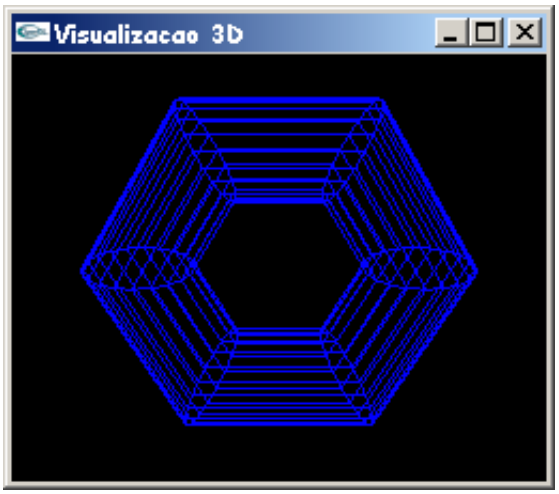

Figura 12.2 – *Torus* (*rings*=6, *nsides*=20)

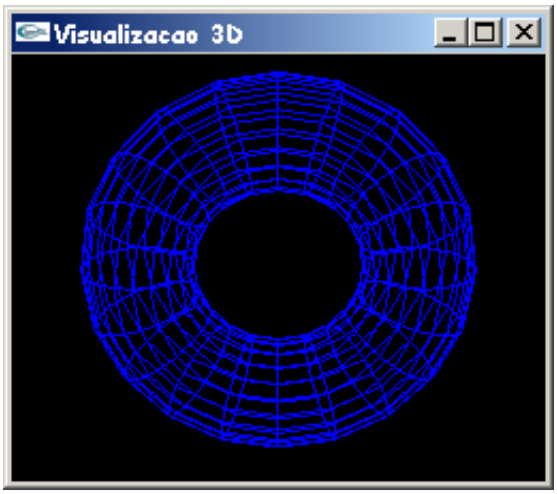

Figura 12.3 – *Torus* (*rings*=20, *nsides*=20)

Todas estas funções também podem ser usadas para desenhar objetos sólidos, ao invés de exibir apenas o seu *wire-frame*. Para isto, basta substituir a *substring Wire* do nome da função por *Solid*. Por exemplo, se substituirmos a chamada à função *glutWireTeapot(50.0f)* por *glutSolidTeapot(50.0f)* a imagem gerada é a apresentada na figura 12.4. Observando esta figura é possível observar que a imagem gerada parece 2D. Isto ocorre porque a versão *Solid* deve ser usada somente quando se está trabalhando com iluminação, conforme descreve a seção 13.

Figura 12.4 – *Teapot* gerado usando a função *glutSolidTeapot*

13. UTILIZANDO LUZES

Para adicionar um pouco de realismo às imagens 3D é necessário adicionar uma ou mais fontes de luz. Inicialmente, é preciso definir o modelo de colorização que será utilizado. OpenGL fornece dois modelos: um polígono preenchido pode ser desenhado com uma única cor (GL_FLAT), ou com uma variação de tonalidades (GL_SMOOTH, também chamado de modelo de colorização de *Gouraud* [4]). A função ***glShadeModel()*** é usada para especificar a técnica de colorização desejada.

Quando objetos 3D sólidos são exibidos, é importante desenhar os objetos que estão mais próximos da posição da câmera, eliminando objetos que ficam "escondidos", ou "parcialmente escondidos", por estes. Sendo assim, algoritmos de remoção de elementos escondidos foram desenvolvidos para determinar as linhas, arestas, superfícies ou volumes que são visíveis ou não para um observador localizado em um ponto específico no espaço.

Conforme descrito por Woo et al. [2], OpenGL possui um *depth buffer* que trabalha através da associação de uma profundidade, ou distância, do plano de visualização (geralmente o plano de corte mais próximo do observador) com cada *pixel* da *window*. Inicialmente, os valores de profundidade são especificados para serem o maior possível através do comando ***glClear(GL_DEPTH_BUFFER_BIT)***. Entretanto, habilitando o *depth-buffering* através dos comandos ***glutInitDisplayMode(GLUT_DEPTH | ...)*** e ***glEnable(GL_DEPTH_TEST)***, antes de cada *pixel* ser desenhado é feita uma comparação com o valor de profundidade já armazenado. Se o valor de profundidade for menor (está mais próximo) o *pixel* é desenhado e o valor de profundidade é atualizado. Caso contrário, as informações do *pixel* são desprezadas.

Em OpenGL a cor de uma fonte de luz é caracterizada pela quantidade de vermelho (R), verde (G) e azul (B) que ela emite, e o material de uma superfície é caracterizado pela porcentagem dos componentes R, G e B que chegam e são refletidos em várias direções. No modelo de iluminação a luz em uma cena vem de várias fontes de luz que podem ser "ligadas" ou "desligadas" individualmente. A luz pode vir de uma direção ou posição (por exemplo, uma lâmpada) ou como resultado de várias reflexões (luz ambiente - não é possível determinar de onde ela vem, mas ela desaparece quando a fonte de luz é desligada).

No modelo de iluminação OpenGL a fonte de luz tem efeito somente quando existem superfícies que absorvem e refletem luz. Assume-se que cada superfície é composta de um material com várias propriedades. O material pode emitir luz, refletir parte da incidente luz em todas as direções, ou refletir uma parte da luz incidente numa única direção, tal com um espelho. Então, conforme descrito por Woo et al. [2], OpenGL considera que a luz é dividida em quatro componentes independentes (que são colocadas juntas):

- Ambiente: resultado da luz refletida no ambiente; é uma luz que vem de todas as direções;
- Difusa: luz que vem de uma direção, atinge a superfície e é refletida em todas as direções; assim, parece possuir o mesmo brilho independente de onde a câmera está posicionada;
- Especular: luz que vem de uma direção e tende a ser refletida numa única direção;
- Emissiva: simula a luz que se origina de um objeto; a cor emissiva de uma superfície adiciona intensidade ao objeto, mas não é afetada por qualquer fonte de luz; ela também não introduz luz adicional da cena.

A cor do material de um objeto depende da porcentagem de luz vermelha, verde e azul incidente que ele reflete. Assim como as luzes, o material possui cor ambiente, difusa e especular diferentes, que determinam como será a luz refletida. Isto é combinado com as propriedades das fontes de luz, de tal maneira que a reflexão ambiente e difusa definem a cor do material. A especular é geralmente cinza ou branca.

Os componentes de cor especificados para a luz possuem um significado diferente dos componentes de cor especificados para os materiais. Para a luz, os números correspondem a uma porcentagem da intensidade total para cada cor. Se os valores R, G e B para a cor da luz são 1, a luz é branca com o maior brilho possível. Se os valores são 0.5 a cor ainda é branca, mas possui metade da intensidade, por isso parece cinza. Se R=G=1 e B=0, a luz parece amarela.

Para os materiais, os números correspondem às proporções refletidas destas cores. Se R=1, G=0.5 e B=0 para um material, este material reflete toda luz vermelha incidente, metade da luz verde e nada da luz azul. Assim, simplificadamente, a luz que chega no observador é dada por (LR.MR, LG.MG, LB.MB), onde (LR, LG, LB) são os componentes da luz e (MR, MG, MB) os componentes do material [2].

O programa abaixo apresenta um exemplo completo da utilização de luzes em OpenGL. A imagem gerada, como mostra a figura 13.1, apenas exibe um exibe um *teapot* azul iluminado, numa janela com fundo preto.

```
// Iluminacao.c - Isabel H. Manssour
// Um programa OpenGL que exemplifica a visualização
// de objetos 3D com a inserção de uma fonte de luz.
// Este código está baseado nos exemplos disponíveis no livro
// "OpenGL SuperBible", 2nd Edition, de Richard S. e Wright Jr.

#include <gl/glut.h>

GLfloat angle, fAspect;

// Função callback chamada para fazer o desenho
void Desenha(void)
{
        // Limpa a janela e o depth buffer
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glColor3f(0.0f, 0.0f, 1.0f);

        // Desenha o teapot com a cor corrente (solid)
        glutSolidTeapot(50.0f);

        glutSwapBuffers();
}
```

```
// Inicializa parâmetros de rendering
void Inicializa (void)
{
        GLfloat luzAmbiente[4]={0.2,0.2,0.2,1.0};
        GLfloat luzDifusa[4]={0.7,0.7,0.7,1.0};           // "cor"
        GLfloat luzEspecular[4]={1.0, 1.0, 1.0, 1.0};     // "brilho"
        GLfloat posicaoLuz[4]={0.0, 50.0, 50.0, 1.0};

        // Capacidade de brilho do material
        GLfloat especularidade[4]={1.0,1.0,1.0,1.0};
        GLint especMaterial = 60;

        // Especifica que a cor de fundo da janela será preta
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

        // Habilita o modelo de colorização de Gouraud
        glShadeModel(GL_SMOOTH);

        // Define a refletância do material
        glMaterialfv(GL_FRONT,GL_SPECULAR, especularidade);
        // Define a concentração do brilho
        glMateriali(GL_FRONT,GL_SHININESS,especMaterial);

        // Ativa o uso da luz ambiente
        glLightModelfv(GL_LIGHT_MODEL_AMBIENT, luzAmbiente);

        // Define os parâmetros da luz de número 0
        glLightfv(GL_LIGHT0, GL_AMBIENT, luzAmbiente);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, luzDifusa );
        glLightfv(GL_LIGHT0, GL_SPECULAR, luzEspecular );
        glLightfv(GL_LIGHT0, GL_POSITION, posicaoLuz );

        // Habilita a definição da cor do material a partir da cor corrente
        glEnable(GL_COLOR_MATERIAL);
        //Habilita o uso de iluminação
        glEnable(GL_LIGHTING);
        // Habilita a luz de número 0
        glEnable(GL_LIGHT0);
        // Habilita o depth-buffering
        glEnable(GL_DEPTH_TEST);

    angle=45;
}

// Função usada para especificar o volume de visualização
void EspecificaParametrosVisualizacao(void)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(angle,fAspect,0.1,500);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(0,80,200, 0,0,0, 0,1,0);
}

// Função callback chamada quando o tamanho da janela é alterado
void AlteraTamanhoJanela(GLsizei w, GLsizei h)
{
        if ( h == 0 ) h = 1;
        glViewport(0, 0, w, h);
        fAspect = (GLfloat)w/(GLfloat)h;
        EspecificaParametrosVisualizacao();
}

// Função callback chamada para gerenciar eventos do mouse
void GerenciaMouse(int button, int state, int x, int y)
{
```

```
        if (button == GLUT_LEFT_BUTTON)
                if (state == GLUT_DOWN) {  // Zoom-in
                        if (angle >= 10) angle -= 5;
                }
        if (button == GLUT_RIGHT_BUTTON)
                if (state == GLUT_DOWN) {  // Zoom-out
                        if (angle <= 130) angle += 5;
                }
        EspecificaParametrosVisualizacao();
        glutPostRedisplay();
}

// Programa Principal
void main(void)
{
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(400,350);
        glutCreateWindow("Visualizacao 3D");
        glutDisplayFunc(Desenha);
        glutReshapeFunc(AlteraTamanhoJanela);
        glutMouseFunc(GerenciaMouse);
        Inicializa();
        glutMainLoop();
}
```

Figura 13.1 – Saída do programa Iluminacao.c

As funções utilizadas neste exemplo, e que ainda não foram descritas, são apresentadas a seguir segundo Woo et al. [2] e Wright e Sweet [3].

- ***glShadeModel(GL_SMOOTH);*** estabelece o modelo de colorização: GL_FLAT (a cor não varia na primitiva que é desenhada - um exemplo de primitiva é o triângulo); ou GL_SMOOTH (a cor de cada ponto da primitiva é interpolada a partir da cor calculada nos vértices). Seu protótipo é: *void glShadeModel(GLenum mode);*. Descrição dos parâmetros: *mode* especifica o modelo de colorização, sendo que o *default* é GL_SMOOTH [3].

- ***glMaterialfv(GL_FRONT,GL_SPECULAR, especularidade);*** estabelece os parâmetros do material que serão usados pelo modelo de iluminação. Possui algumas variações cujos protótipos são:

  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);*

  Descrição dos parâmetros: *face* determina se as propriedades do material dos polígonos que estão sendo especificadas são *front* (GL_FRONT), *back* (GL_BACK) ou ambas (GL_FRONT_AND_BACK); *pname* para as duas primeiras variações especifica o parâmetro de um único valor que está sendo determinado (atualmente apenas GL_SHININESS possui um único valor como parâmetro); para as duas últimas variações, que recebem um vetor como parâmetro, pode determinar as seguintes propriedades do material: GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_SHININESS, GL_AMBIENT_AND_DIFFUSE ou GL_COLOR_INDEXES; *param* (GLfloat ou GLint) especifica o valor que será atribuído para o parâmetro determinado por *pname* (neste caso, GL_SHININESS); *params* (GLfloat* ou GLint*) um vetor de números reais ou inteiros que contém as componentes da propriedade que está sendo especificada. Através desta função é possível determinar as propriedades de refletância do material dos polígonos. As propriedades GL_AMBIENT, GL_DIFFUSE e GL_SPECULAR afetam a maneira na qual as componentes de luz incidente são refletidas. GL_EMISSION é usado para materiais que possuem "luz própria". GL_SHININESS pode variar de 0 a 128 (quanto maior o valor, maior é a área de *highlight* especular na superfície [4]). GL_COLOR_INDEXES é usado para as propriedades de refletância do material no modo de índice de cores [3].
- ***glLightModelfv(GL_LIGHT_MODEL_AMBIENT, luzAmbiente);*** estabelece os parâmetros do modelo de iluminação usado por OpenGL. É possível especificar um, ou todos os três modelos: GL_LIGHT_MODEL_AMBIENT é usado para especificar a luz ambiente *default* para uma cena, que tem um valor RGBA *default* de (0.2, 0.2, 0.2, 1.0); GL_LIGHT_MODEL_TWO_SIDE é usado para indicar se ambos os lados de um polígono são iluminados (por *default* apenas o lado frontal é iluminado); GL_LIGHT_MODEL_LOCAL_VIEWER modifica o cálculo dos ângulos de reflexão especular; Possui algumas variações cujos protótipos são:

  glLightModelf(GLenum pname, GLfloat param);

  glLightModeli(GLenum pname, GLint param);

  *glLightModelfv(GLenum pname, const GLfloat *params);*

  *glLightModeliv(GLenum pname, const GLint *params);*

  Descrição dos parâmetros: *pname* especifica um parâmetro do modelo de iluminação: GL_LIGHT_MODEL_AMBIENT, GL_LIGHT_MODEL_LOCAL_VIEWER ou GL_LIGHT_MODEL_TWO_SIDE; *param* (GLfloat ou GLint) para GL_LIGHT_MODEL_LOCAL_VIEWER um valor 0.0 indica que os ângulos da componente especular tomam a direção de visualização como sendo paralela ao eixo z, e qualquer outro valor indica que a visualização ocorre a partir da origem do sistema de referência da câmera; para GL_LIGHT_MODEL_TWO_SIDE um valor 0.0 indica que somente os polígonos frontais são incluídos nos cálculos de iluminação, e qualquer outro valor indica que todos os polígonos são incluídos nos cálculos de iluminação; *params* (GLfloat* ou GLint*) para GL_LIGHT_MODEL_AMBIENT ou GL_LIGHT_MODEL_LOCAL_VIEWER, aponta para um vetor de números inteiros ou reais; para GL_LIGHT_MODEL_AMBIENT o conteúdo do vetor indica os valores das componentes RGBA da luz ambiente [3].
- ***glLightfv(GL_LIGHT0, GL_AMBIENT, luzAmbiente);*** estabelece os parâmetros da fonte de luz para uma das oito fontes de luz disponíveis. Possui algumas variações cujos protótipos são:

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);*
As duas primeiras variações requerem apenas um único valor para determinar uma das seguintes propriedades: GL_SPOT_EXPONENT, GL_SPOT_CUTOFF, GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION e GL_QUADRATIC_ATTENUATION. As duas últimas variações são usadas para parâmetros de luz que requerem um vetor com múltiplos valores (GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION e GL_SPOT_DIRECTION). Descrição dos parâmetros: *light* especifica qual fonte de luz está sendo alterada (varia de 0 a GL_MAX_LIGHTS); valores constantes de luz são enumerados de GL_LIGHT0 a GL_LIGHT7; *pname* especifica qual parâmetro de luz está sendo determinado pela chamada desta função (GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION, GL_SPOT_DIRECTION, GL_SPOT_EXPONENT, GL_SPOT_CUTOFF. GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, GL_QUADRATIC_ATTENUATION); *param* (GLfloat ou GLint) para parâmetros que são especificados por um único valor (*param*); estes parâmetros, válidos somente para *spotlights*, são GL_SPOT_EXPONENT, GL_SPOT_CUTOFF, GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION e GL_QUADRATIC_ATTENUATION. *params* (GLfloat* ou GLint*) um vetor de valores que descrevem os parâmetros que estão sendo especificados [3].

- ***glEnable(GL_COLOR_MATERIAL);*** conforme já explicado na seção 6, a função *glEnable* é usada para habilitar uma variável de estado OpenGL. Neste caso, estão sendo habilitadas: GL_COLOR_MATERIAL (atribui a cor para o material a partir da cor corrente), GL_DEPTH_TEST (controla as comparações de profundidade e atualiza o *depth buffer*), GL_LIGHTING (habilita a iluminação) e GL_LIGHT0 (habilita a luz de número 0) [2].
- ***glutSwapBuffers();*** e ***glutInitDisplayMode(GLUT_DOUBLE...);*** usadas para evitar que a imagem fique "piscando" a cada interação (por exemplo, quando se faz *zoom-in* e *zoom-out*).

14. MALHA DE POLÍGONOS

Como a maioria dos objetos em Computação Gráfica é representada através de uma malha de polígonos, logo abaixo é apresentada uma função que exemplifica o uso de uma malha de polígonos para fazer o desenho de um cubo.

```
void Desenha(void)
{

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3f(0.0f, 0.0f, 1.0f);

        // Desenha um cubo

        glBegin(GL_QUADS);                          // Face posterior
                glNormal3f(0.0, 0.0, 1.0);          // Normal da face
                glVertex3f(40.0, 40.0, 40.0);
                glVertex3f(-40.0, 40.0, 40.0);
                glVertex3f(-40.0, -40.0, 40.0);
                glVertex3f(40.0, -40.0, 40.0);
        glEnd();
```

```
        glBegin(GL_QUADS);                          // Face frontal
                glNormal3f(0.0, 0.0, -1.0);         // Normal da face
                glVertex3f(40.0, 40.0, -40.0);
                glVertex3f(40.0, -40.0, -40.0);
                glVertex3f(-40.0, -40.0, -40.0);
                glVertex3f(-40.0, 40.0, -40.0);
        glEnd();
        glBegin(GL_QUADS);                          // Face lateral esquerda
                glNormal3f(-1.0, 0.0, 0.0);         // Normal da face
                glVertex3f(-40.0, 40.0, 40.0);
                glVertex3f(-40.0, 40.0, -40.0);
                glVertex3f(-40.0, -40.0, -40.0);
                glVertex3f(-40.0, -40.0, 40.0);
        glEnd();
        glBegin(GL_QUADS);                          // Face lateral direita
                glNormal3f(1.0, 0.0, 0.0);          // Normal da face
                glVertex3f(40.0, 40.0, 40.0);
                glVertex3f(40.0, -40.0, 40.0);
                glVertex3f(40.0, -40.0, -40.0);
                glVertex3f(40.0, 40.0, -40.0);
        glEnd();
        glBegin(GL_QUADS);                          // Face superior
                glNormal3f(0.0, 1.0, 0.0);          // Normal da face
                glVertex3f(-40.0, 40.0, -40.0);
                glVertex3f(-40.0, 40.0, 40.0);
                glVertex3f(40.0, 40.0, 40.0);
                glVertex3f(40.0, 40.0, -40.0);
        glEnd();
        glBegin(GL_QUADS);                          // Face inferior
                glNormal3f(0.0, -1.0, 0.0);         // Normal da face
                glVertex3f(-40.0, -40.0, -40.0);
                glVertex3f(40.0, -40.0, -40.0);
                glVertex3f(40.0, -40.0, 40.0);
                glVertex3f(-40.0, -40.0, 40.0);
        glEnd();
        glutSwapBuffers();
}
```

Quando se trabalha com uma malha, normalmente são passadas como argumento para a função *glBegin()* as primitivas GL_QUADS, GL_POLYGON e GL_TRIANGLES. GL_TRIANGLE_STRIP e GL_QUAD_STRIP também podem ser usadas, desde que se tenha certeza de que os vértices e faces estão ordenados da maneira correta.

Conforme se observa no código acima, é necessário informar o vetor normal de cada face, pois OpenGL usa esta informação para calcular a cor de cada face de acordo com os parâmetros de iluminação. A função usada para especificar o vetor normal é *glNormal3f*, e os seus parâmetros são as componentes x, y e z do vetor normal, respectivamente.

BIBLIOGRAFIA

[1] HILL, Francis S. **Computer Graphics Using OpenGL**. 2nd ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc., 2000. 922 p.

[2] WOO, Mason; NEIDER, Jackie; DAVIS, Tom; SHREINER, Dave. **OpenGL Programming Guide: the official guide to learning OpenGL, version 1.2**. 3rd ed. Reading, Massachusetts: Addison Wesley, 1999. 730 p.

[3] WRIGHT, Richard S. Jr.; SWEET, Michael. **OpenGL SuperBible**. 2nd ed. Indianapolis, Indiana: Waite Group Press, 2000. 696 p.

[4] FOLEY, J. et al. **Computer graphics**: principles and practice. 2nd ed. New York: Addison Wesley, 1990.