Aula 24

Biblioteca OpenGL

Tutorial de OpenGL

http://www.ingleza.com.br/opengl/11-2.html

OpenGL é uma interface (Biblioteca) para dispositivos de hardware

- consiste em cerca de 150 comandos
- para especificar os objetos
- operações para gerar aplicativos tridimensionais interativos.

OpenGL foi desenvolvido independentemente da interface de hardware para ser implementado em múltiplas plataformas de hardware

Operações por vértices

As coordenadas espaciais são projetadas de uma posição no mundo 3D a uma posição na tela

Se as características avançadas estiverem habilitadas permitidas, este estágio é o mais ocupado

Se texturização for usada, as coordenadas da textura podem ser geradas e aplicadas

Se as luzes forem habilitadas, os cálculos da luz serão executados com:

- os vértices transformados
- a normal da superfície
- a posição da fonte de luz
- as propriedades de materiais
- a outras informações de luzes para produzir um valor da cor

Funções gráficas do OPENGL

Buffer de acumulação

- Trata-se de um buffer no qual múltiplos frames renderizados, podem ser compostos para produzir uma única imagem.
- Usado para efeitos tais como a profundidade de campo, “blur” de movimento, e de anti-aliasing da cena

Alfa Blending

- Provê mecanismos para criar objetos transparentes
- Usando a informação alfa, um objeto pode ser definido como totalmente transparente até totalmente opaco.

Funções gráficas do OPENGL

Modo "Color-Index"

- Buffer de Cores que armazena índices de cores das componentes vermelhas, verdes, azuis, e alfa das cores (RGBA).

Display Lists

- Uma lista nomeada de comandos de OpenGL. Os índices de um Display list podem ser pré-processados e podem conseqüentemente executar mais eficientemente do que o mesmo conjunto de comandos do OpenGL executados no modo imediato.

Double buffering

- Usado para fornecer uma animação suave dos objetos
- Cada cena sucessiva de um objeto em movimento pode ser construída em "background" ou no buffer "invisível" e então apresentado
- Permite que somente as imagens completas sejam apresentadas na tela

Funções gráficas do OPENGL

FeedBack

- Um modo onde OpenGL retorna à aplicação, a informação geométrica processada (cores, posições do pixel, e assim por diante)

Gouraud Shading

- Interpolação suave das cores através de um segmento de polígono ou de linha
- As cores são atribuídas em vértices e linearmente interpoladas através da primitiva para produzir uma variação relativamente suave na cor

Modo Imediato

- A execução de comandos OpenGL quando eles são chamados, possui resultado melhor do que os “Display Lists”

Iluminação e sombreamento de materiais

- A habilidade de computar exatamente a cor de algum ponto dado as propriedades materiais para a superfície

Funções gráficas do OPENGL

Operações de pixel

- Armazena, transforma, traça e processa aumento e redução de imagens

Executores polinomiais

- Para suportar as NURBS (non-uniform rational B-splines).

Primitivas

- Um ponto, uma linha, um polígono, um bitmap, ou uma imagem.

Primitivas da rasterização

- bitmaps e retângulos de pixels

Modo RGBA

- Buffers de cores armazenam componentes vermelhos, verdes, azuis, e alfa da cor

Ambiente de desenvolvimento

Instalação do OPENGL

As bibliotecas do OpenGL são distribuídas como parte dos sistemas operacionais da Microsoft, porém as mesmas podem ser baixadas no site oficial do OpenGL : http://www.opengl.org. Estas bibliotecas também estão disponíveis em outros sistemas operacionais por padrão, tais como, MacOS e Unix Solaris, além de estarem disponíveis no Linux.

Instalação do GLUT

O GLUT é um conjunto de ferramentas para escrita de programas OpenGL, independente do sistema de janelas

Ele implementa um sistema de janelas simples através de sua API, para os programas OpenGL

GLUT provê uma API portável, o que permite que programas trabalhem tanto em ambientes baseados em WIN32 quanto X11

O GLUT suporta :

- Janelas múltiplas para renderização OpenGl
- Resposta a eventos baseados em Callback de funções
- Uma rotina “idle” e “timers”
- Criação de menus pop-up simples
- Suporte para bitmaps e fontes
- Uma miscelânea de funções para gerenciamento de janelas.

Instalando o GLUT no DEV-C++

O Dev-C++ é um ambiente de desenvolvimento integrado para a linguagem de programação C/C++. O compilador Dev-C++, que dá suporte a compiladores baseados no GCC (GNU Compiler Collection), pode ser obtido em http://www.bloodshed.net/devcpp.html.

A versão mais atual do Dev-C++ (4.9.8 ou superior) já inclui as bibliotecas do GLUT por padrão devendo apenas ter o passo 2 executado. Para versões anteriores siga os seguintes passos :

1. Faça o download do arquivo glut-devc.zip e descompacte o mesmo;
2. Mova o arquivo glut.h para a pasta GL do DevC++ (C:\Dev-C++\Include\GL);
3. Mova os arquivos glut32.def e libglut.a para a pasta Lib do DevC++ (C:\Dev-C++\Lib);
4. Mova o arquivo glut32.dll para a mesma pasta onde se encontram os arquivos opengl32.dll e glu32.dll;

Usando o OPENGL no Lazarus

- **LCL**
- A Lazarus Component Library pode ser usada com o OpenGL também
- Lazarus inclui um TOpenGLControl – um controle LCL com um contexto OpenGL
- O pacote lazarus LazOpenGLContext pode ser encontrado em: lazarus/components/opengl/lazopenglcontext.lpkUm exemplo pode ser encontrado em: lazarus/examples/openglcontrol/openglcontrol_demo.lpi.

Usando o OPENGL no Lazarus

- **LCL / GLFW / GLUT**
- Quando usar GLUT ou LCL?
- GLUT é melhor se você quer fazer tudo por sua conta
- LCL é melhor para aplicações normais
  - Por exemplo, um editor 3D precisa de uma janela OpenGL e o resto é uma aplicação normal, com botões, caixas, janelas, etc.
- GLUT precisa de uma dll no windows, enquanto que o LCL executa pelo Lazarus, entretanto, o executável LCL é maior

Usando a GLUT no Lazarus

- Crie um programa Simples (é parecido com um programa console)

- Vai gerar um código assim:

program Project1;

begin
end.

Fazer em sala

- Salve em uma pasta

Usando a GLUT no Lazarus

Coloque as DLLs na mesma pasta

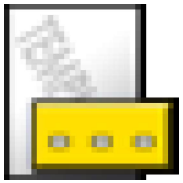
glut32.dll

glut.def

glut.txt

glut32.lib

glut.h



glut.template

Pegue eles aqui:
https://www.dropbox.com/s/72v308ji6s0hz73/LAZARUS_GLUT.rar?dl=0

Usando a GLUT no Lazarus

Copie o código abaixo em cima do código original

```
program project1;
{$Apptype GUI}

Uses   GL, GLU, GLUT;  // Put glut32.dll in program folder if necessary

procedure display; cdecl;
begin
  glClearColor (1.0, 1.0, 1.0, 1.0); // white background
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity;
  glOrtho(0, 200, 0, 200, -1, 1);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f (1.0, 0.0, 0.0);  // red
```

Usando a GLUT no Lazarus

```
  glBegin(GL_POLYGON);
    // Irregular convex pentagon
    glVertex2i(5, 155);
    glVertex2i(10, 190);
    glVertex2i(40, 180);
    glVertex2i(175, 75);
    glVertex2i(50, 20);
  glEnd;

  glutSwapBuffers;
end;
```

Usando a GLUT no Lazarus

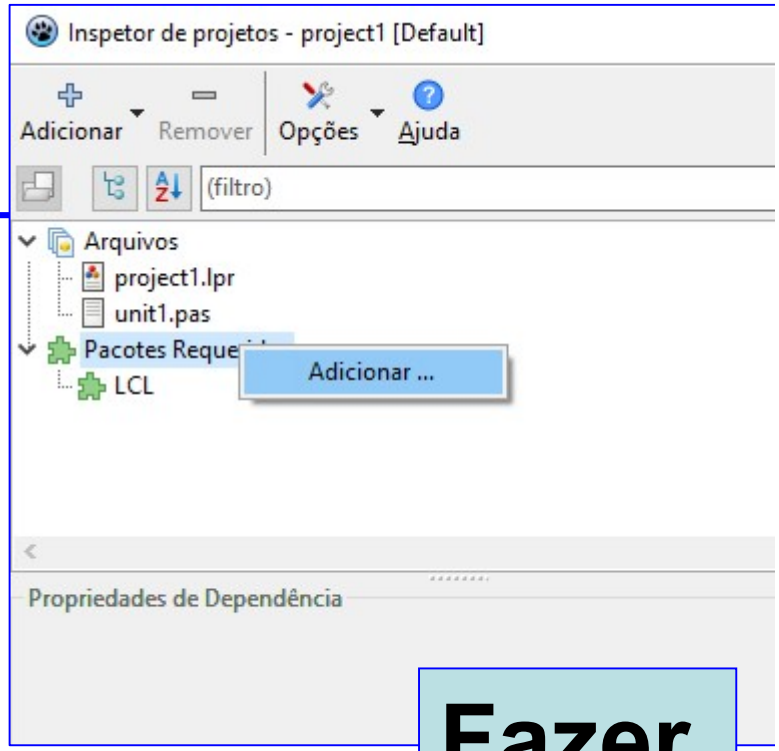
```
begin
  glutInit(@argc, @argv);
  glutInitDisplayMode (GLUT_DOUBLE OR GLUT_RGB OR
     GLUT_DEPTH);
  glutInitWindowSize (200, 200);
  glutCreateWindow ('Glut Test');
  glutDisplayFunc(@display);
  glutMainLoop;
end.
```

Usando o OPENGL no Lazarus

- Veja mais em:

http://wiki.freepascal.org/OpenGL_Tutorial

Usando o OPENGL no Lazarus - LCL



- Inicie o Lazarus
- Escolha **Projeto / Novo Projeto**
- Escolha **Aplicação**
- Escolha **Projeto / Inspetor de Projetos**
- Escolha **Adicionar um Item**
- Escolha a aba **Novo Requerimento**
- Escolha **Nome do Pacote = lazopenglcontext**
- Escolha o Botão **Criar Novo Requerimento**
- Agora, cole o código abaixo sobre o código orioginal :
- E clique no Evento OnCreate do Form, para eles se comunicarem (TForm1.FormCreate(Sender: TObject); com o OnCreate)

Fazer em sala

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, OpenGLContext, gl;

type
  { TForm1 }
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure GLboxPaint(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;
```

```
var
  Form1: TForm1;
implementation
{$R *.lfm}
 var   GLBox:TOpenGLControl;
{ TForm1 }
procedure TForm1.GLboxPaint(Sender: TObject);
begin
  glClearColor(0.27, 0.53, 0.71, 1.0); //Set blue background
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
  glLoadIdentity;
  glBegin(GL_TRIANGLES);
    glColor3f(1, 0, 0);              glVertex3f( 0.0, 1.0, 0.0);
    glColor3f(0, 1, 0);              glVertex3f(-1.0,-1.0, 0.0);
    glColor3f(0, 0, 1);              glVertex3f( 1.0,-1.0, 0.0);
  glEnd;
  GLbox.SwapBuffers;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  GLbox:= TOpenGLControl.Create(Form1);
  GLbox.AutoResizeViewport:= true;
  GLBox.Parent := self;
  GLBox.MultiSampling:= 4;
  GLBox.Align := alClient;
  GLBox.OnPaint := @GLboxPaint;
  //for "mode delphi" this would be "GLBox.OnPaint := GLboxPaint"
  GLBox.invalidate;
end;
end.
```

Sintaxe de Comandos do OpenGL

Os comandos OpenGL obedecem um padrão para os nomes de funções e constantes

Todos os comandos utilizam o prefixo **gl** em letras minúsculas.

OpenGL define constantes com as iniciais GL_, em letras maiúsculas, e usa um “underscore” para separar as palavras (Ex. GL_COLOR_BUFFER_BIT).

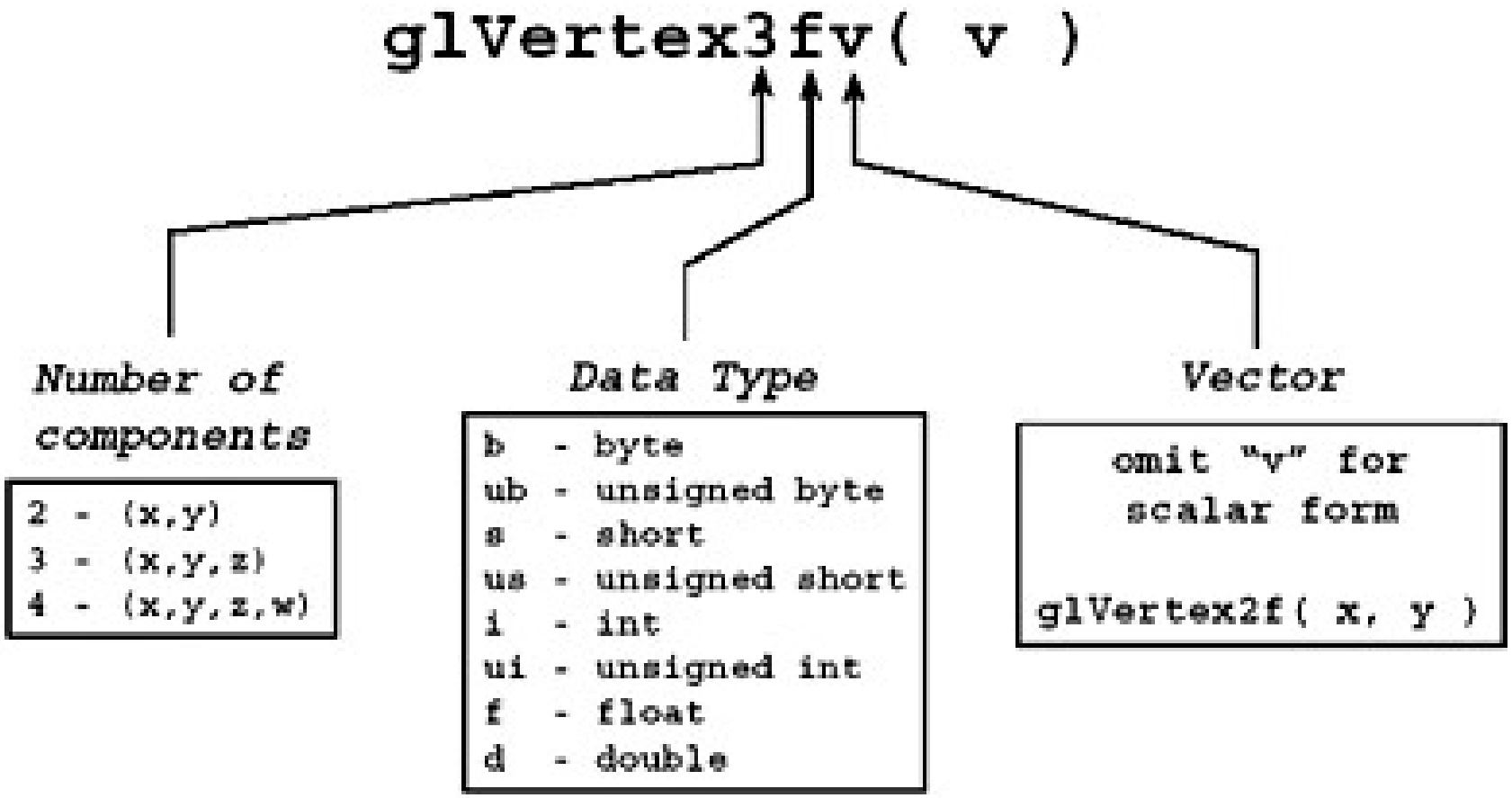
Em algumas funções algumas seqüências de caracteres extras aparecem,como sufixo, no nome destas funções

Ex. glColor3f e glVertex3f

No comando glColor3f(), o “3” do sufixo indica que três argumentos são necessários;

Outra versão de “Color” necessita de 4 argumentos

O “f” do sufixo indica que os argumentos são do tipo números de ponto flutuante.


glVertex3fv(v)
Number of components
2 - (x,y)
3 - (x,y,z)
4 - (x,y,z,w)
Data Type
b - byte
ub - unsigned byte
s - short
us - unsigned short
i - int
ui - unsigned int
f - float
d - double
Vector
omit "v" for scalar form
glVertex2f(x, y)

Estrutura Básica de Programas OpenGL

Um programa OpenGL deve conter, um mínimo de requisitos para sua perfeita execução

Normalmente alguns passos devem ser seguidos para criação destes programas

Estes passos são :

- Declaração dos arquivos de header para o OpenGL
- Configurar e abrir a janela.
- Inicializar os estados no OpenGL
- Registrar as funções de “callback”
- Renderização;
- Redimensionamento
- Entradas : teclado, mouse, etc.
- Entrar no loop de processamento de eventos

O programa exemplo abaixo irá ilustrar estes passos

```
#include <GL/gl.h>
#include <GL/glut.h>
void main( int argc, char** argv )
{
    int mode = GLUT_DOUBLE | GLUT_RGB;
    glutInitDisplayMode( mode );
    glutInitWindowSize(400,350);
    glutInitWindowPosition(10,10);
    glutCreateWindow( argv[0] );
    init();
    glutDisplayFunc( display );
    glutReshapeFunc( resize );
    glutKeyboardFunc( key );
    glutIdleFunc( idle );
    glutMainLoop();
}
```

• Os arquivos de header, contém as rotinas e declarações para usar o OpenGL/GLUT com a linguagem c/c++

• As funções glutInitDisplayMode() e glutCreateWindow() compõem o passo de configuração da janela

O modo da janela que é argumento para a função glutInitDisplayMode(),indica a criação de uma janela double-buffered (GLUT_DOUBLE) com o modo de cores RGBA (GLUT_RGB)

O primeiro significa que os comandos de desenho são executados para criar uma cena fora da tela para depois rapidamente colocá-la na view (ou 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

A função glutCreateWindow() cria a janela com base no parâmetros definidos nas funções glutInitWindowSize (Tamanho da janela em pixels) e glutInitWindowPosition (coordenadas para criação da janela)

Esta janela conterá o nome especificado em seu parâmetro de entrada

• Em seguida é chamada a rotina init(), a qual contem a inicialização do programa

• O próximo passo é o registro das funções de callback, que serão usadas no programa

• Finalmente, o programa irá entrar em um processo de loop, o qual interpreta os eventos e chamadas das rotinas especificadas como callback

Rotinas de Callback

As funções de callback são aquelas executadas quando qualquer evento ocorre no sistema, eventos tais como:

- redimensionamento de janela
- desenho da mesma
- entradas de usuários (teclado, mouse, ou outro dispositivo de entrada)
- ocorrência de animações.

Assim, o desenvolvedor associará uma ação à ocorrência de um evento

GLUT oferece suporte a muitos diferentes tipos de ações de callback:

glutDisplayFunc() – quando um pixel na janela necessita ser atualizado

glutReshapeFunc() – quando a janela é redimensionada

glutKeyboardFunc() – quando uma tecla do teclado é pressionada

glutMouseFunc() – quando o usuário pressiona um botão do mouse

glutMotionFunc() – quando o usuário movimenta o mouse enquanto mantém um botão do mesmo pressionado

glutPassiveMouseFunc() – quando o mouse é movimentado, independente do estado dos botões

glutIdleFunc() – quando nada está acontecendo, muito útil para animações

Exemplo de um programa OpenGL

Este exemplo simples, apenas desenha um quadrado na tela.

```
/* Exemplo1.c - Marcionílio Barbosa Sobrinho
 * Programa simples que apresenta o desenho de um quadrado
 * Objetivo : Demonstrar funções de gerenciamento de
 *                janelas e funções de callback
 * Referência do Código: OpenGL Programming Guide - RedBook
*/
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h&lg;
void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);        /* Limpa o Buffer de Pixels */

  glColor3f (1.0, 1.0, 1.0);                // Define a cor padrão como branco
```

Exemplo de um programa OpenGL

```
  /* desenha um simples retângulo com as coordenadas
  * (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)                    */
  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();
  /* Inicia o processo de desenho através dos dados bufferizados
*/
  glFlush ();
}
void init (void)
{
   /* Seleciona a cor de fundo para limpeza da tela */
   glClearColor (0.0, 0.0, 0.0, 0.0);

   /* inicializa os valores de visualização */
   glMatrixMode(GL_PROJECTION);
```

Exemplo de um programa OpenGL

```
/* Faz com que a matriz corrente seja inicializada  com a matriz identidade
(nenhuma transformação é acumulada)   */
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

/*          Cria a janela       */
int main(int argc, char** argv)
{
  /* Estabelece o modo de exibição a ser utilizado pela janela a ser criada
     neste caso utiliza-se de um buffer simples, ou seja, a apresentação será
     imediata à execução Define o modo de cores como RGBA          */
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

   /*       Determina o tamanho em pixels da  janela a ser criada        */
   glutInitWindowSize (250, 250);

   /*    Estabelece a posição inicial para criação da   janela    */
   glutInitWindowPosition (100, 100);
```

Exemplo de um programa OpenGL

```
   /*  Cria uma janela com base nos parâmetros especificados nas funções
       glutInitWindowSize e glutInitWindowPosition com o nome de título
       especificado em seu argumento    */
   glutCreateWindow ("Exemplo 1");

 /* Especifica os parâmetros inicias para as variáveis de estado do OpenGL */
   init ();

   // Associa a função display como uma função de callback
   glutDisplayFunc(display);

   /* Inicia a execução do programa OpenGL.  O programa irá executar num
      loop infinito devendo  o desenvolvedor especificar as condições de saída
      do mesmo através de interrupções no próprio programa ou através de
      comandos de mouse ou teclado como funções de callback      */
   glutMainLoop();
   return 0;
}
```

Criação de Primitivas

O OpenGL apresenta apenas 10 tipos de primitivas distintas, porém permite a criação de estruturas mais complexas

Pontos

Todos cálculos são feitos como se os vértices fossem tridimensionais, até mesmo os bidimensionais (z=0)

OpenGL trabalha com coordenadas homogêneas de geometria projetiva tridimensional, então todos os vértices são representados com quatro coordenadas de ponto-flutuante (x, y, z, w)

Linhas

Em OpenGL, o termo linha refere-se a segmento de linha

Polígonos

Polígonos são áreas fechadas por um loop simples de segmentos de linhas, especificados por vértices e seus pontos finais

Polígonos podem ter o seu interior preenchidos ou não

Restrições:
- As bordas de um polígono OpenGL não podem cruzar-se
- Os polígonos devem ser convexos (não podem ter recortes)

Desenhando Primitivas

No OpenGL todos objetos geométricos são descritos como um jogo ordenado de vértices. Para tal operação é utilizada a função glVertex*()

Exemplo :

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

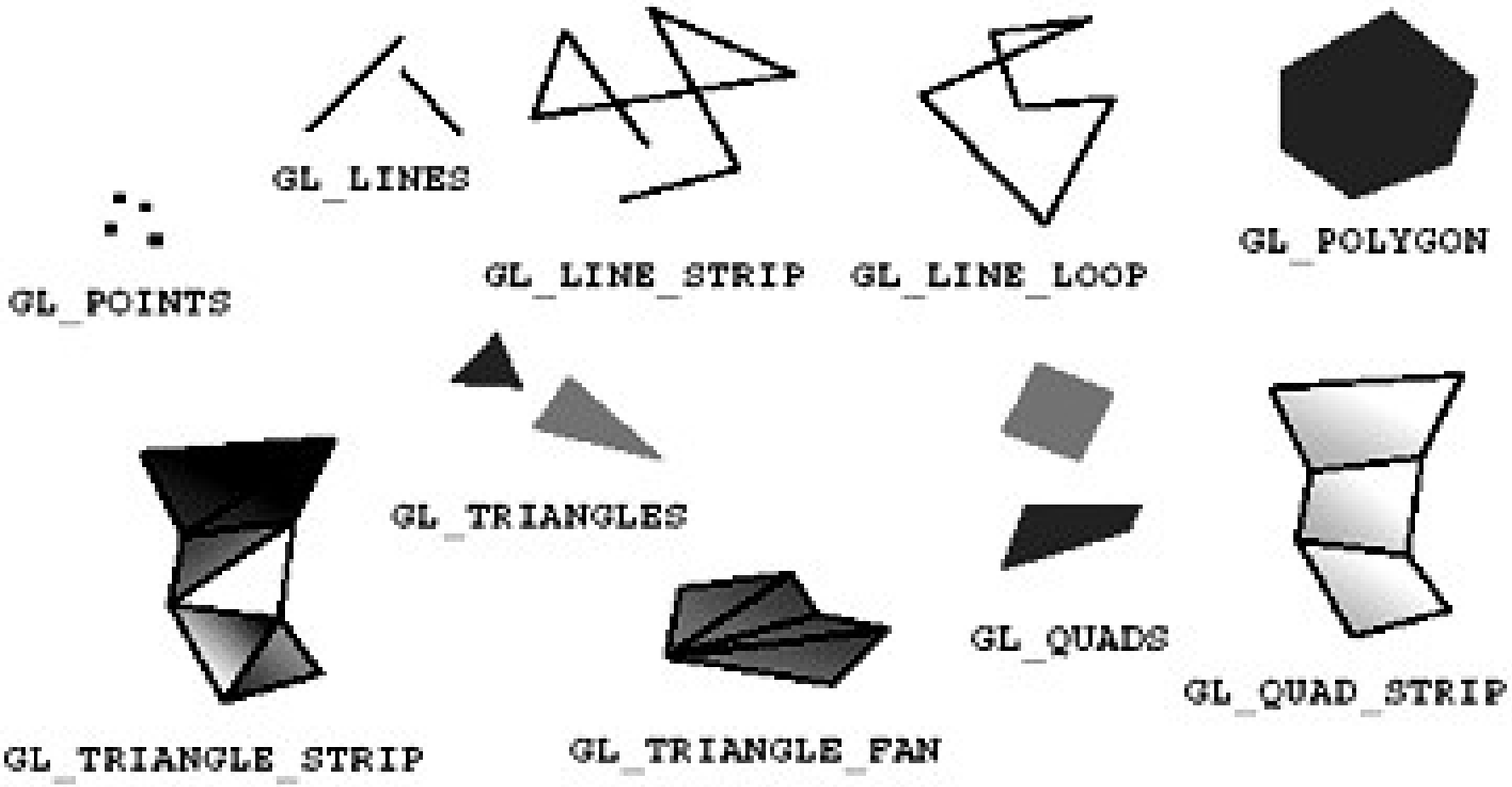
Após os vértices serem especificados, é preciso dizer ao OpenGL que tipo de primitiva será criada, colocando os vértices dentro do par **glBegin()** e **glEnd()**

O parâmetro do comando glBegin() é o tipo da primitiva a ser desenhada

Os tipos possíveis para parâmetros são :

Valor do parâmetro	Significado
GL_POINTS	Pontos Individuais
GL_LINES	Pares de vértices interpretados como segmentos de linha individuais
GL_LINE_STRIP	Série de segmentos de linha conectados
GL_LINE_LOOP	Como o anterior, porém com um segmento adicionado entre último e primeiro vértices
GL_TRIANGLES	Triplos vértices interpretados como triângulos
GL_TRIANGLE_STRIP	Faixas de triângulos unidas
GL_TRIANGLE_FAN	Leque de triângulos unidos
GL_QUADS	Quádruplo de vértices interpretados como polígonos de quatro lados
GL_QUAD_STRIP	Faixa quadrilateral unida
GL_POLYGON	limite de um polígono simples, convexo

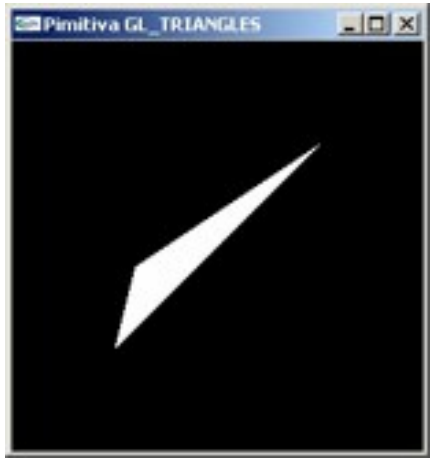
O comando glEnd() marca o fim de uma lista de dados de vértices.



A figura abaixo mostra a aplicação de algumas primitivas

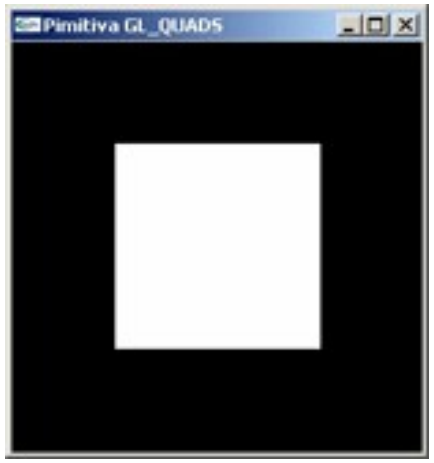
GL_TRIANGLES

```
glBegin(GL_TRIANGLES);
glVertex3f (0.25, 0.25, 0.0);
glVertex3f (0.75, 0.75, 0.0);
glVertex3f (0.30, 0.45, 0.0);
glEnd();
```



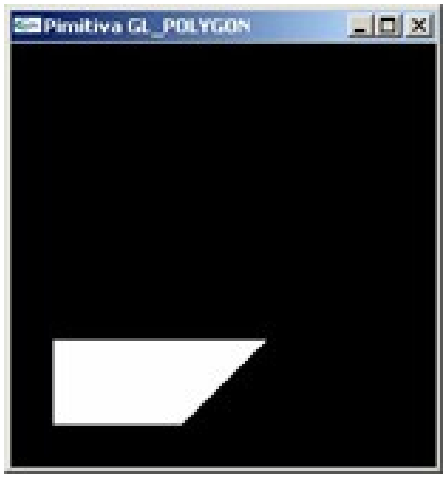
GL_QUADS

```
glBegin(GL_QUADS);
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();
```



GL_POLYGON

```
glBegin(GL_POLYGON);
glVertex3f( 0.10, 0.10 ,0.0 );
glVertex3f( 0.10, 0.30,0.0);
glVertex3f( 0.40, 0.30,0.0);
glVertex3f( 0.60, 0.30,0.0);
glVertex3f( 0.40, 0.10,0.0);
glEnd();
```



Cores no OpenGL

No OpenGL é possível trabalhar com cores de dois modos diferente : Modo RGBA e Modo de Índice de cores

Escolhendo entre RGBA e Índice de Cores

Por prover maior flexibilidade que o modo de Índice de cores, normalmente o modo de cores utilizado é o RGBA

- O modo índice de cores é muito útil em truques de animações de mapeamento de cores e desenho de camadas

Definindo o Modo de cores

O comando para definição do modo de cores é :

void glutInitDisplayMode(unsigned int mode);

O modo deverá ser um ou mais da seguinte tabela

Modo	Descrição
GLUT_RGBA	Seleção do modo RGBA para a janela. Se nem o parâmetro GLUT_RGBA ou GLUT_INDEX forem definidos ele será o padrão
GLUT_RGB	Apelido para o GLUT_RGBA
GLUT_INDEX	Modo de índice de cores. A especificação deste modo anula o modo GLUT_RGBA se este também for especificado como modo
GLUT_SINGLE	Seleciona apenas um buffer para janela. Este é o valor padrão se nem GLUT_DOUBLE ou GLUT_SINGLE forem especificados.
GLUT_DOUBLE	Seleciona dois buffers para a janela. Sobrepõe o GLIT_SINGLE se especificado
GLUT_ACCUM	Ativa o modo de acumulação de buffer para a janela
GLUT_ALPHA	Ativa a componente ALFA de cor para a janela
GLUT_DEPTH	Ativa o buffer de profundidade para a janela
GLUT_STENCIL	Ativa o buffer de estêncil
GLUT_MULTISAMPLE	Selecione uma janela com opção de "multisampling". Se "multisampling" não está disponivel, uma janela de "non-multisampling" será escolhida automaticamente.
GLUT_STEREO	Seleção de uma janela no modo estéreo
GLUT_LUMINANCE	Selecione uma janela com um modelo de cores de Luminancia. Este modo provê a funcionalidade do RGBA de OpenGL, mas os componentes verdes e azuis não são mantidos no frame buffer. Ao invés disso o componente vermelho de cada pixel é convertido para um índice entre zero e glutGet(GLUT_WINDOW_COLORMAP_SIZE)-1 e é observado em um mapa de core por janela para determinar a cor de pixels dentro da janela. O colormap inicial de janelas de GLUT_LUMINANCE é inicializado para ser uma escala de cinza linear, mas pode ser modificado com as rotinas de colormap da GLUT

Para especificar a cor em um determinado vértice, o comando a ser usado é

glColor*()

onde o * representa o sufixo do número de coordenadas, o tipo de dados e o vetor.

A faixa de valores de cores é representada por valores de ponto flutuante que variam de 0 a 1

O OpenGL faz a conversão dos tipos de dados para valores de ponto flutuante

Exemplo da utilização de cores

```
/* Exemplo2.c - Marcionílio Barbosa Sobrinho
 * Programa simples que apresenta o desenho de um
 * quadrado com variação de cores nos vertices
 * Objetivo : Demonstrar a utilização de cores nos objetos
 * Referência do Código: OpenGL Programming Guide - RedBook
*/
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>
GLfloat vermelho;
GLfloat verde;
GLfloat azul;

void display(void)
{
   /* Limpa o Buffer de Pixels */
   glClear (GL_COLOR_BUFFER_BIT);

   // Define a cor padrão com base nos parametros
   glColor3f (vermelho, verde, azul);
```

```
   /* desenha um simples retângulo com as coordenadas
   * (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)
  */
   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();
   /* Inicia o processo de desenho através dos dados bufferizados   */
   glFlush ();
}
void init (void)
{
     /* Define os parâmetros de cores para obter a cor branca */
     vermelho = 1.0;
     verde = 1.0;
     azul = 1.0;
    /* Seleciona a cor de fundo para limpeza da tela */
    glClearColor (0.0, 0.0, 0.0, 0.0);
```

```
/* inicializa os valores de visualização */
    glMatrixMode(GL_PROJECTION);

   /* Faz a matriz corrente ser inicializada com a matriz identidade  (nenhuma
transformação é acumulada) */
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

/* Função responsável pelo controle do teclado.
   Dependendo da tecla pressionada : R,G,B, adiciona uma constante ao valor
   da mesma e redesenha novamente a cena com a  Nova cor obtida destes
   parâmetros.
   Se a tecla 'O' for pressionada volta o estado original da imagem.
   O operador ternário está comentado por apresentar incompatibilidades com
   o compilador Dev-C++, porém não existem incompatibilidades com o
   Borland C++ Builder nem com MS Visual C++
*/
```

```
void teclado(unsigned char tecla, int x, int y)
{
    switch (tecla) {
            case 'R':
            case 'r':// Incrementa o valor do parâmetro da cor Vermelho
                      // (vermelho - 0.1) < 0 ? vermelho = 1 : vermelho -= 0.1;
                    vermelho = vermelho - 0.1;
                    if (vermelho < 0)
                      vermelho = 1;
                    break;
            case 'G':
            case 'g':// Incrementa o valor do parâmetro da cor Verde
                    // (verde - 0.1 ) < 0 ? verde = 1 : verde -= 0.1;
                    verde = verde - 0.1;
                    if (verde < 0)
                      verde = 1;
                    break;
```

```
case 'B':
        case 'b':// Incrementa o valor do parâmetro da cor Azul
                // (azul - 0.1) < 0 ? azul= 1 : azul -= 0.1;
                azul = azul - 0.1;
                if (azul < 0)
                   azul = 1;
                break;
                  case 'O':
        case 'o':
            vermelho = 1.0;
                        verde = 1.0;
                  azul = 1.0;
          break;
   }
   glutPostRedisplay();
}
```

```
 /*    Funcão principal do programa.   */
int main(int argc, char** argv)
{

   /*  Estabelece o modo de exibição a ser utilizado pela janela a ser criada
neste caso utiliza-se de um buffer simples, ou seja, a apresentação será
imediata à execução Define o modo de cores como RGBA   */
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

   /*  Determina o tamanho em pixels da janela a ser criada    */
   glutInitWindowSize (250, 250);

   /*  Estabelece a posição inicial para criação da janela        */
   glutInitWindowPosition (100, 100);

/* Cria uma janela com base nos parâmetros especificados nas funções
glutInitWindowSize e glutInitWindowPosition com o nome de título
especificado em seu argumento  */
   glutCreateWindow ("Exemplo 2");
```

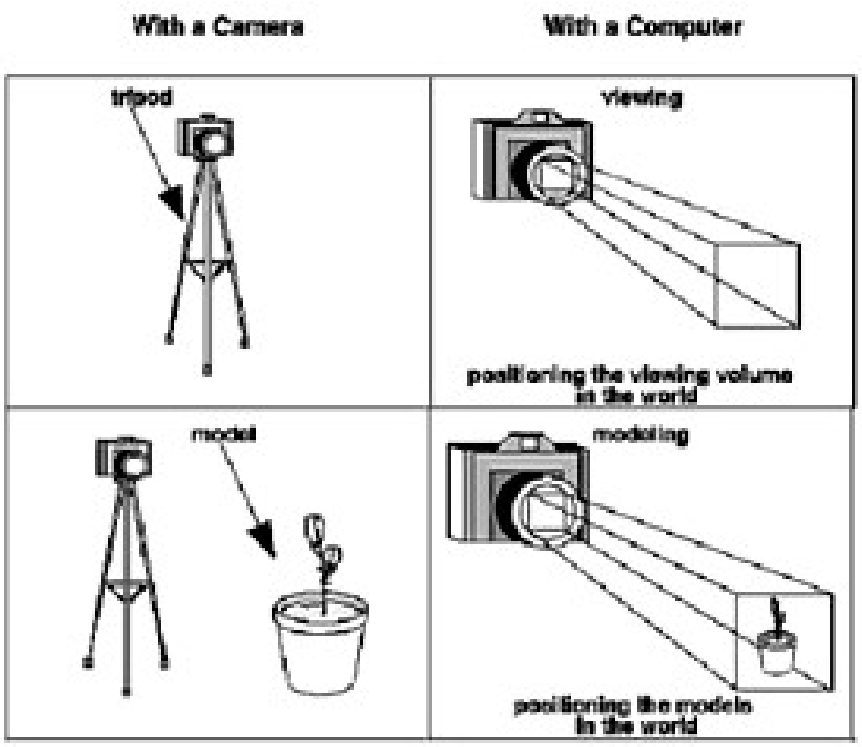
```
/*   Habilita a captura dos eventos de teclado     */

   glutKeyboardFunc(teclado);

/*  Especifica os parâmetros inicias para as variáveis de estado do OpenGL  */

   init ();

   // Associa a função display como uma função de callback

   glutDisplayFunc(display);

/* Inicia a execução do programa OpenGL. O programa irá executar num loop
    infinito devendo o desenvolvedor especificar as condições de saída do
    mesmo através de interrupções no próprio programa ou através de
   comandos de mouse ou teclado como funções de callback   */

   glutMainLoop();
   return 0;
}
```

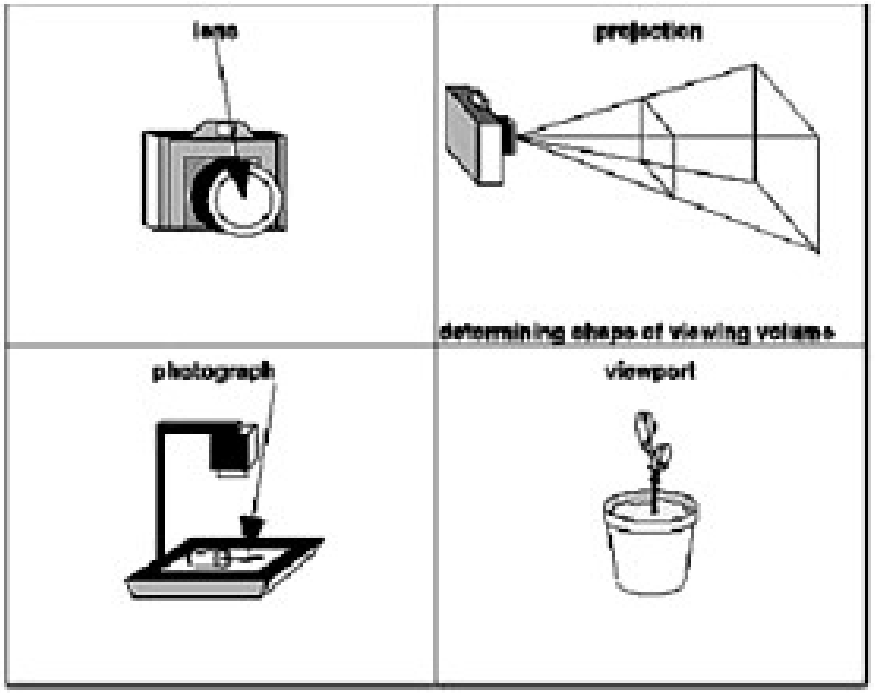
Transformações

O processo de transformação para produção de uma determinada cena é análogo à uma fotografia obtida com uma câmara

1. Montar o tripé e apontar a câmara à cena (transformação de visualização)
2. Organizar a cena a ser fotografada (modelagem de transformação)
3. Escolha de uma lente da câmara (transformação de projeção)
4. Determinar o tamanho da fotografia final - por exemplo, no caso de uma fotografia maior que a cena original (transformação de viewport)

Após estes passos, a foto será feita (a cena será desenhada)


With a Camera
With a Computer
tripod
viewing
positioning the viewing volume in the world
model
modeling
positioning the models in the world


lens
projection
determining shape of viewing volume
photograph
viewport

Transformação de Objetos

As três rotinas de OpenGL para modelar transformações são:

glTranslate * ()

glRotate * ()

glScale * ()

OpenGL executa estas transformações através de produto de matrizes (combinadas em uma única matriz)

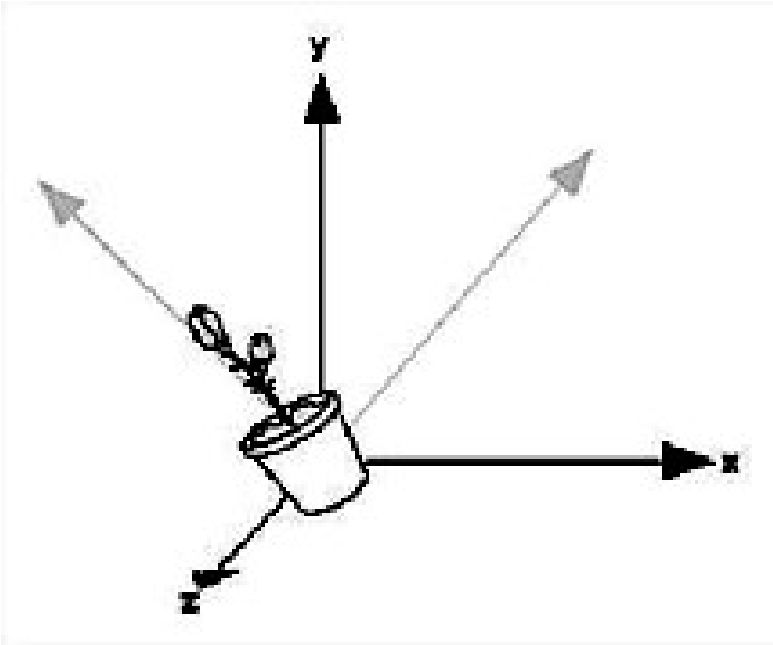
Rotação

A rotação usa glRotatef(Ângulo, x, y, z)

Rotação de "Ângulo" graus ao redor do eixo definido pelo vetor x,y,z no sentido anti-horário

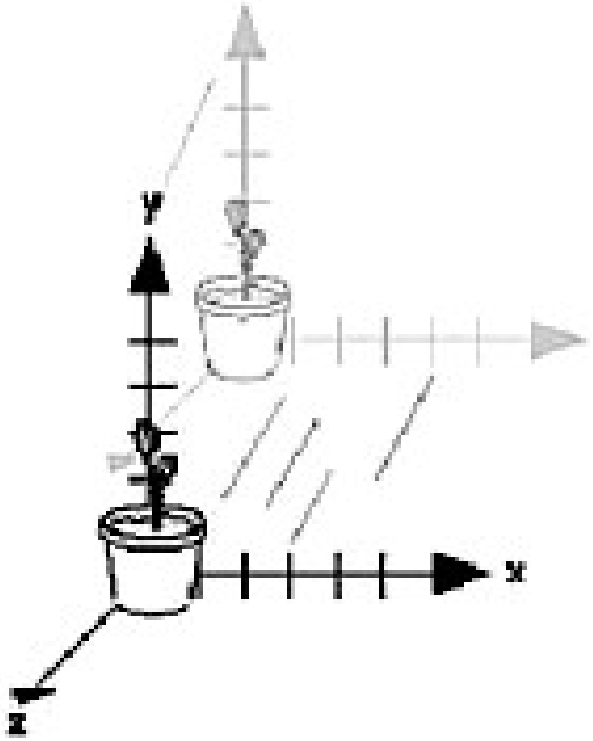
Ex : glRotatef(45.0, 0.0, 0.0, 1.0),

Rotaciona um objeto num ângulo de 45º



Translação :

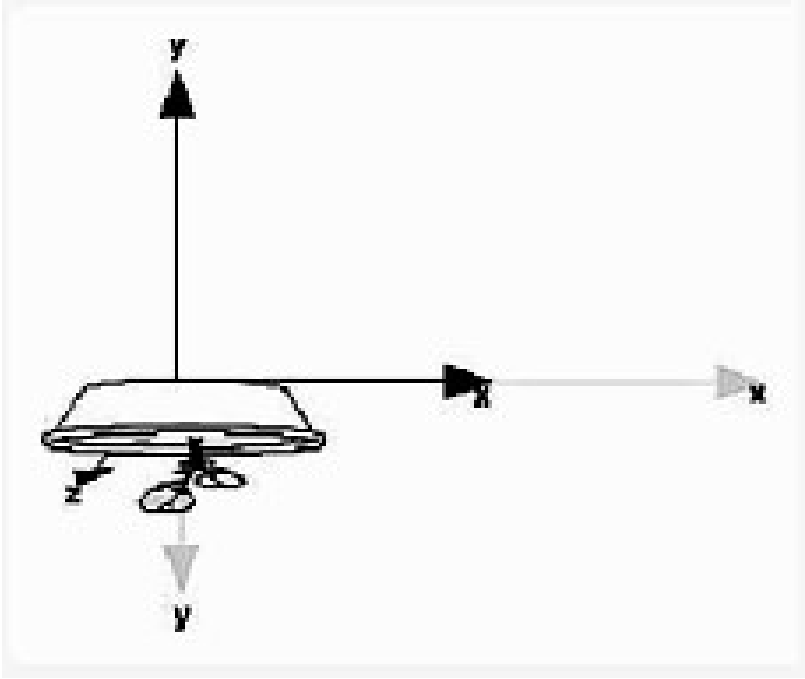
A translação usa glTranslatef(Tx, Ty, Tz)



Escala :

A escala usa glScalef(Ex, Ey, Ez)

Ex.: glScalef(2.0, -0.5, 1.0)



Exemplo de Transformação de Objetos

```
/* Exemplo3.c - Marcionílio Barbosa Sobrinho
 * Programa que apresenta as transformações aplicadas a uma
primitiva
 * Objetivo : Demonstrar a utilização de transformação de objetos
 * Referência do Código: OpenGL Programming Guide – RedBook    */
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>

GLfloat escala;
GLfloat translada;
GLfloat rotaciona;
```

```
void display(void)
{
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  /* Limpa o Buffer de Pixels */
  glClear(GL_COLOR_BUFFER_BIT);
  /* Estabelece a cor da primitiva */
  glColor3f (1.0f,1.0f,1.0f);
  /* Efetua a operação de translação */
  glTranslatef(translada, 0.0f, 0.0f);
  /* Efetua a operação de escala em Y */
  glScalef(1.0f, escala, 1.0f);
  /* Efetua a operação de rotação em Z */
  glRotatef(rotaciona, 0.0f, 0.0f, 1.0f);
  /* desenha um simples retângulo  */
  glBegin(GL_QUADS);
    glVertex3f (0.025, 0.025, 0.0);
    glVertex3f (0.075, 0.025, 0.0);
    glVertex3f (0.075, 0.075, 0.0);
    glVertex3f (0.025, 0.075, 0.0);
  glEnd();
  /* Inicia o processo de desenho  através dos dados bufferizados  */
  glFlush ();
}
```

```
void init (void)
{
     /*   Define os parâmetros de cores para  obter a cor branca   */
     escala = 1;
     translada = 0;
     rotaciona = 0;
    /* Seleciona a cor de fundo para limpeza da tela */
    glClearColor (0.0, 0.0, 0.0, 0.0);
    /* inicializa os valores de visualização */
    glMatrixMode(GL_PROJECTION);
    /* Faz a matriz corrente ser inicializada com a matriz identidade (nenhuma
transformação é acumulada) */
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
/*   Função responsável pelo controle do teclado. Dependendo da tecla
pressionada : R,S,T, irá efetuar respectivamente as operações de Rotação,
Escala e Translação   */
```

```
void teclado(unsigned char tecla, int x, int y)
{
    switch (tecla) {
            case 'S':
            case 's':// Incrementa o valor do parâmetro de escala
                    escala = escala + 0.5;
                    break;
            case 'T':
            case 't':// Incrementa o valor do parâmetro de translacao
                    translada = translada + 0.05;
                    break;
            case 'R':
            case 'r':// Incrementa o valor do ângulo de rotação
                    rotaciona = rotaciona - 5.0;
                    break;
                        case 'O':
            case 'o':
                translada = 0.0;
                escala = 1.0;
                rotaciona = 0;
              break;
    }
    glutPostRedisplay();
}
```

```
/*    Funcão principal do programa.*/
int main(int argc, char** argv)
{

/*  Estabelece o modo de exibição a ser utilizado  pela janela a ser criada neste caso
utiliza-se de um buffer simples, ou seja, a apresentação  será imediata à execução.
Define o modo de cores como RGBA    */
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

  /*    Determina o tamanho em pixels da  janela a ser criada   */
  glutInitWindowSize (500, 500);

  /*   Estabelece a posição inicial para criação da janela   */
  glutInitWindowPosition (100, 100);

  /*   Cria uma janela com base nos parâmetros especificados nas funções
glutInitWindowSize e glutInitWindowPosition com o nome de título especificado em seu
argumento    */
  glutCreateWindow ("Exemplo 3 - Transformações");
  /*  Habilita a captura dos eventos de teclado     */
  glutKeyboardFunc(teclado);
```

```
/*  Especifica os parâmetros inicias para as variáveis de estado do OpenGL    */
   init ();

   // Associa a função display como uma função de callback
   glutDisplayFunc(display);

   /*  Inicia a execução do programa OpenGL. O programa irá executar num loop
infinito devendo o desenvolvedor especificar as condições de saída do mesmo
através de interrupções no próprio programa ou através de comandos de mouse
ou teclado como funções de callback */
   glutMainLoop();
   return 0;
}
```

Transformação de Visualização

Para alcançar um certa composição de cena na imagem final ou se move a câmara, ou se move o objeto na direção oposta

Os comandos de transformação de visão devem ser chamados antes de qualquer execução de transformações de modelagem.

O comando no OpenGL é :

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

O ponto de vista é dado por eyex, eyey, e eyez (onde está o olho) e centerx, centery, e centerz (para onde está olhando)

O upx, upy e upz indicam a direção que é para cima (câmara para cima ou deitada)

Exemplo de Transformação de Visualização

```
/* Exemplo4.c - Marcionílio Barbosa Sobrinho
 * Programa que apresenta as transformações de visualização em
uma cena
 * Objetivo:Demonstrar a utilização de transformação de
 *          visualização com o comando glLookAt()
 * Referência do Código: OpenGL Programming Guide – RedBook
 *                    Example 3-1 : Transformed Cube: cube.c
 */
#include <windows.h>
#include <GL/glu.h>
#include <GL/glut.h>

GLfloat eyex,eyey,eyez,centrox, centroy, centroz;
```

```
void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel (GL_FLAT);
   eyex = 0.0;
   eyey = 0.0;
   eyez = 5.0;
   centrox=0.0;
   centroy=0.0;
   centroz=0.0;
}

void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glLoadIdentity (); /* clear the matrix */
```

```
   /* viewing transformation */
   gluLookAt (eyex, eyey, eyez, centrox, centroy, centroz, 0.0, 1.0, 0.0);
   glutWireCube (1.0);
   glFlush ();
}

void reshape (int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
   glMatrixMode (GL_MODELVIEW);
}
```

```
/* Controla os eventos de teclado ao se pressionar as teclas  X, Y ou Z
os valores do parametro relativo a eye da    função glLookAt serão
modificados   */

void teclado(unsigned char tecla, int x, int y)
{
    switch (tecla) {
            case 'x':// Incrementa o valor de eyex
                    eyex = eyex + 0.5;
                    break;
            case 'y':// Incrementa o valor de eyey
                    eyey = eyey + 0.5;
                    break;
            case 'z':// Incrementa o valor de eyez
                    eyez = eyez + 0.5;
                    break;
```

```
            case 'X':// Incrementa o valor de centrox
                    centrox = centrox + 0.5;
                    break;
            case 'Y':// Incrementa o valor de centroy
                    centroy = centroy + 0.5;
                    break;
            case 'Z':// Incrementa o valor de centroz
                    centroz = centroz + 0.5;
                    break;
                 case 'O':
            case 'o':
                eyex = 0.0;             eyey = 0.0;             eyez = 5.0;
                centrox=0.0;            centroy=0.0;            centroz=0.0;
               break;
    }
    glutPostRedisplay();
}
```

```
/*Programa principal */

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Exemplo 4 - Visualização de Transformação");
  glutKeyboardFunc(teclado);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
  return 0;
}
```

Transformação de Projeção

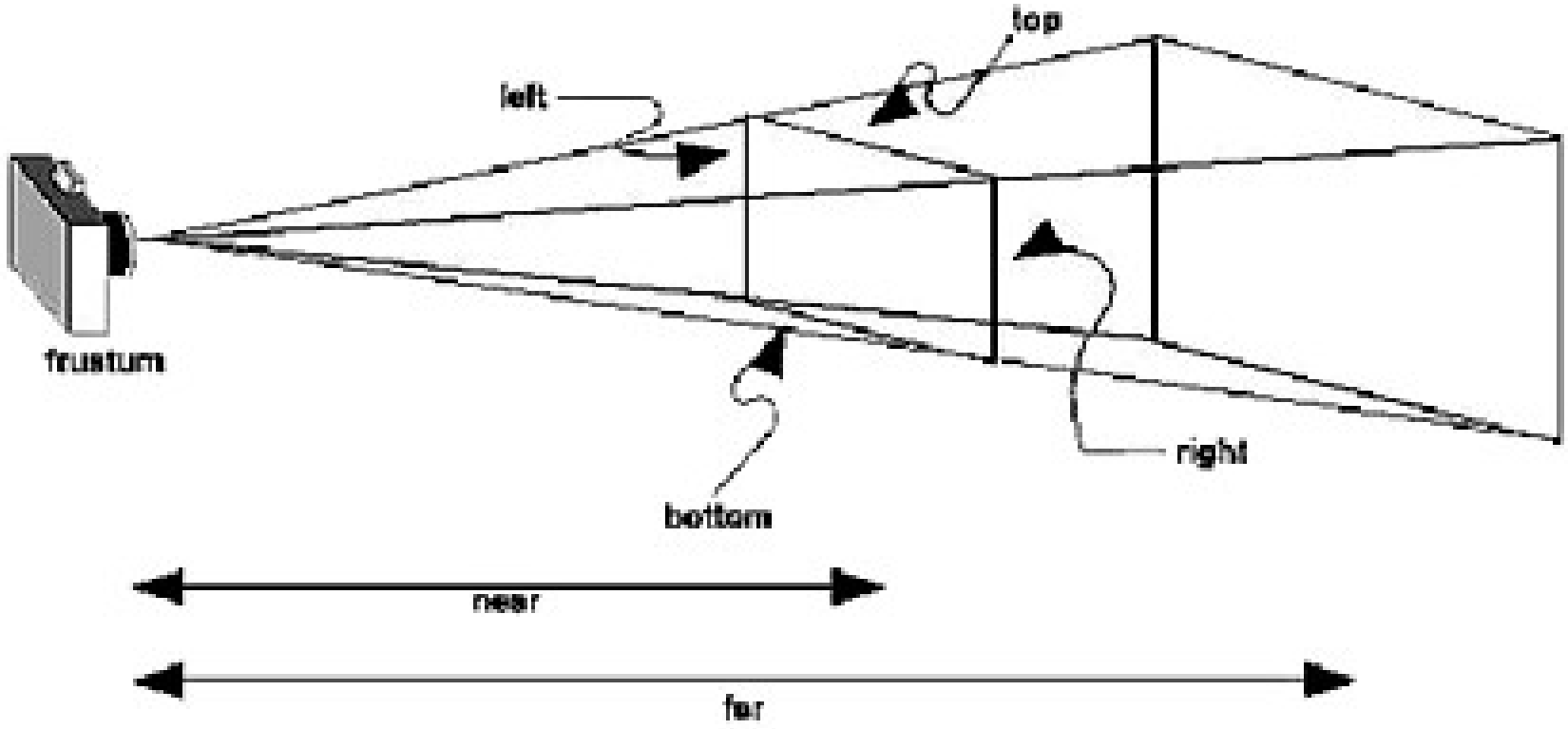
A transformação de projeção define um volume de visualização, que define como um objeto é projetado sobre a tela (quer dizer, usando perspectiva ou projeção ortográfica), e define os objetos que serão cortados da imagem final

Projeção Perspectiva

Quanto mais longe um objeto está da câmara, menor aparece na imagem

Isto acontece porque usa um frustum de uma pirâmide (uma pirâmide sem o topo)

Muito usado para obter um grau de realismo, pois é semelhante á forma de visualização do olho humano



O comando para definir um frustum é:
void **glFrustum**(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

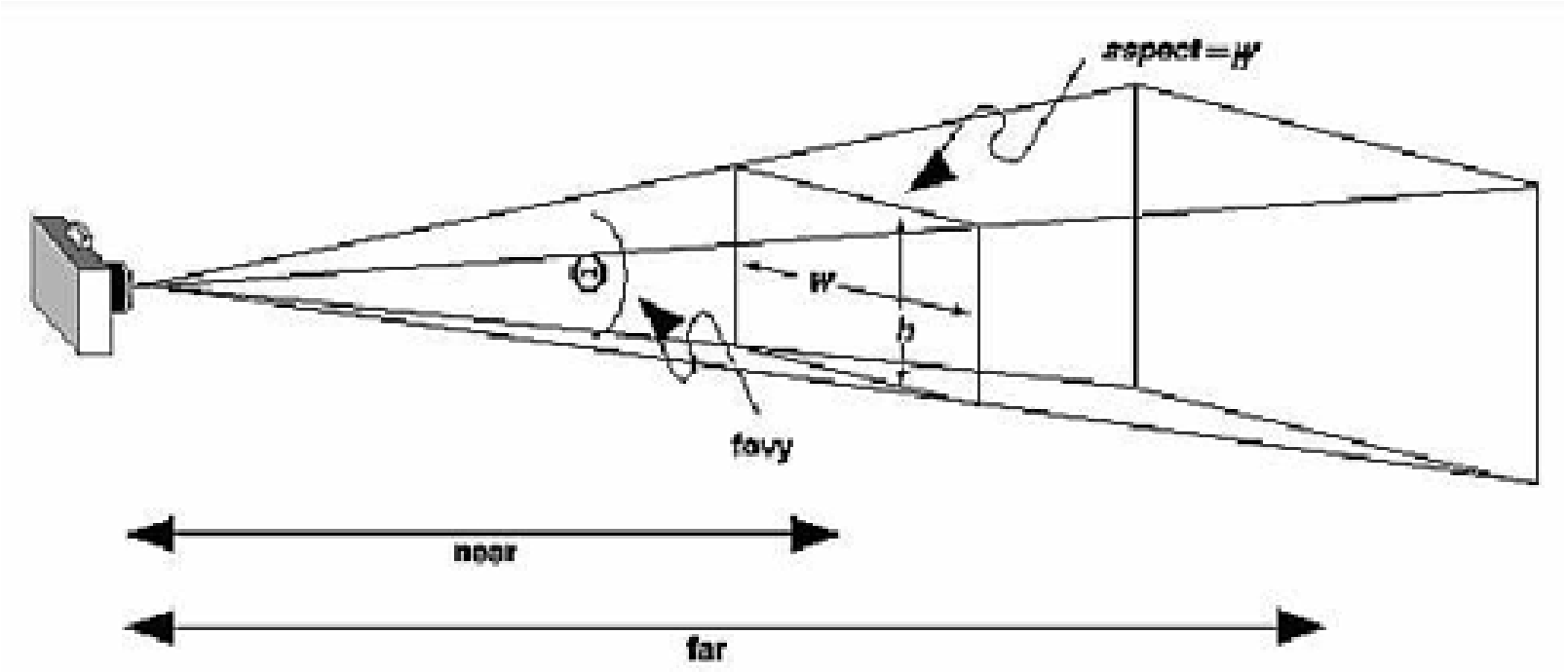
Cria uma matriz para um frustum de visão perspectiva e multiplica pela matriz atual

O frustum do volume de visão é definido pelos parâmetros: (left, botton, -near) e (right, top, -near) pelas coordenadas (x, y, z) específica do canto inferior esquerdo e canto de superior-direito do próximo plano de recorte ; near e far dão as distâncias do ponto de vista para o near e far do plano de recorte.

Eles sempre devem ser positivos

gluPerspective()

Também cria um volume de visão (da mesma forma que glFrustum()), mas de modo diferente



void gluPerspective(GLdouble fovy, GLdouble aspect,GLdouble near, GLdouble far);

Este comando cria uma matriz para um frustum de visão perspectiva simétrico e multiplica a matriz atual por isto.

fovy é o ângulo do campo de visão no plano de x-z; seu valor deve estar entre [0.0,180.0]. aspecto é a relação de aspecto do frustum, sua largura dividida por sua altura. near e far são distâncias entre o ponto de vista e os planos de corte, ao longo do z-eixo negativo.

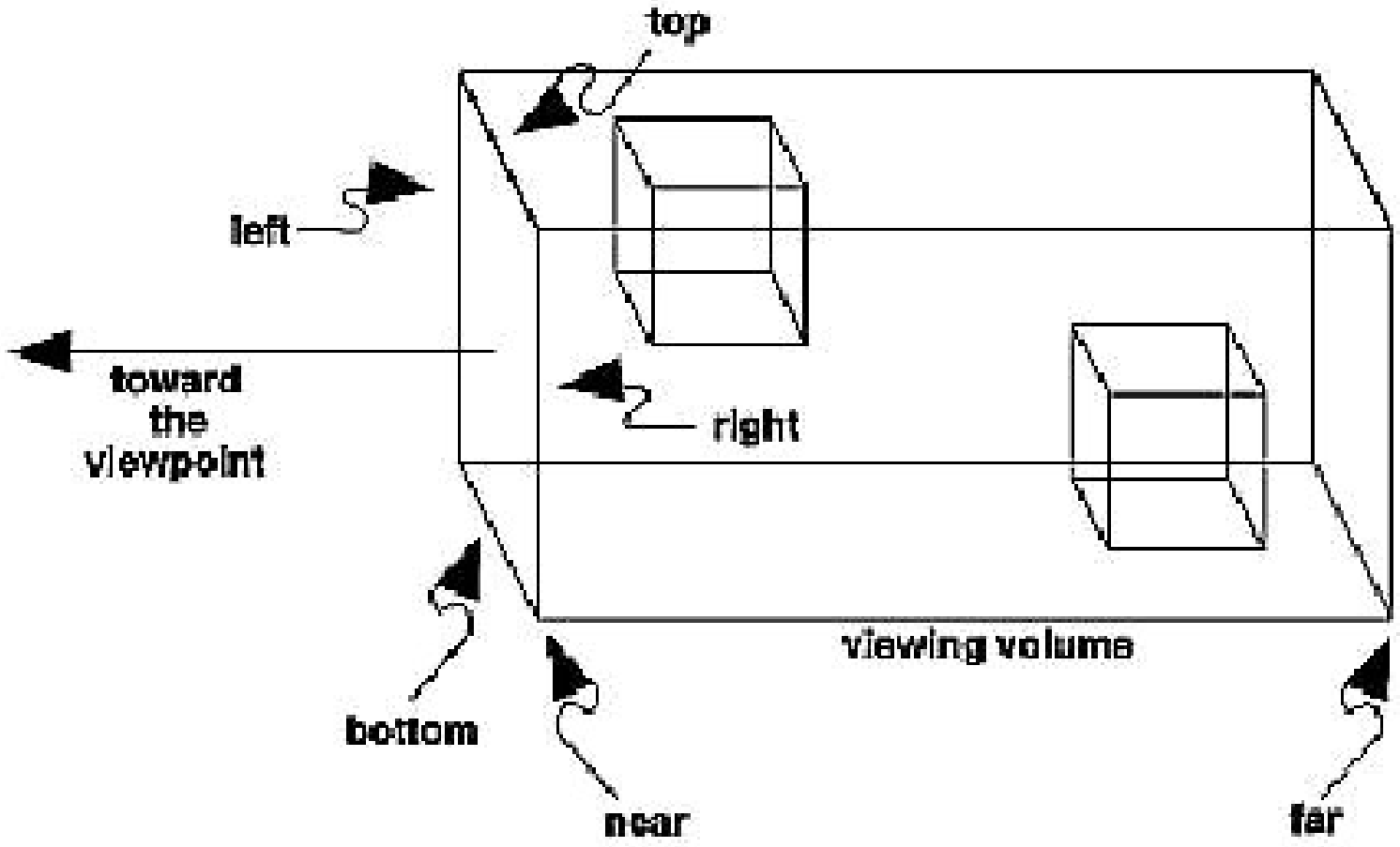
Eles devem sempre serem positivos.

Projeção Ortográfica

O volume de visão é um paralelepípedo retangular (uma caixa)

Diferentemente da projeção de perspectiva, a distancia do objeto à câmara não afeta como o tamanho do objeto

É usada para aplicações como criar plantas arquitetônicas e projetos CAD , onde é crucial manter os tamanhos atuais de objetos e ângulos entre eles da mesma forma como eles serão projetados.



void **glOrtho**(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far);

Este comando cria uma matriz para um volume visão paralelo ortográfico e multiplica a matriz atual por esta. (left, botton, -near) e (right, top, -near) são os pontos proximos ao plano de corte que é traçado no canto inferior esquerdo e canto de superior direito da janela de visualização , respectivamente.

Ambos near e far podem ser positivos ou negativos.

Transformação de Viewport

O viewport é a região da janela em que a imagem é desenhada

O viewport é medido em coordenadas de janela que refletem a posição relativa de pixels na tela do canto de inferior-esquerdo da janela

Porém, se o viewport não é fixado, ele será todo o retângulo da janela que é aberta

glViewport () é usado para escolher uma região de desenho menor (por exemplo, criar visões múltiplas na mesma janela)

Sintaxe do comando :

void **glViewport**(GLint x, GLint y, GLsizei width, GLsizei height);

Define um retângulo de pixel na janela na qual a imagem final é desenhada

O parâmetro (x, y) especifica o canto inferior-esquerdo do viewport, e largura e altura são o tamanho do retângulo do viewport

Os valores de viewport iniciais são (0, 0, winWidth, winHeight), com winWidth e winHeight são o tamanho da janela
(Caso não seja especificado o viewport).

Formas 3D pré-definidas no GLUT

O GLUT tem as seguintes formas geométricas pré-definidas :

- Esfera;
- Cubo;
- Cone;
- Toróide;
- Dodecaedro;
- Octaedro;
- Tetraedro;
- Icosaedro;
- Teapot

Esfera :

void glutSolidSphere(GLdouble radius, GLdouble slices, GLdouble stack)

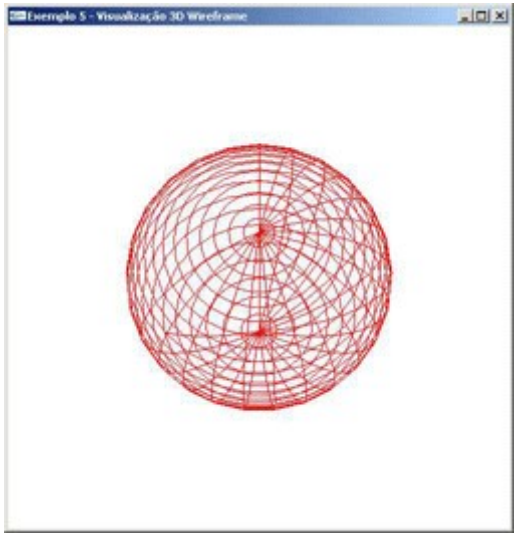
void glutWireSphere(GLdouble radius, GLdouble slices, GLdouble stack)

Parâmetros :

radius :Raio da Esfera

slices : Número de subdivisões ao redor do eixo Z (linhas de longitude)

stack : Número de subdivisões ao longo do eixo Z (a linhas de latitude).



Cubo

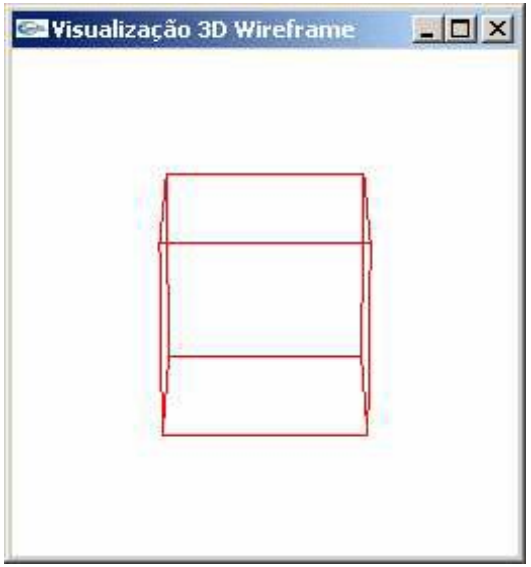
void glutSolidCube (GLdouble size)

void glutWireCube (GLdouble size)

Parâmetros :

size :Tamanho do cubo

Exemplo : glutWireCube(20);



Cone

void glutSolidCone(GLdouble base ,GLdouble height,Glint slices,Glint stacks)

void glutWireCone(GLdouble base ,GLdouble height,Glint slices,Glint stacks)

Parâmetros :

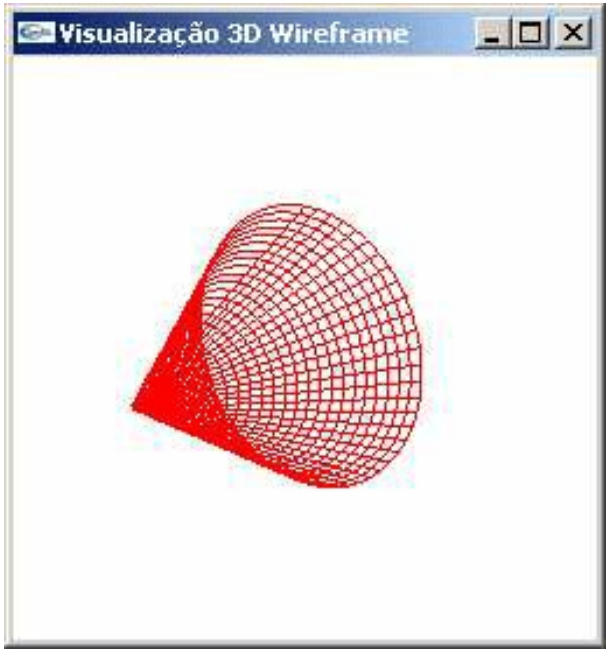
base:Raio da base do cone

height: A altura do cone

slices : Número de subdivisões ao redor do eixo Z (linhas de longitude)

stack : Número de subdivisões ao longo do eixo Z (a linhas de latitude).

Exemplo : glutWireCone(24,40,55,25);



Toroide

void glutSolidTorus(GLdouble innerRadius,GLdouble outerRadius,Glint nsides,Glint rings)

void glutWireTorus(GLdouble innerRadius,GLdouble outerRadius,Glint nsides,Glint rings)

Parâmetros :

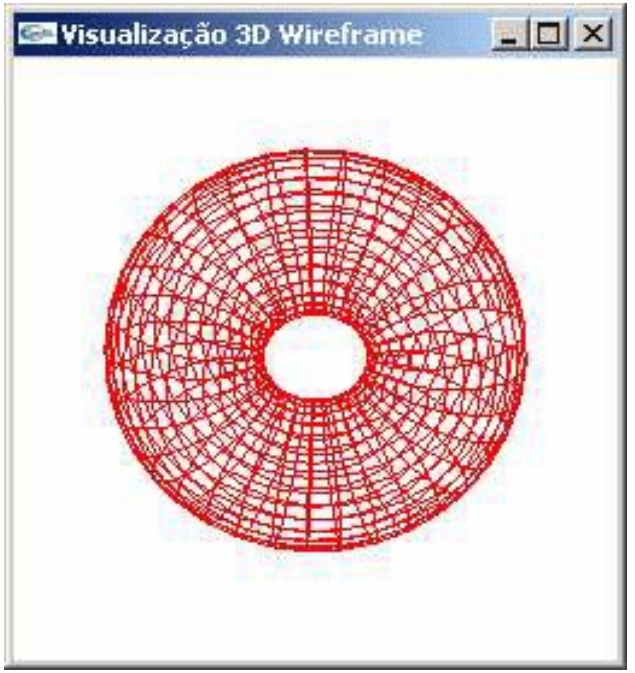
innerRadius:Raio interno do toróide.

outerRadius: Raio externo do toróide

nsides: Número de lados para cada seção radial.

rings: Número de subdivisões radiais do toróide.

Exemplo : glutWireTorus(15,25,30,35);



Dodecaedro :

void glutSolidDodecahedron ()

void glutWireDodecahedron ()

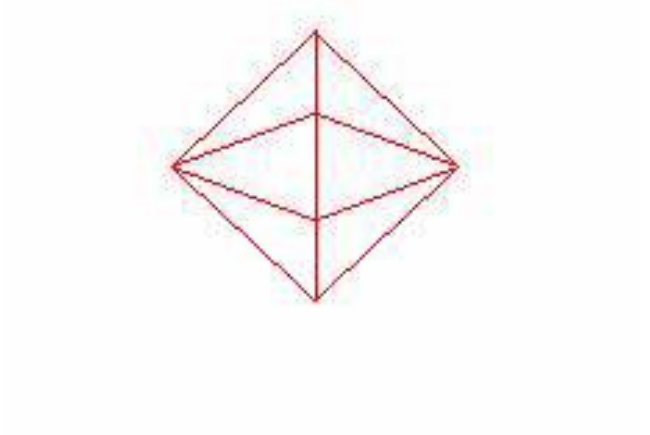
Exemplo : glutWireDecahedron ();

Octaedro

void glutSolidOctahedron()

void glutWireOctahedron()

Exemplo : glutWireOctahedron()

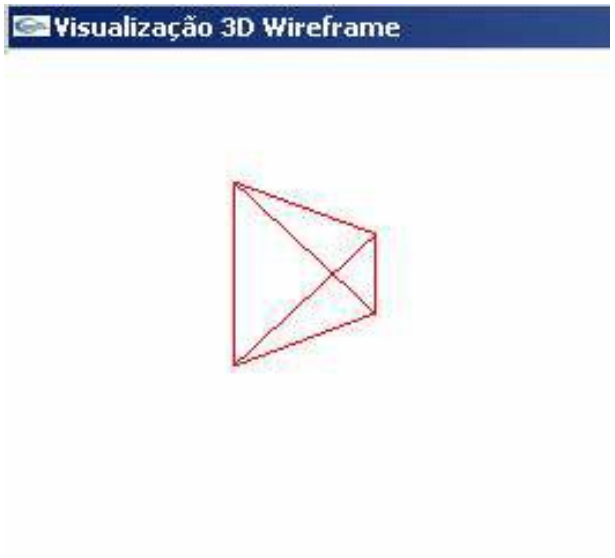


Tetraedro

void glutSolidTetrahedron()

void glutWireTetrahedron()

Exemplo : glutWireTetrahedron()

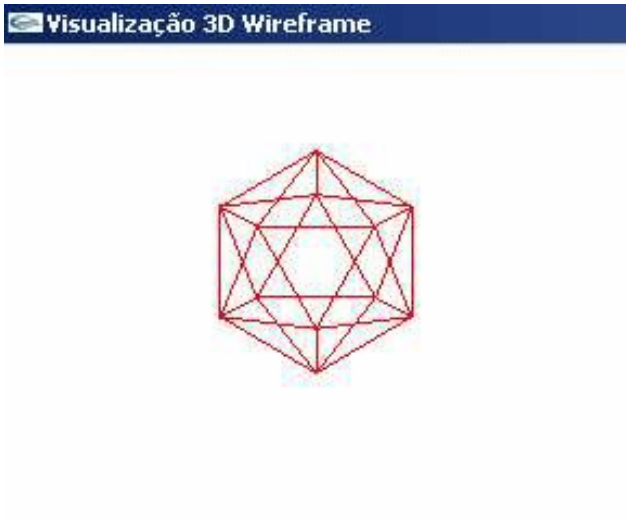


Icosaedro

void glutSolidIcosahedron(),

void glutWireIcosahedron()

Exemplo : glutWireIcosahedron();



Teapot

void glutSolidTeapot (GLdouble size);

void glutWireTeapot (GLdouble size);

Parâmetro :

size : Tamanho do “teapot”.

Exemplo : glutWireTeapot(50.0);