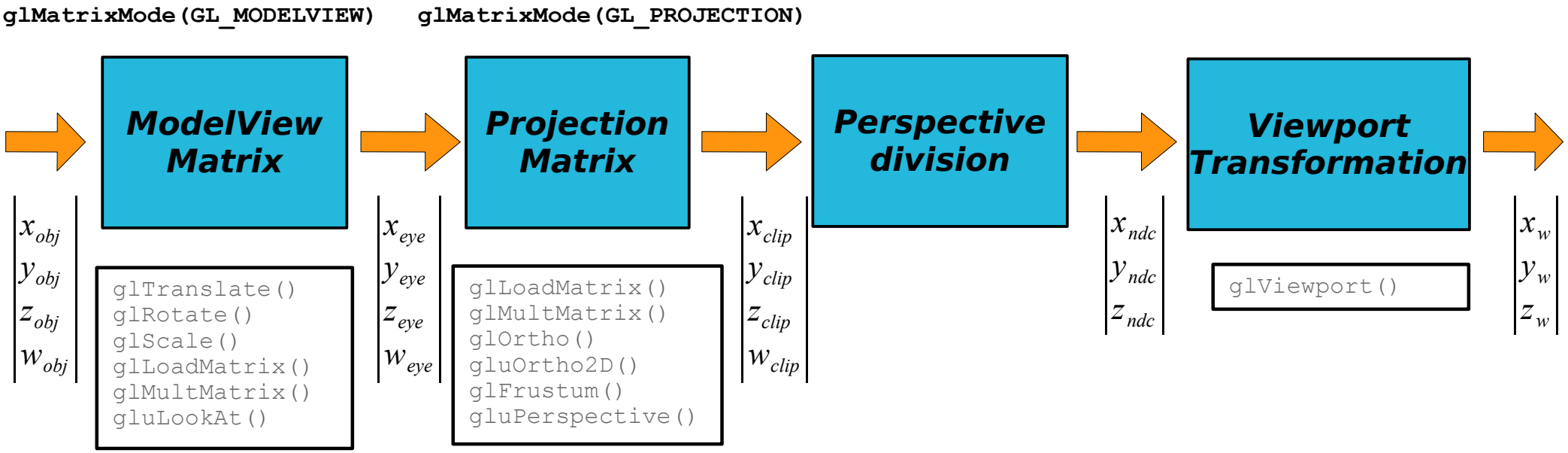
Unidade 7 - Transformações em OpenGL

IME 04-10842
Computação Gráfica
Professor Guilherme Mota
Professor Gilson Costa

Transformações dos Vértices



Modos de Matriz

```
void glMatrixMode(GLenum  mode);
```

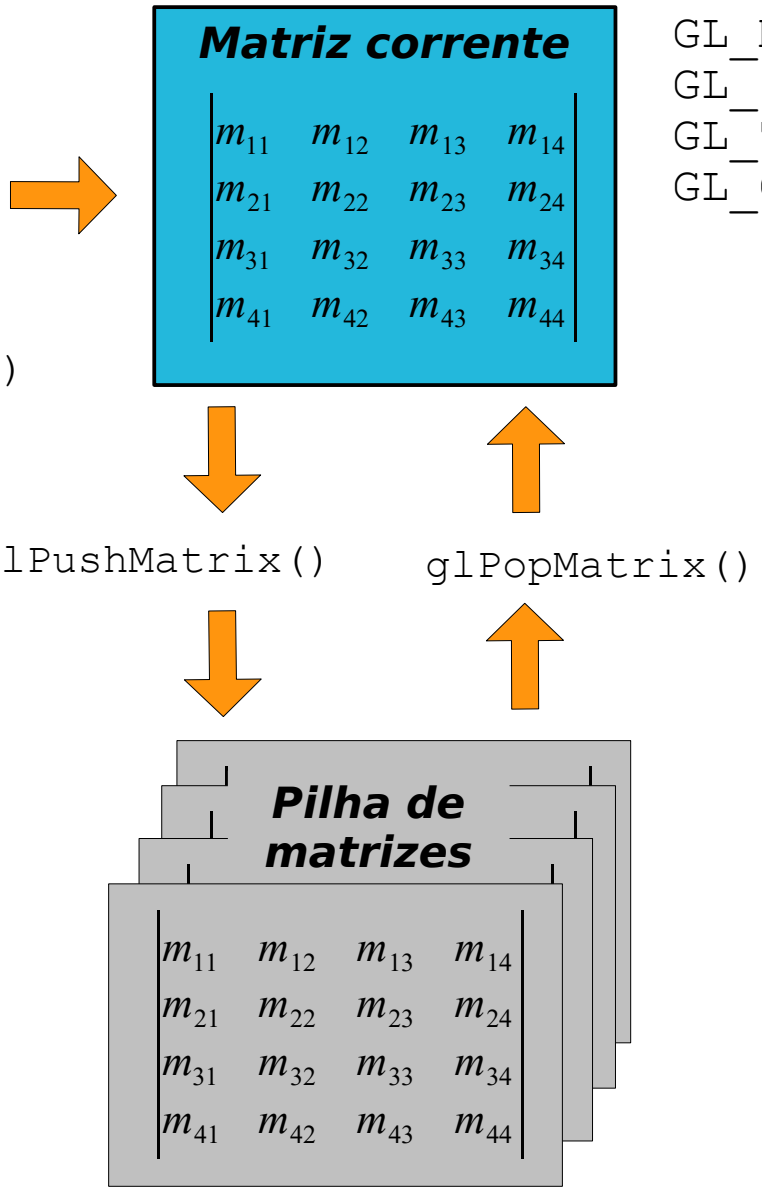
- Especifica que pilha de matrizes será utilizada nas próximas operações.

- mode:
  - GL_MODELVIEW
  - GL_PROJECTION
  - GL_TEXTURE
  - GL_COLOR

Pilha de Matrizes

Combinação:
glMultMatrix()
glTranslate()
glRotate()
glScale()

Substituição:
glLoadMatrix()
glLoadIdentity()



GL_MODELVIEW
GL_PROJECTION
GL_TEXTURE
GL_COLOR

Operações com Matrizes

Operações nas Pilhas de Matrizes

$MC = Matriz\ Corrente$
$MT = Matriz\ Fornecida$

- Combinação $MC = MC \cdot MT$
  - `glMultMatrix()`
  - `glTranslate()`
  - `glRotate()`
  - `glScale()`

- Substituição $MC = MT$
  - `glLoadIdentity`
  - `glLoadMatrix`

Combinação

```
void glMultMatrixf(const GLfloat *  m)
```

- Muda o valor da matriz corrente (MC) da pilha de matrizes em uso
- `*m` - vetor de 16 posições contendo os elementos da matriz de transformação (MT)
- A matriz de transformação é formada coluna a coluna

Matriz de transformação

$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix}$$

- MC = MC . MT

Combinação

```
void glTranslatef(GLfloat x, GLfloat
 y, GLfloat z)
```

- Muda o valor da matriz corrente (MC) da pilha de matrizes em uso impondo uma translação
- `x`, `y` e `z` – momentos de translação
- A matriz de translação MT é :

Matriz de translação

$$\begin{vmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- MC = MC . MT

Combinação

```
void glRotatef(GLfloat  angle,
GLfloat  x,  GLfloat  y,  GLfloat  z);
```

- Muda o valor da matriz corrente (MC) da pilha de matrizes em uso
- `x`, `y` e `z` - vetor de referência
- `angle` – ângulo da rotação
- A matriz de rotação é dada pela referência

Matriz de rotação

$$\begin{vmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ yx(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ xz(1-c)-ys & yz(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$c = \cos(angle)$$
$$s = \text{sen}(angle)$$

- MC = MC . MT

Combinação

```
void glScalef(GLfloat  x,  GLfloat  y,
 GLfloat  z)
```

- Muda o valor da matriz corrente (MC) da pilha de matrizes em uso
- `x`, `y` e `z` - fatores de escala
- A mudança de escala (S) é dada por:

Mudança de escala

$$\begin{vmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- MC = MC . S

Substituição

`void glLoadIdentity(void)`

- Muda matriz corrente (MC) da pilha de matrizes em uso
- $MC = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$

Substituição

```
void glLoadMatrixf(const GLfloat *  m)
```

- Muda a matriz corrente (MC) da pilha de matrizes em uso
- `*m` - vetor de 16 posições contendo os elementos da nova matriz corrente (MC)
- A matriz de transformação é formada coluna a coluna

Matriz de Corrente

$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix}$$

Modelagem

Modelagem

- No OpenGL “modelagem” está relacionada ao conjunto de transformações que mapeiam o espaço/referencial local do objeto para o referencial global (espaço do mundo).
- Essas transformações devem ser aplicadas na matriz `GL_MODELVIEW`

```
glMatrixMode(GL_MODELVIEW)
```

MC = MC . MT

- `glTranslate()`
- `glRotate()`
- `GlScale()`
- `glMultiMatrix()`

MC = MT

- `GlLoadIdentity()`
- `glLoadMatrix()`

Visualização

Visualização

- No OpenGL “visualização” está relacionada ao conjunto de transformações que mapeiam o espaço do mundo para o espaço da câmera (ou do olho).
- Essas transformações devem ser aplicadas na matriz `GL_MODELVIEW`

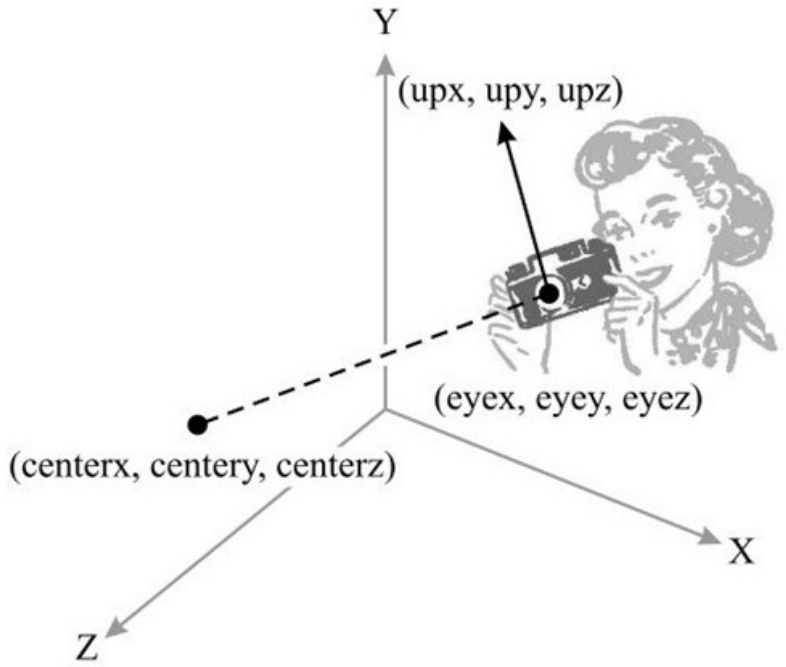
```
glMatrixMode(GL_MODELVIEW)

glLookAt(eyeXYZ, centerXYZ, upXYZ)
```

Visualização

`void glLookAt(eyeXYZ, centerXYZ, upXYZ)`

- `eye` - coordenadas do centro de perspectiva da câmara
- `center` - coordenadas do ponto de referência para a direção de visualização
- `up` - direção do eixo vertical da câmara



Visualização

```
void glLookAt(eyeXYZ, centerXYZ, upXYZ)
```

$$f = c - e$$
$$f' = \frac{f}{|f|}$$
$$u' = \frac{u}{|u|}$$
$$s = f \times u'$$
$$u'' = s \times f$$

$$M = \begin{vmatrix} s_x & s_y & s_z & 0 \\ u''_x & u''_y & u''_z & 0 \\ -f'_x & -f'_y & -f'_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Projeção

Projeção

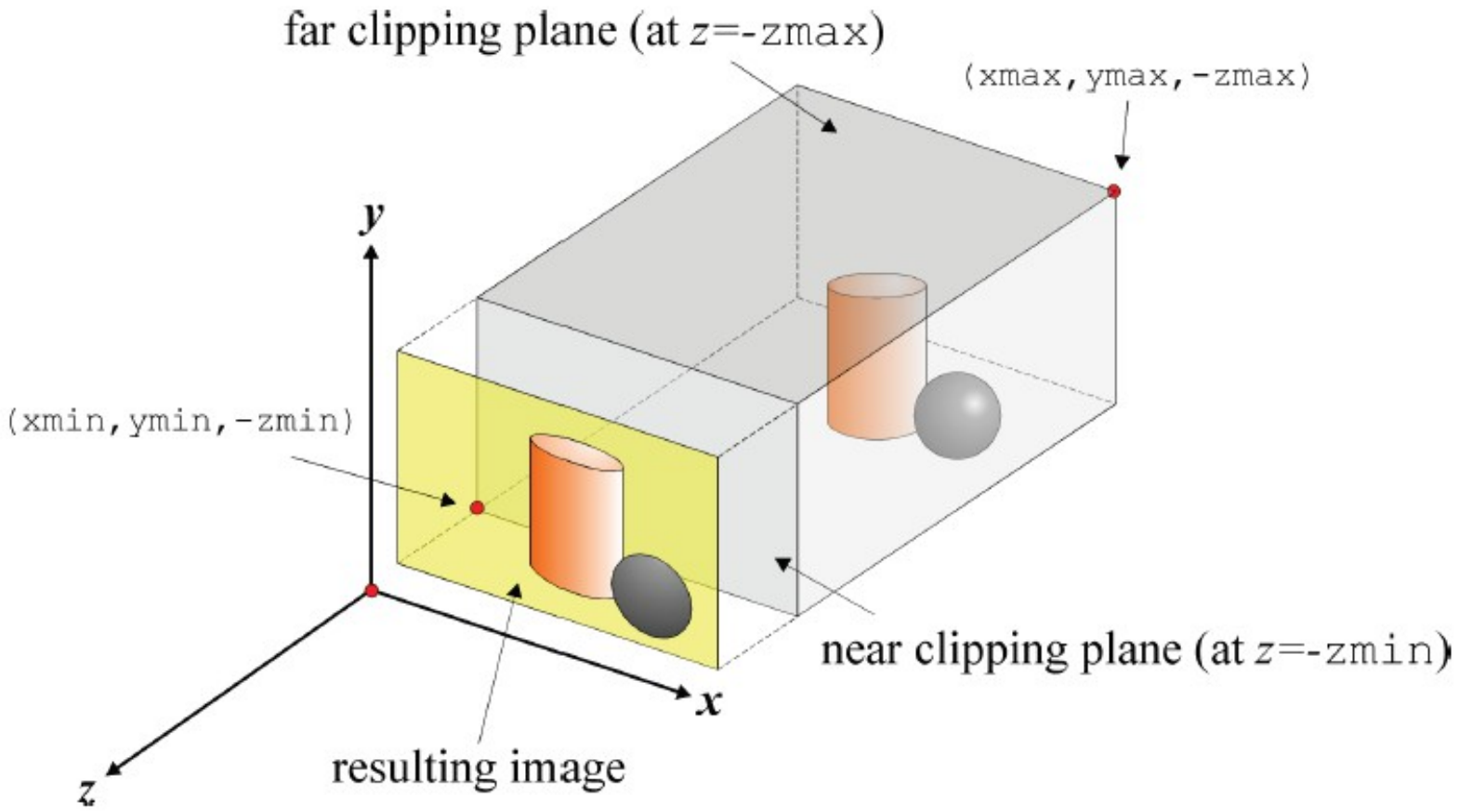
- “Projeção” está relacionada a transformações que fazem o mapeamento do espaço da câmera para o espaço da tela.
- Essas transformações devem ser aplicadas na matriz `GL_PROJECTION`

```
glMatrixMode(GL_PROJECTION)

glOrtho(left,right,bottom,top,near,far)
glFrustum(left,right,bottom,top,near,far)
gluPerspective(fovy, aspect, near, far)
```

Projeção

```
void glOrtho(xmin,xmax,ymin,ymax,zmin,zmax);
```



Projeção

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

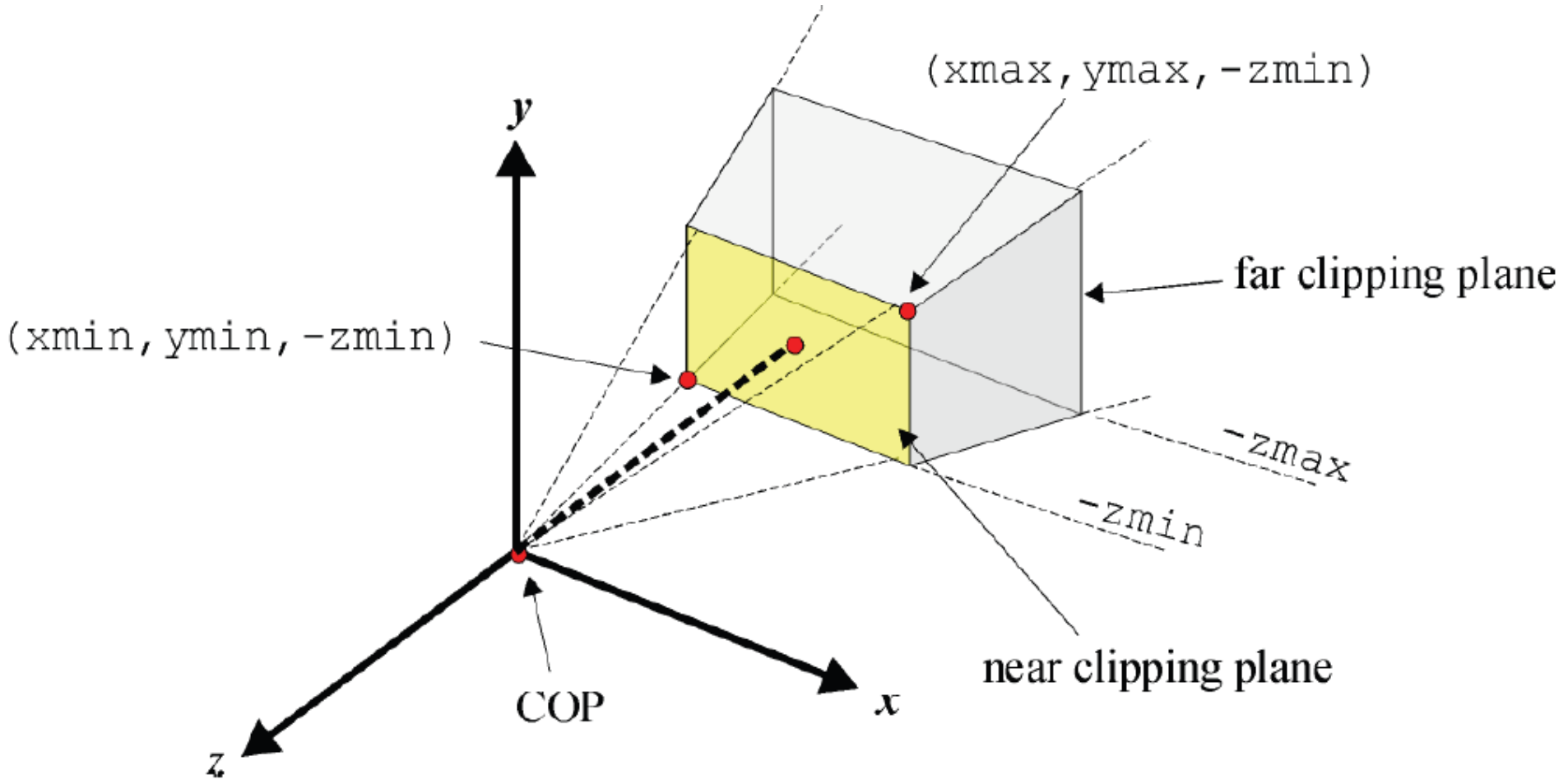
$$M = \begin{vmatrix} \frac{2}{right-left} & 0 & 0 & A \\ 0 & \frac{2}{top-bottom} & 0 & B \\ 0 & 0 & \frac{-2}{far-near} & C \\ 0 & 0 & -1 & 0 \end{vmatrix}$$

$$A = -\frac{right+left}{right-left} \qquad C = -\frac{far+near}{far-near}$$

$$B = -\frac{top+bottom}{top-bottom}$$

Projeção

```
void glFrustum(xmin,xmax,ymin,ymax,zmin,zmax);
```



Projeção

```
void glFrustum(left,right,bottom,top,near,far);
```

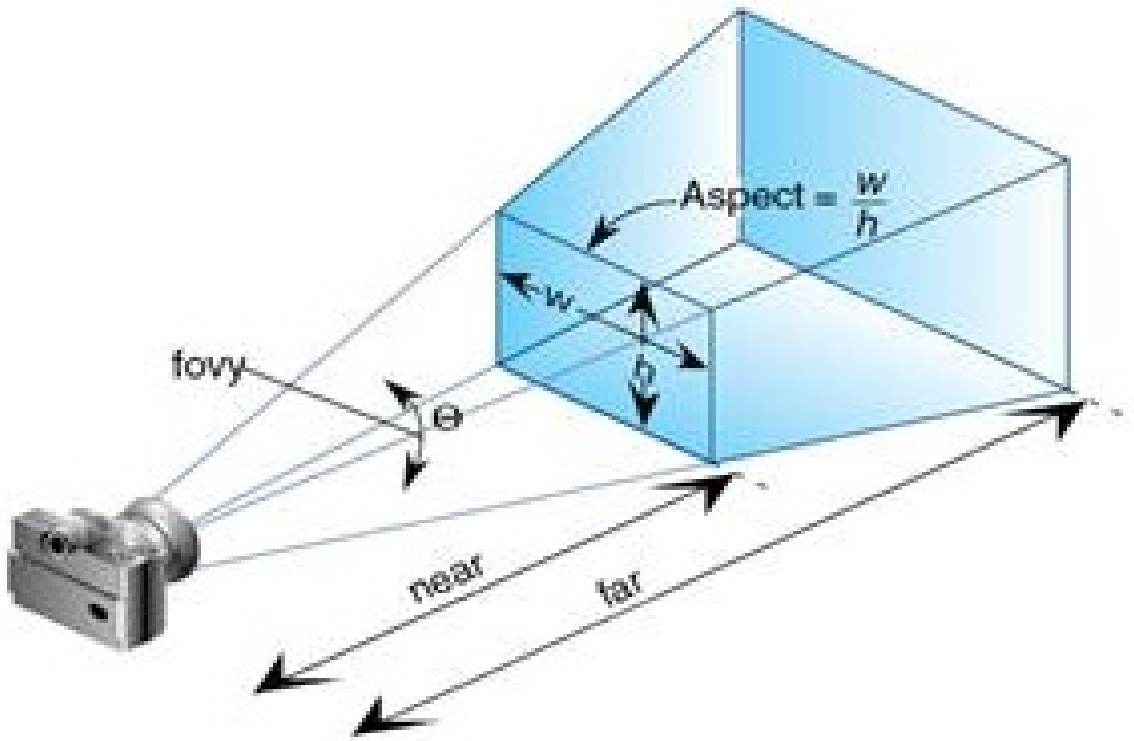
$$M = \begin{vmatrix} \frac{2\,near}{right-left} & 0 & A & 0 \\ 0 & \frac{2\,near}{top-bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{vmatrix}$$

$$A = \frac{right+left}{right-left} \qquad C = -\frac{far+near}{far-near}$$

$$B = \frac{top+bottom}{top-bottom} \qquad D = -\frac{2\ far\ near}{far-near}$$

Projeção

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



Projeção

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

$$M = \begin{vmatrix} \frac{f}{aspect} & 0 & A & 0 \\ 0 & f & B & 0 \\ 0 & 0 & \frac{far+near}{near-far} & \frac{2\ far\ near}{near-far} \\ 0 & 0 & -1 & 0 \end{vmatrix}$$

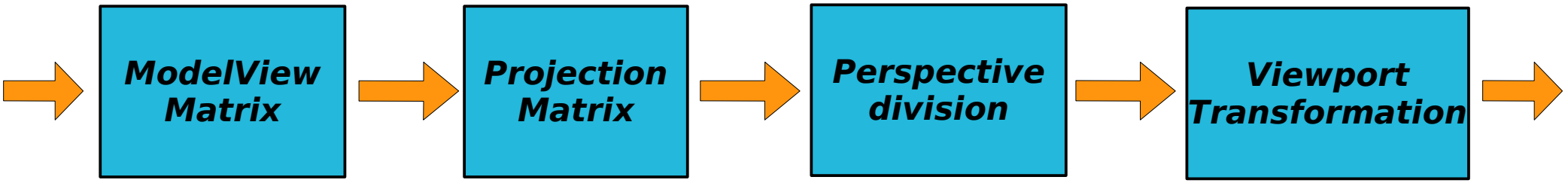
$$f = \cot\left(\frac{fovy}{2}\right)$$

Sequência de Transformações

Sequência de Transformações de Vértices

$$\begin{vmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{vmatrix} = M_{ModelView} \begin{vmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{vmatrix}$$

$$\begin{vmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{vmatrix} = \begin{vmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{vmatrix}$$



$$\begin{vmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{vmatrix}$$

$$\begin{vmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{vmatrix} = M_{Projection} \begin{vmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{vmatrix}$$

$$\begin{vmatrix} x_w \\ y_w \\ z_w \end{vmatrix} = \begin{vmatrix} \frac{w}{2} x_{ndc} + (x + \frac{w}{2}) \\ \frac{h}{2} y_{ndc} + (y + \frac{h}{2}) \\ \frac{f-n}{2} z_{ndc} + \frac{(f+n)}{2} \end{vmatrix}$$

```
glViewport(x, y, w, h)
glDepthRange(n, f)
```

Exemplo 1
Pontos de Vista

Exemplo 1 (Pontos de Vista)

```
#include <stdio.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

// variaveis globais
GLint Width = 800;
GLint Height = 800;
static GLfloat spin_z = 0.0;
static GLfloat spin_y = 0.0;
char objectId = 'b';

// prototipos
void drawObject(void); // desenha o bule
void spinDisplay(void); // rotaciona bule
void initLighting(void); // define a fonte de luz
void reshape(int width, int height);    // callback de redesenho da
                                        // janela glut
void display(void); // callback de desenho
void keyboard(unsigned char key, int x, int y); // callback de teclado
```

Exemplo 1 (Pontos de Vista)

```
// programa principal
int main(int argc, char** argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
      glutInitWindowSize(Width, Height);
      glutCreateWindow("Bule em 4 vistas");

      initLighting();

      glutDisplayFunc(display);
      glutReshapeFunc(reshape);
      glutKeyboardFunc(keyboard);

      glutMainLoop();

      return 0;
}
```

Exemplo 1 (Pontos de Vista)

```
// desenha o objeto
void drawObject(void)
{
      switch (objectId)
      {
      case 'b':
          glColor3f(1.0, 1.0, 1.0);
          glutSolidTeapot(2.0);
          break;
      case 'o':
          glColor3f(1.0, 1.0, 1.0);
          glutSolidDodecahedron();
          break;
      default:
          return;
      }
}
```

Exemplo 1 (Pontos de Vista)

```
// rotaciona objeto
void spinDisplay(void)
{
      if (spin_z > 360.0)
          spin_z = spin_z - 360.0;
      if (spin_y > 360.0)
          spin_y = spin_y - 360.0;
      glutPostRedisplay();
}

// callback de redesenho da janela glut
void reshape(int width, int height)
{
      Width = width;
      Height = height;
      //glViewport(0, 0, width, height);
      glutPostRedisplay();
}
```

Exemplo 1 (Pontos de Vista)

```
// define a fonte de luz (LIGHT0)
void initLighting(void)
{
      GLfloat lightposition[] = { 3.0, -3.0, 3.0, 0.0 };

      glDepthFunc(GL_LESS);
      glEnable(GL_DEPTH_TEST);

      glEnable(GL_LIGHT0);
      glEnable(GL_LIGHTING);
      glLightfv(GL_LIGHT0, GL_POSITION, lightposition);
      glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

      glEnable(GL_COLOR_MATERIAL);

      glClearColor(0.0, 1.0, 1.0, 0.0);
}
```

Exemplo 1 (Pontos de Vista)

```
// callback de teclado
void keyboard(unsigned char key, int x, int y)
{
      cout << key;
      switch (key) {
      case 27:
          Exit(0);  break;
      case 'a':
          spin_z = spin_z - 2.0; spinDisplay(); break;
      case 's':
          spin_z = spin_z + 2.0; spinDisplay(); break;
      case 'z':
          spin_y = spin_y - 2.0; spinDisplay(); break;
      case 'w':
          spin_y = spin_y + 2.0; spinDisplay(); break;
      case 'o':
          if (objectId == 'b') objectId = 'o'; else objectId = 'b';
          glutPostRedisplay(); break;
      default:
          Return; }
}
```

Exemplo 1 (Pontos de Vista)

```
// callback de desenho
void display(void)
{
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

      // viewport topo/esquerda
      glViewport(0, Height / 2, Width / 2, Height / 2);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-3.0, 3.0, -3.0, 3.0, 1.0, 5.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      gluLookAt(0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0);
      drawObject();
```

Exemplo 1 (Pontos de Vista)

```
// viewport topo/direita
glViewport(Width / 2, Height / 2, Width / 2, Height / 2);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-3.0, 3.0, -3.0, 3.0, 1.0, 50.0);
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadIdentity();
gluLookAt(5.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
drawObject();

// viewport baixo/esquerda
glViewport(0, 0, Width / 2, Height / 2);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-3.0, 3.0, -3.0, 3.0, 1.0, 50.0);
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadIdentity();
gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
drawObject();
```

Exemplo 1 (Pontos de Vista)

```
        // viewport baixo/direita
        glViewport(Width / 2, 0, Width / 2, Height / 2);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glFrustum(-3.0, 3.0, -3.0, 3.0, 3.0, 6.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        glRotatef(45.0, 1.0, 0.0, 0.0);
        glRotatef(spin_z, 0.0, 0.0, 1.0);
        glRotatef(spin_y, 0.0, 1.0, 0.0);
        drawObject();

        glutSwapBuffers();
}
```

Exemplo 1 (Pontos de Vista)

Experimente:

- Aperte as teclas “a”, “s”, “w”, “z” para rodar o bule.
- Em tempo de execução, altere o tamanho da janela criada pelo programa.
- Altere o programa para que o bule não apareça truncado no viewport superior direito e nem no inferior esquerdo (quando rotacionado).
- Aperte a tecla “o” para trocar o objeto.

Primitivas 3D

Primitivas 3D GLUT

- A função `glutSolidTeapot(GLdoouble size);` é usada para desenhar um teapot (bule de chá) sólido.
- O parâmetro size indica um raio aproximado do teapot. Uma esfera com este raio irá "envolver" totalmente o modelo.
- Assim como a função teapot, a biblioteca GLUT também possui funções para desenhar outros objetos 3D.

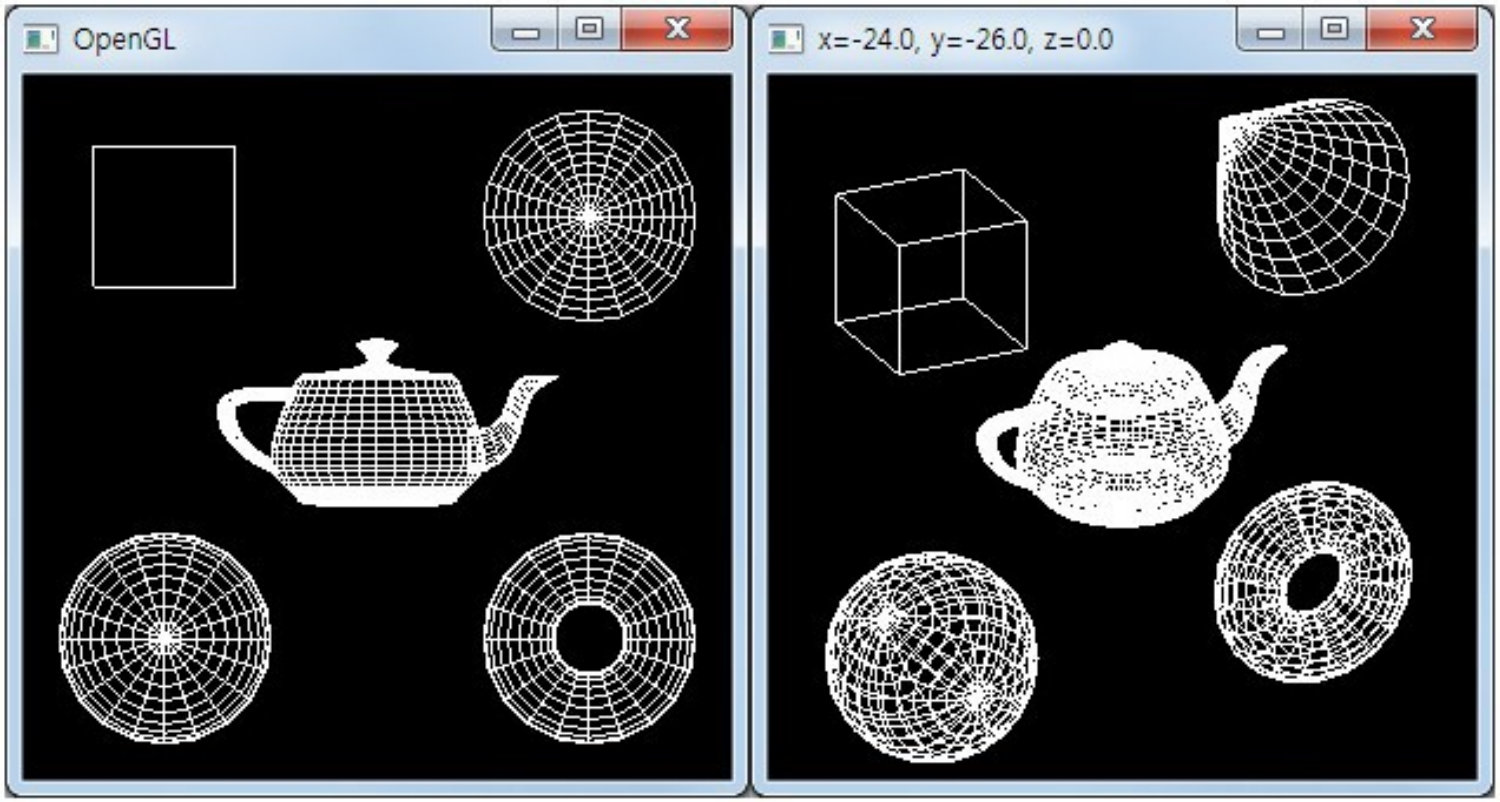
Primitivas 3D GLUT

Wire-frame:

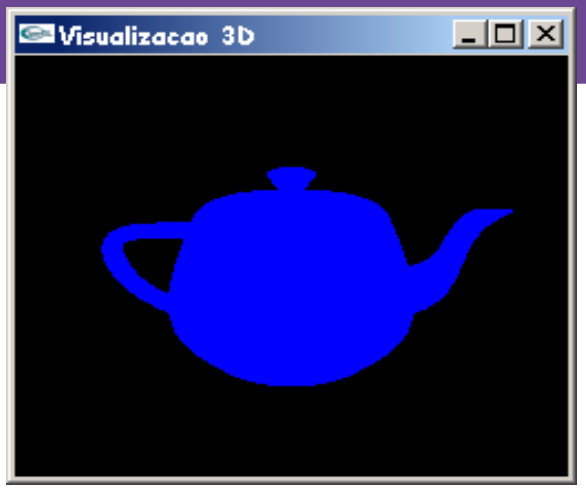
- `void glutWireCube(GLdouble size);`
- `void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);`
- `void glutWireCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks);`
- `void glutWireTorus(GLdouble innerRadius, Gldouble outerRadius, GLint nsides, GLint rings);`
- `void glutWireIcosahedron(void);`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

Primitivas 3D GLUT

Wire-frame:



Primitivas 3D GLUT



Sólidos:

- `void glutSolidCube(GLdouble size);`
- `void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);`
- `void glutSolidCone(GLdouble radius, Gldouble height, GLint slices, GLint stacks);`
- `void glutSolidTorus(GLdouble innerRadius, Gldouble outerRadius, GLint nsides, GLint rings);`
- `void glutSolidIcosahedron(void);`
- `void glutSolidOctahedron(void);`
- `void glutSolidTetrahedron(void);`
- `void glutSolidDodecahedron(GLdouble radius);`

Iluminação
Materiais

Iluminação - Materiais

- Definição do modelo de colorização: `glShadeModel()`.
- Dois modelos: um polígono preenchido/sólido 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).

Iluminação - Materiais

- Quando objetos 3D sólidos são exibidos, é importante desenhar os objetos que estão mais próximos do observador (ou posição da câmera), eliminando objetos que ficam "escondidos", ou "parcialmente escondidos".
- OpenGL possui um *depth buffer* que trabalha através da associação de uma profundidade, ou distância, do plano de visualização com cada pixel da window.

Iluminação - Materiais

- Inicialmente, os valores de profundidade são especificados para serem o maior possível através do comando `glClear(GL_DEPTH_BUFFER_BIT)`.
- 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, o pixel é desenhado e o valor de profundidade é atualizado.
- Caso contrário o pixel é desprezado.

Iluminação - Materiais

- Em OpenGL a cor de uma fonte de luz é caracterizada pela quantidade de vermelho (R), verde (G) e azul (B) que ela emite.
- 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).

Iluminação - Materiais

- 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 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 luz incidente em todas as direções, ou refletir uma parte da luz incidente numa única direção, tal com um espelho.

Iluminação - Materiais

- 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.

Iluminação - Materiais

- A cor do material de um objeto depende da porcentagem de luz vermelha, verde e azul incidente que ele reflete.
- Para os materiais, os números correspondem às proporções refletidas destas cores. Se R=1, G=0.5 e B=0 o material reflete toda luz vermelha incidente, metade da luz verde e nada da azul.
- 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.

Exemplo 2
Iluminação

Exemplo 2 (Iluminação)

```
#include <stdlib.h>
#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);
      // Especifica sistema de coordenadas de projeção
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(angle, fAspect, 0.4, 500);
      // Especifica sistema de coordenadas do modelo
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      gluLookAt(0, 80, 200, 0, 0, 0, 0, 1, 0);
      // Desenha o teapot com a cor corrente (solid)
      glColor3f(0.6f, 0.4f, 0.1f);
      glutSolidTeapot(50.0f);

      glutSwapBuffers();
}
```

Exemplo 2 (Iluminação)

```
// Inicializa parâmetros de iluminação
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);
```

Exemplo 2 (Iluminação)

```
        // 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;
}
```

Exemplo 2 (Iluminação)

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

Exemplo 2 (Iluminação)

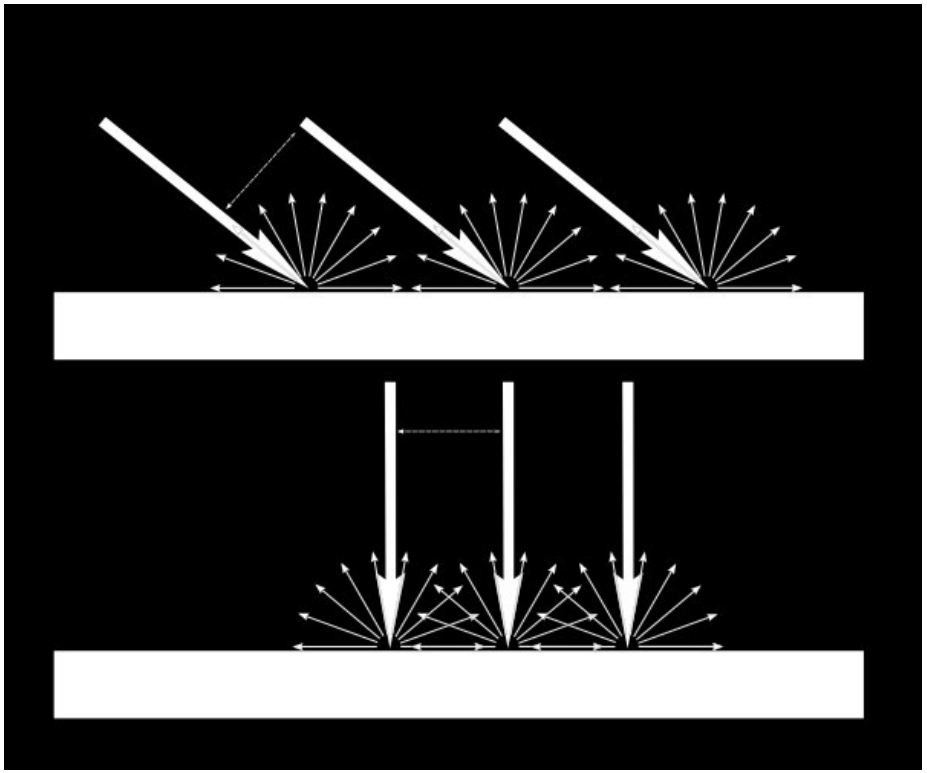
```
// Programa Principal
int main(int argc, char** argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
      glutInitWindowSize(400, 350);
      glutCreateWindow("Bule Iluminado");
      glutDisplayFunc(Desenha);
      glutReshapeFunc(AlteraTamanhoJanela);
      glutMouseFunc(GerenciaMouse);
      Inicializa();
      glutMainLoop();
}
```

Iluminação - Normais

- As primitivas 3D do GLUT e GLU já vem preparadas para a iluminação.
- Obviamente os parâmetros de iluminação da cena e dos materiais devem ser definidos pelo programador.
- Mas quando se cria objetos 3D a partir de primitivas básicas do OpenGL, deve-se estabelecer para cada vértice que faz parte de uma superfície plana do objeto 3D seu vetor normal.

Iluminação - Normais

- O vetor normal à superície: auxilia o OpenGL a saber a quantidade de luz que incide sobre a superfície.



Exemplo 3
Normais

Exemplo 3 (Normais)

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

// variaveis globais
GLfloat xRotated, yRotated, zRotated; //angulos de rotacao
bool normal; //indica se normais estão sendo usadas

// define a fonte de luz (LIGHT0)
void initLighting(void)
{
      GLfloat lightposition[] = { 3.0, 3.0, 3.0, 0.0 };
      glDepthFunc(GL_LESS);
      glEnable(GL_DEPTH_TEST);
      glEnable(GL_LIGHT0);
      glEnable(GL_LIGHTING);
      glLightfv(GL_LIGHT0, GL_POSITION, lightposition);
      glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
      glEnable(GL_COLOR_MATERIAL);
      glClearColor(0.0, 0.0, 1.0, 0.0);
}
```

Exemplo 3 (Normais)

```
// callback para redimensionar janela glut
void resizeWindow(int x, int y)
{
      if (y == 0 || x == 0) return;
      //define projecao
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glFrustum(-1.5, 1.5, -1.5, 1.5, 3.0, 10.0);
      glMatrixMode(GL_MODELVIEW);
      if (x>y) glViewport(0, 0, y, y);
      else glViewport(0, 0, x, x);
}
// callback de teclado
void keyboard(unsigned char key, int x, int y)
{
      switch (key) {
      case 27: exit(0); break;
      case 'n': normal = !normal;
      Default: break;
      }
}
```

Exemplo 3 (Normais)

```
// desenha cubo
void drawCube(void)
{
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glTranslatef(0.0f, 0.0f, -5);
      glRotatef(yRotated, 0, 1, 0);
      glRotatef(zRotated, 0, 0, 1);

      // Desenha um cubo
      glColor3f(1.0f, 1.0f, 0.0f);
      glBegin(GL_QUADS);         // Face posterior
      if (normal) glNormal3f(0.0, 0.0, 1.0);   // Normal da face
      glVertex3f(1.0f, 1.0f, 1.0f);
      glVertex3f(-1.0f, 1.0f, 1.0f);
      glVertex3f(-1.0f, -1.0f, 1.0f);
      glVertex3f(1.0f, -1.0f, 1.0f);
      glEnd();
```

Exemplo 1 (Pontos de Vista)

```
glBegin(GL_QUADS);          // Face frontal
if (normal) glNormal3f(0.0, 0.0, -1.0);  // Normal da face
glVertex3f(1.0f, 1.0f, -1.0f);
glVertex3f(1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glEnd();
glBegin(GL_QUADS);          // Face lateral esquerda
if (normal) glNormal3f(-1.0, 0.0, 0.0);  // Normal da face
glVertex3f(-1.0f, 1.0f, 1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f, 1.0f);
glEnd();
glBegin(GL_QUADS);          // Face lateral direita
if (normal) glNormal3f(1.0, 0.0, 0.0);   // Normal da face
glVertex3f(1.0f, 1.0f, 1.0f);
glVertex3f(1.0f, -1.0f, 1.0f);
glVertex3f(1.0f, -1.0f, -1.0f);
glVertex3f(1.0f, 1.0f, -1.0f);
glEnd();
```

Exemplo 3 (Normais)

```
        glBegin(GL_QUADS);          // Face superior
        if (normal) glNormal3f(0.0, 1.0, 0.0);   // Normal da face
        glVertex3f(-1.0f, 1.0f, -1.0f);
        glVertex3f(-1.0f, 1.0f, 1.0f);
        glVertex3f(1.0f, 1.0f, 1.0f);
        glVertex3f(1.0f, 1.0f, -1.0f);
        glEnd();
        glBegin(GL_QUADS);          // Face inferior
        if (normal) glNormal3f(0.0, -1.0, 0.0);  // Normal da face
        glVertex3f(-1.0f, -1.0f, -1.0f);
        glVertex3f(1.0f, -1.0f, -1.0f);
        glVertex3f(1.0f, -1.0f, 1.0f);
        glVertex3f(-1.0f, -1.0f, 1.0f);
        glEnd();

        glutSwapBuffers();
}
```

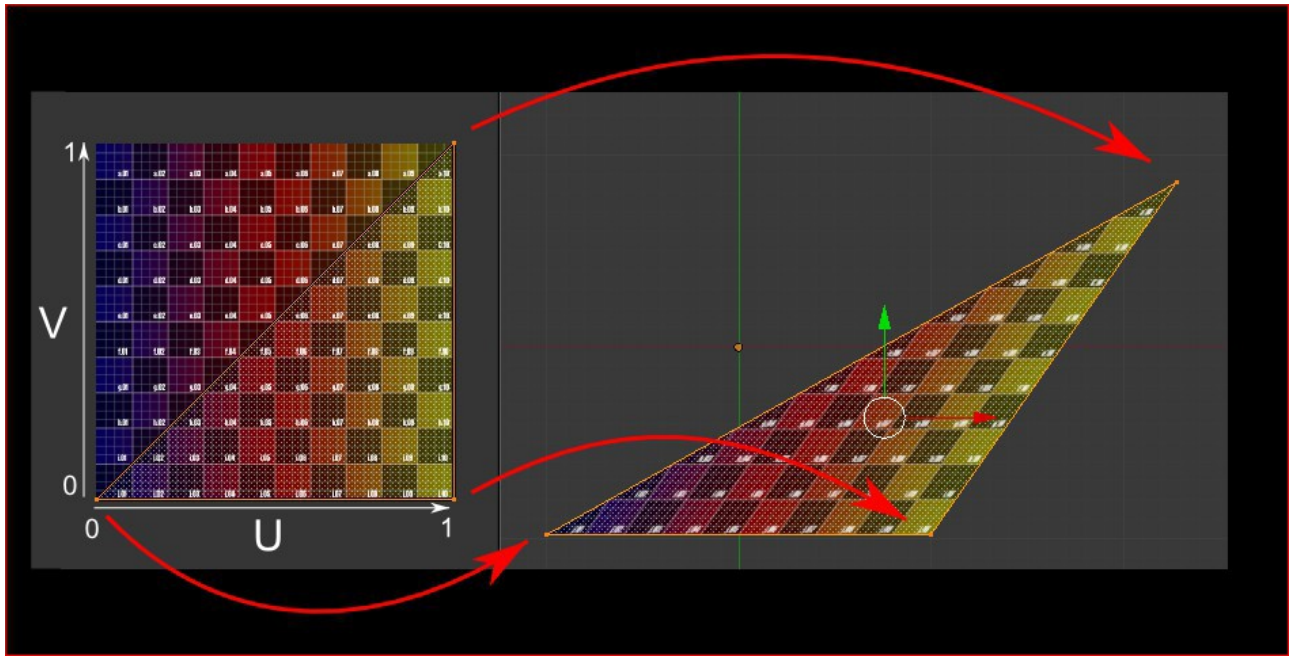
Exemplo 3 (Normais)

```
// callback idle
void idleFunc(void)
{
      yRotated += 0.01;
      zRotated += 0.01;
      drawCube();
}
int main(int argc, char** argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
      glutInitWindowSize(240, 240);
      glutInitWindowPosition(100, 100);
      glutCreateWindow("Cubo iluminado");
      initLighting();
      glutDisplayFunc(drawCube);
      glutReshapeFunc(resizeWindow);
      glutKeyboardFunc(keyboard);
      glutIdleFunc(idleFunc);
      glutMainLoop();
      return 0;
}
```

Textura

Iluminação - Normais

- Textura pode ser lida de um ou mais arquivos de imagem e mapeada para os vértices da primitiva OpenGL a ser desenhada.



Exemplo 4
Textura

Exemplo 4 (Textura)

```
#include <stdlib.h>
#include <GL/glut.h>
#include "RgbImage.h"

// variaveis globais
GLfloat xRotated, yRotated, zRotated; //angulos de rotacao
GLuint  texture[1];                   // id da textura
char* filename = "./textura.bmp"; //arquivo com a textura

// callback para redimensionar janela glut
void resizeWindow(int x, int y)
{
      if (y == 0 || x == 0) return;
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glFrustum(-1.5, 1.5, -1.5, 1.5, 3.0, 10.0);
      glMatrixMode(GL_MODELVIEW);
      if (x>y) glViewport(0, 0, y, y);
      else glViewport(0, 0, x, x);
}
```

Exemplo 4 (Textura)

```
// desenha cubo
void drawScene(void)
{
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glBindTexture(GL_TEXTURE_2D, texture[0]);

      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glTranslatef(0.0, 0.0, -5);
      glRotatef(yRotated, 0, 1, 0);
      glRotatef(zRotated, 0, 0, 1);

      glBegin(GL_QUADS);          // Face posterior
      glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
      glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
      glTexCoord2f(0.0f, 1.0f); glVertex3f(1.0f, 1.0f, -1.0f);
      glTexCoord2f(0.0f, 0.0f); glVertex3f(1.0f, -1.0f, -1.0f);
      glEnd();
```

Exemplo 4 (Textura)

```
glBegin(GL_QUADS);          // Face frontal
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
glEnd();
glBegin(GL_QUADS);          // Face lateral esquerda
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
glEnd();
glBegin(GL_QUADS);          // Face lateral direita
glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(1.0f, -1.0f, 1.0f);
glEnd();
```

Exemplo 4 (Textura)

```
        glBegin(GL_QUADS);              // Face superior
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, 1.0f, 1.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f, -1.0f);
        glEnd();
        glBegin(GL_QUADS);              // Face inferior
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(1.0f, -1.0f, -1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(1.0f, -1.0f, 1.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
        glEnd();

        glutSwapBuffers();
}
```

Exemplo 4 (Textura)

```
// callback de teclado
void keyboard(unsigned char key, int x, int y)
{
      switch (key) {
      case 27:
          exit(0);
          break;
      default:
          break;
      }
}

// callback idle
void idleFunc(void)
{
      yRotated += 0.01;
      zRotated += 0.01;
      drawScene();
}
```

Exemplo 4 (Textura)

```
int main(int argc, char** argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
      glutInitWindowSize(240, 240);
      glutInitWindowPosition(100, 100);
      glutCreateWindow("Cubo com textura");

      glEnable(GL_TEXTURE_2D);
      loadTextureFromFile(filename);

      glutDisplayFunc(drawScene);
      glutReshapeFunc(resizeWindow);
      glutKeyboardFunc(keyboard);
      glutIdleFunc(idleFunc);

      glutMainLoop();
      return 0;
}
```

Exemplo 5
Textura Esfera

Exemplo 5 (Textura Esfera)

```
#include <windows.h>
#include <iostream>
#include <stdlib.h>
#include <GL/glut.h>
#include "RgbImage.h"
using namespace std;

char* filename = "./earthmap1k.bmp"; //image file with the texture
char view = 's';
GLuint _textureId; //The id of the texture
GLUquadric *quad;
float rotationAngle;

void handleKeypress(unsigned char key, int x, int y) {
       switch (key) {
       case 27: //Escape key
           exit(0); break;
       case 't': //Top view
           view = key; glutPostRedisplay();  break;
       case 's': //Side view
           view = key; glutPostRedisplay();  break;
       case 'b': //Bottom view
           view = key; glutPostRedisplay();  break;
       }
}
```

Exemplo 5 (Textura Esfera)

```
//Makes the image into a texture, and returns the id of the texture
GLuint loadTexture(char *filename) {
       GLuint textureId;

       RgbImage theTexMap(filename); //Image with texture

       glGenTextures(1, &textureId); //Make room for our texture
       glBindTexture(GL_TEXTURE_2D, textureId); //Texture to be edited
       //Map the image to the texture
       glTexImage2D(GL_TEXTURE_2D,   //Always GL_TEXTURE_2D
           0,                        //0 for now
           GL_RGB,                   //Format OpenGL uses for image
           theTexMap.GetNumCols(),   //Width
           theTexMap.GetNumRows(),   //Height
           0,                        //The border of the image
           GL_RGB,                   //GL_RGB: pixels are stored in RGB format
           GL_UNSIGNED_BYTE,         //GL_UNSIGNED_BYTE:pixels are stored as
                                     //unsigned numbers
           theTexMap.ImageData());   //The actual pixel data
       return textureId; //Returns the id of the texture
}
```

Exemplo 5 (Textura Esfera)

```
void initRendering() {
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_NORMALIZE);
        glEnable(GL_COLOR_MATERIAL);

        quad = gluNewQuadric();
        _textureId = loadTexture(filename);
}

void handleResize(int w, int h) {
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(45.0, (float)w / (float)h, 1.0, 200.0);
}
```

Exemplo 5 (Textura Esfera)

```
void drawScene() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, _textureId);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        switch (view) {
        case 't': //Top view
            gluLookAt(0.0f,0.0f,20.0f,0.0f,0.0f,0.0f,0.0f,-1.0f,0.0f);
            break;
        case 's': //Side view
            gluLookAt(20.0f,0.0f,0.0f,0.0f,0.0f,0.0f,0.0f,0.0f,1.0f);
            break;
        case 'b': //Bottom view
            gluLookAt(0.0f,0.0f,-20.0f,0.0f,0.0f,0.0f,0.0f,1.0f,0.0f);
            break;
        }
        glRotatef(rotationAngle, 0.0f, 0.0f, 1.0f);
        gluQuadricTexture(quad, 1);
        gluSphere(quad, 4, 40, 40);
        glutSwapBuffers();
}
```

Exemplo 5 (Textura Esfera)

```
void update(int value)
{
        rotationAngle += 1.0f;
        if (rotationAngle>360.f)
            rotationAngle -= 360;
        glutPostRedisplay();
        glutTimerFunc(25, update, 0);
}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(800, 800);

        glutCreateWindow("Textured Earth");
        initRendering();
        glutTimerFunc(25, update, 0);
        glutDisplayFunc(drawScene);
        glutKeyboardFunc(handleKeypress);
        glutReshapeFunc(handleResize);

        glutMainLoop();
        return 0;
}
```

Fim