Computação Gráfica (IME 04-10842)
2022.2

Modelagem e Visualização no OpenGL

Gilson. A. O. P. Costa (IME/UERJ)

gilson.costa@ime.uerj.br

Conteúdo

- Transformações no OpenGL
- Manipulação de Matrizes
- Modelagem e Visualização
- Projeção
- Primitivas 3D
- Exemplos

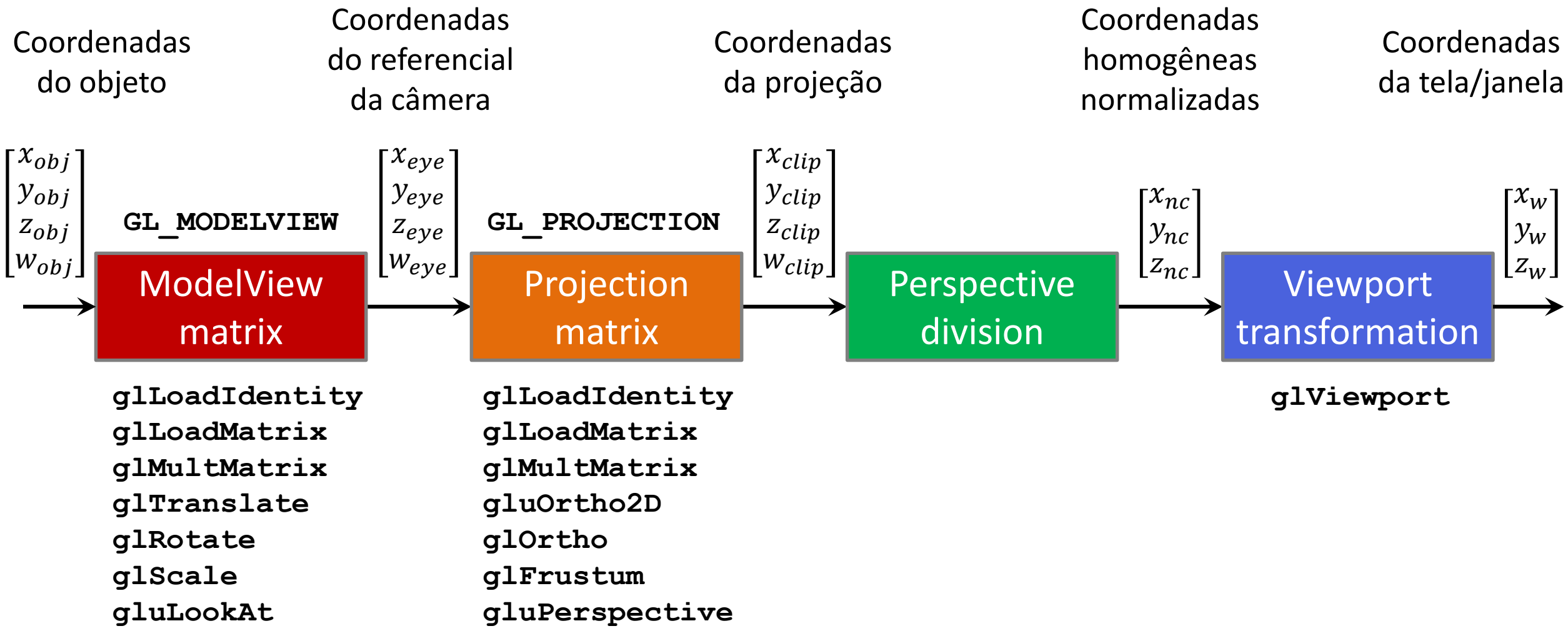
Transformações no OpenGL

- No OpenGL as transformações são combinadas em matrizes específicas:

  `GL_MODELVIEW`, **`GL_PROJECTION`**, **`GL_TEXTURE`**, **`GL_COLOR`**.

- As matrizes **`GL_MODELVIEW`** e **`GL_PROJECTION`** definem transformações que são aplicadas às coordenadas de todos os pontos/objetos modelados.
- Cada uma das matrizes está associada a uma pilha, de forma a que uma matriz pode ser armazenada e recuperada posteriormente.
- As matrizes na pilha representam estados de uma aplicação.

Sequência de Transformações



Manipulação de Matrizes

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

- Especifica a *matriz corrente*: a matriz que será utilizada nas próximas operações.
- O parâmetro **`mode`** pode tomar os seguintes valores: **`GL_MODELVIEW`**, **`GL_PROJECTION`**, **`GL_TEXTURE`**, **`GL_COLOR`**.
- Especifica também a pilha de matrizes correspondente à matriz corrente.

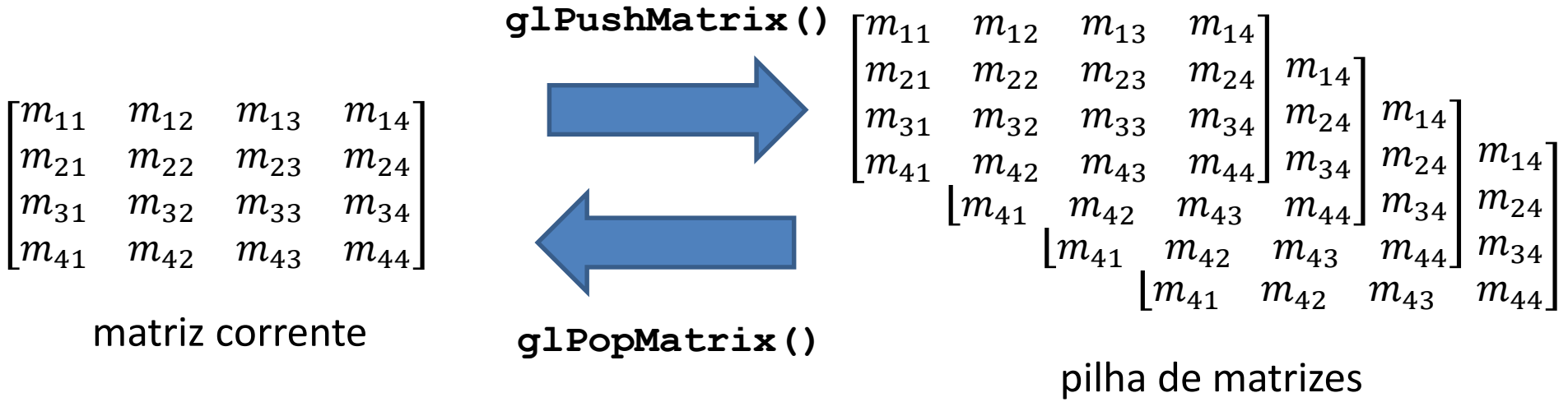
Manipulação de Matrizes

`void glPushMatrix(void);`

- Armazena a matriz corrente no topo da pilha de matrizes correspondente.

`void glPopMatrix(void);`

- Retira a matriz do topo da pilha, substituindo a matriz corrente.



Manipulação de Matrizes

- Certas funções produzem matrizes que são combinadas com a matriz corrente, e.g., **`glTranslate`**, **`glRotate`**, **`glScale`**, **`glMultMatrix`**.
- As matrizes produzidas por tais funções são multiplicadas pela matriz corrente, e o resultado se torna a própria matriz corrente.
- Certas funções simplesmente substituem a matriz corrente por outra: **`glLoadIdentity`** e **`glLoadMatrix`**.

Manipulação de Matrizes

`void glMultMatrixf(const GLfloat *m)`

- Multiplica a matriz corrente pela matriz especificada em **`*m`**.
- O parâmetro **`*m`** é um vetor de 16 posições.
- A matriz correspondente é formada coluna a coluna:

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

matriz corrente ← matriz corrente × $\mathbf{M}$

Manipulação de Matrizes

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

- Multiplica a matriz corrente pela matriz de translação especificada por **`x`**, **`y`** e **`z`**:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

matriz corrente ← matriz corrente × $\mathbf{T}$

Manipulação de Matrizes

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

- Multiplica a matriz corrente pela matriz de rotação especificada por **`angle`**, **`x`**, **`y`** e **`z`**.
- Os parâmetros **`x`**, **`y`** e **`z`** especificam um vetor, sobre o qual a rotação será feita.
- O parâmetro **`angle`** especifica o ângulo de rotação.

$$c = \cos(angle) \qquad s = \sin(angle)$$

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

matriz corrente ← matriz corrente × $\mathbf{R}$

Manipulação de Matrizes

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

- Multiplica a matriz corrente pela matriz de escala especificada por **`x`**, **`y`** e **`z`**:

$$\mathbf{S} = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

matriz corrente ← matriz corrente × $\mathbf{S}$

Manipulação de Matrizes

`void glLoadIdentity(void)`

- Substitui a matriz corrente pela matriz identidade:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

matriz corrente $\leftarrow \mathbf{I}$

Manipulação de Matrizes

`void glLoadMatrixf(const GLfloat *m)`

- Substitui a matriz corrente pela matriz especificada em **`*m`**.
- O parâmetro **`*m`** é um vetor de 16 posições.
- A matriz correspondente é formada coluna a coluna:

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

matriz corrente $\leftarrow \mathbf{M}$

Modelagem e Visualização (Model × View)

- No OpenGL *modelagem* está relacionada ao conjunto de transformações que mapeiam o referencial local (espaço do objeto) para o referencial global (espaço do mundo).
- Estas transformações devem ser aplicadas à matriz **`GL_MODELVIEW`**.
- As transformações/operações geralmente usadas estão associadas às funções: **`glTranslate`**, **`glRotate`**, **`glScale`**, **`glMultMatrix`**, **`glLoadIdentity`** e **`glLoadMatrix`**.

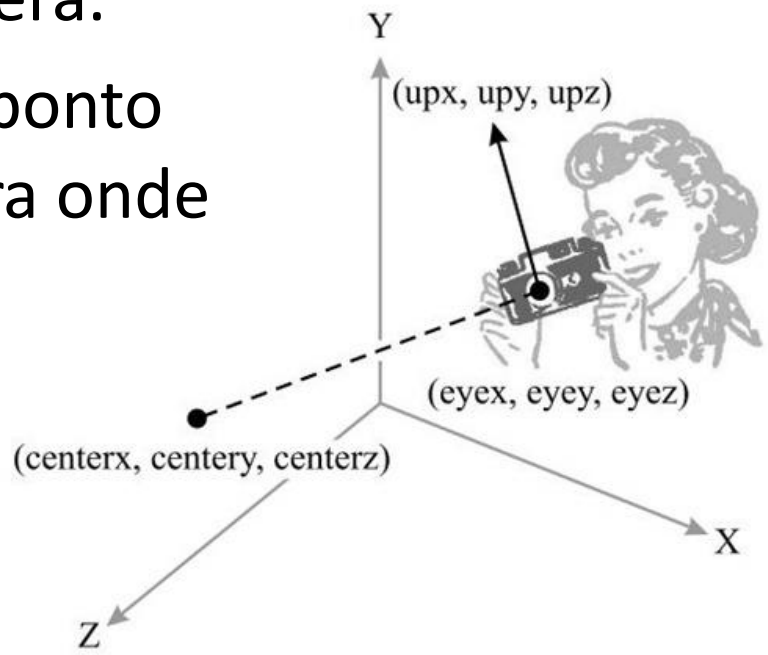
Modelagem e Visualização (Model × View)

- No OpenGL *visualização* está relacionada a transformações que mapeiam o referencial global (espaço do mundo) para o referencial da câmera (espaço da câmera, ou do olho).
- Estas transformações devem ser aplicadas à matriz **`GL_MODELVIEW`**.
- As transformações/operações geralmente usadas estão associadas às funções: **`glLoadIdentity`** e **`glLookAt`**.
- Num programa, as funções relacionadas à *visualização* devem ser usadas **antes** das funções relacionadas à *modelagem*!

Modelagem e Visualização (Model × View)

`void glLookAt(eyeXYZ, centerXYZ, upXYZ)`

- O parâmetro **`eye`** indica as coordenadas da câmera.
- O parâmetro **`center`** indica as coordenadas do ponto de referência para a direção de visualização (para onde a câmera está olhando).
- O parâmetro **`up`** indica a direção/vetor correspondente ao eixo vertical da câmera.



Modelagem e Visualização (Model × View)

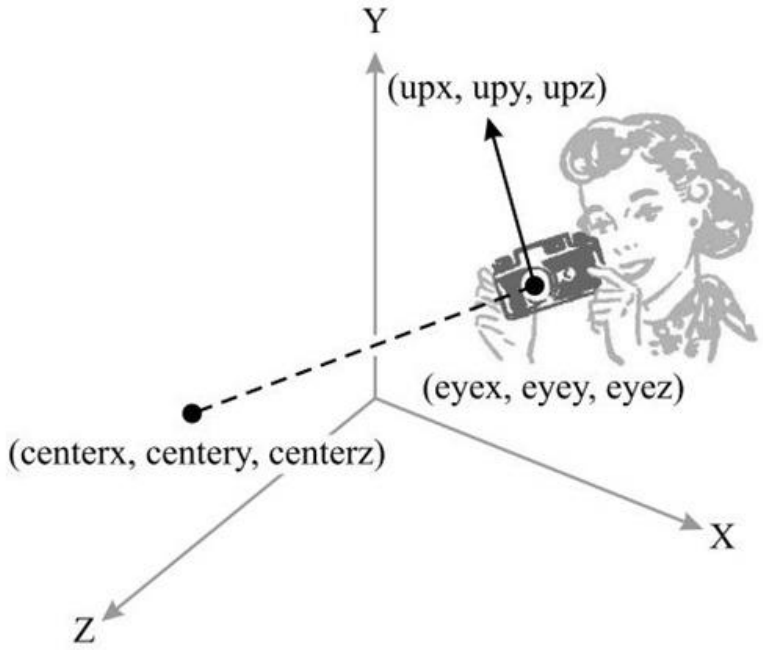
`void glLookAt(eyeXYZ, centerXYZ, upXYZ)`

$$f = c - e \qquad f' = \frac{f}{|f|}$$

$$u' = \frac{u}{|u|} \qquad s = f \times u' \qquad u'' = s \times f$$

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

matriz corrente ← matriz corrente × $\mathbf{L}$

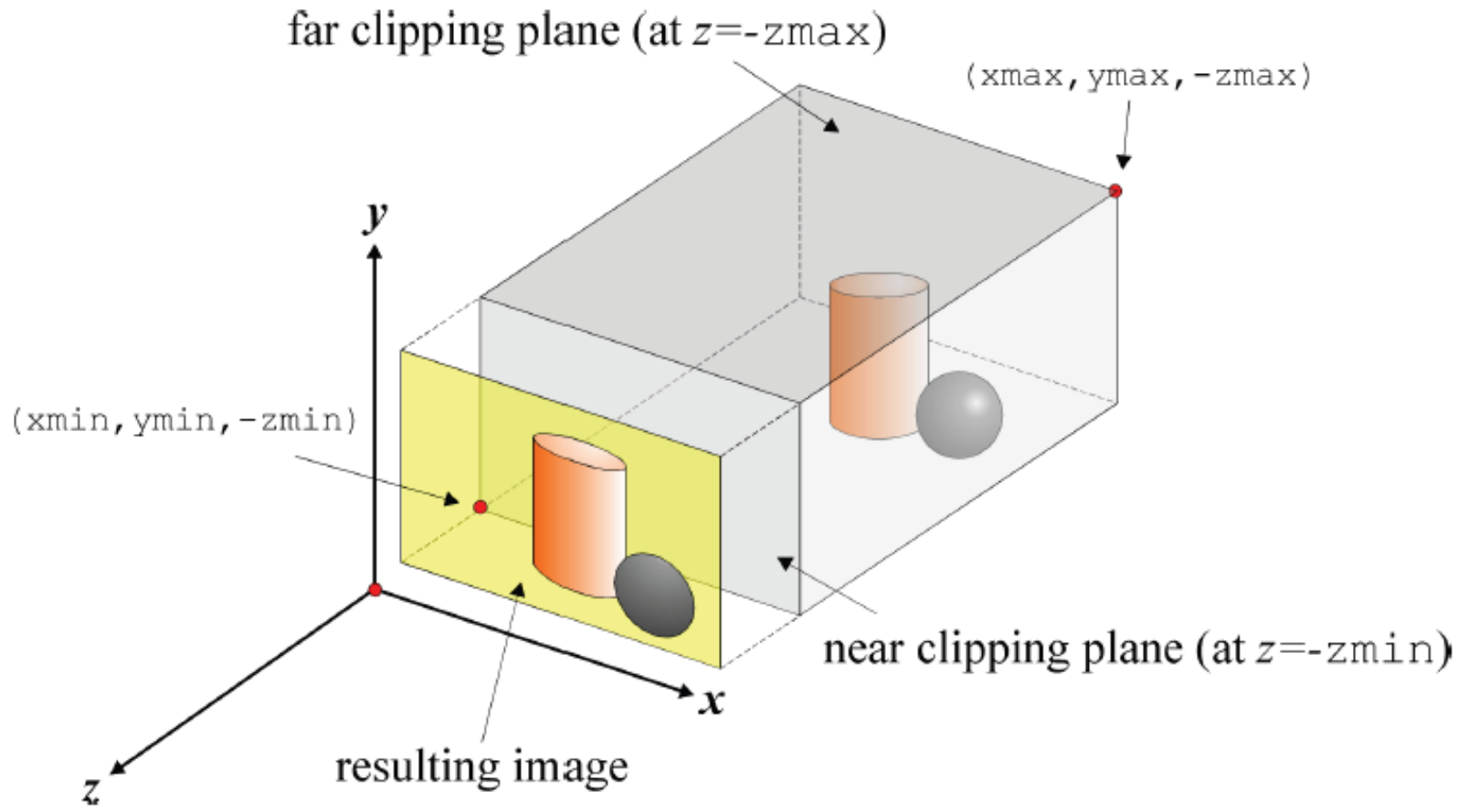


Projeção

- No OpenGL *projeção* está relacionada transformações que fazem a projeção dos pontos/objetos no espaço da câmera sobre um plano de projeção.
- Estas transformações devem ser aplicadas à matriz **`GL_PROJECTION`**.
- As transformações/operações geralmente usadas estão associadas às funções: **`glOrtho`**, **`glFrustum`**, **`gluPerspective`** e **`glLoadIdentity`**.
- Num programa, as funções relacionadas à *projeção* devem ser usadas **antes** das funções relacionadas à *modelagem* e *visualização*!

Projeção

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



Projeção

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

$$\mathbf{O} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & t_x \\ 0 & \frac{2}{top-bottom} & 0 & t_y \\ 0 & 0 & \frac{-2}{far-near} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$t_x = -\frac{right+left}{right-left}$$

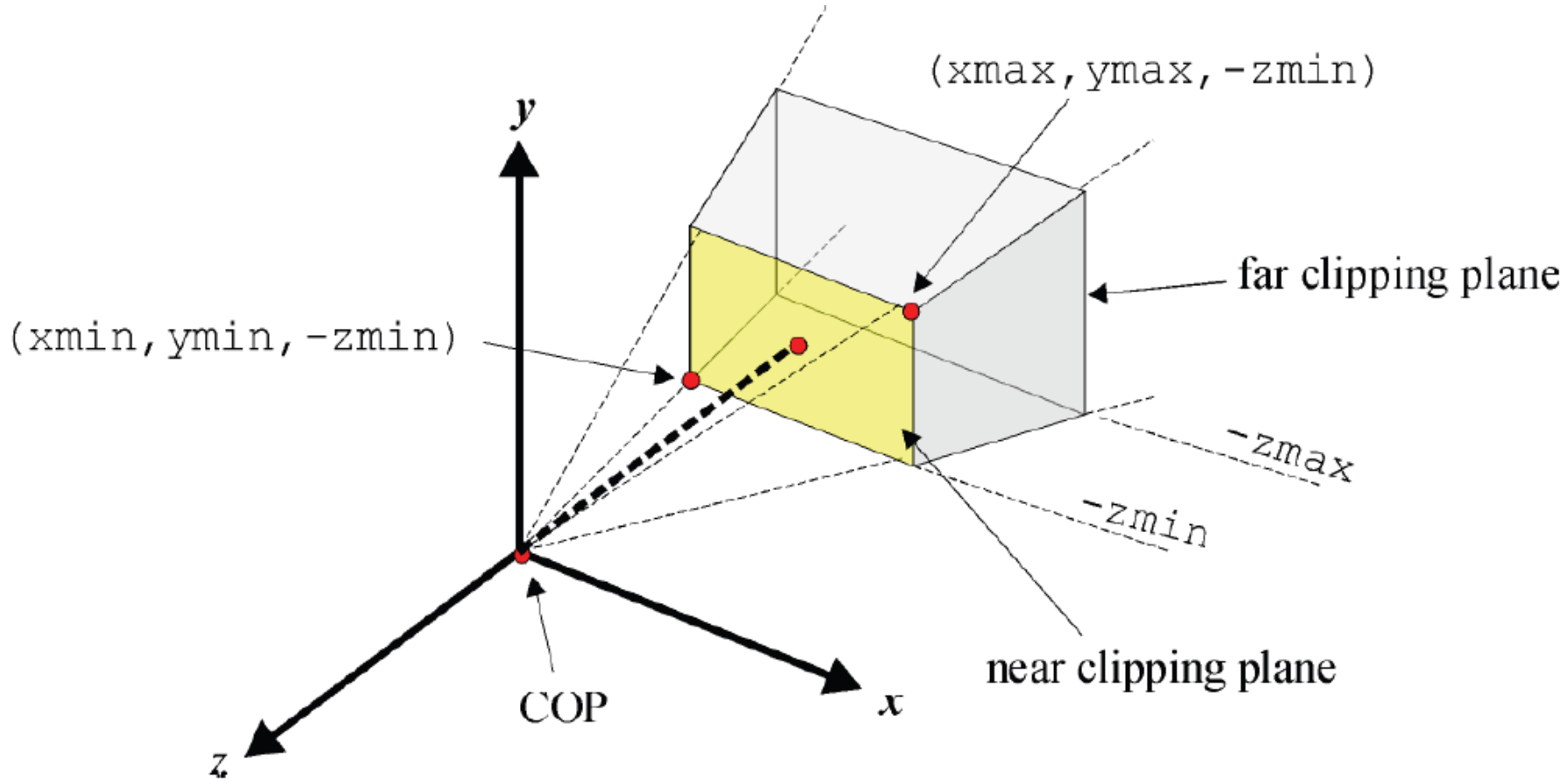
$$t_y = -\frac{top+bottom}{top-bottom}$$

$$t_z = -\frac{far+near}{far-near}$$

matriz corrente ← matriz corrente × $\mathbf{O}$

Projeção

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



Projeção

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

$$\mathbf{F} = \begin{bmatrix} \frac{2 \times near}{right - left} & 0 & a & 0 \\ 0 & \frac{2 \times near}{top - bottom} & b & 0 \\ 0 & 0 & \frac{far + near}{far - near} & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$a = \frac{right + left}{right - left}$$

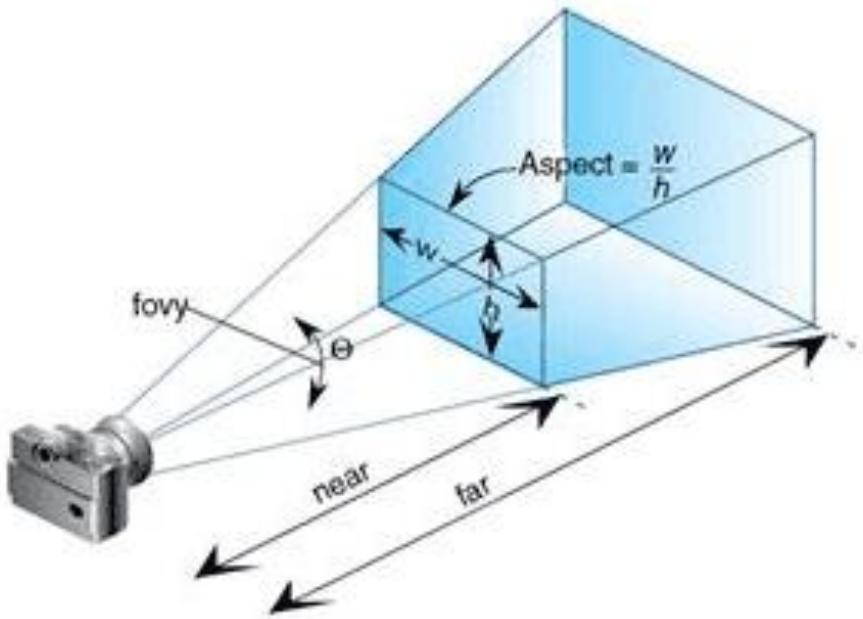
$$b = \frac{top + bottom}{top - bottom}$$

$$c = \frac{2 \times far \times near}{far - near}$$

matriz corrente ← matriz corrente × $\mathbf{F}$

Projeção

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



Projeção

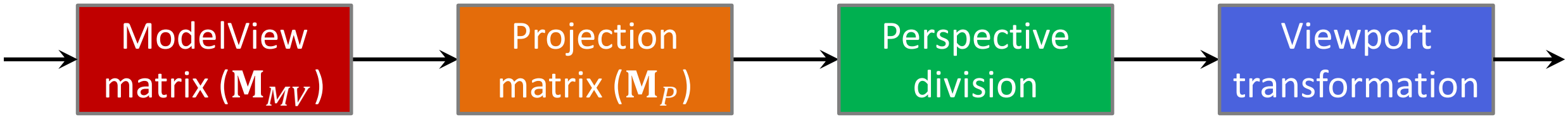
```
void gluPerspective(fovy,aspect,near,far)
```

$$\mathbf{P} = \begin{bmatrix} \frac{f}{aspect} & 0 & a & 0 \\ 0 & f & b & 0 \\ 0 & 0 & \frac{far + near}{near - far} & \frac{2 \times far \times near}{near - far} \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad f = cotangent\left(\frac{fovy}{2}\right)$$

matriz corrente ← matriz corrente × $\mathbf{P}$

Sequência de Transformações

$$\begin{bmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{bmatrix} \times \mathbf{M}_{MV} = \begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix} \qquad \begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix} \times \mathbf{M}_{P} = \begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} \qquad \begin{bmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{bmatrix} = \begin{bmatrix} x_{nc} \\ y_{nc} \\ z_{nc} \end{bmatrix} \qquad \begin{bmatrix} \frac{w}{2} x_{nc} + x + \frac{w}{2} \\ \frac{h}{2} y_{nc} + y + \frac{h}{2} \\ \frac{f-n}{2} z_{nc} + \frac{f+n}{2} \end{bmatrix} = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$




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

Exemplo #6: 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
void display(void); // callback de desenho
void keyboard(unsigned char key, int x, int y); // callback de teclado
```

Exemplo #6: 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 #6: 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 #6: 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 #6: 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 #6: 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 #6: 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 #6: 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);
glLoadIdentity();
gluLookAt(5.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
drawObject();
```

Exemplo #6: Pontos de Vista

```
// 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);
glLoadIdentity();
gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
drawObject();
```

Exemplo #6: 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 #6: Pontos de Vista

Experimente:

- Aperte as teclas “a” e “s” 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 do GLUT

- A função **`glutSolidTeapot(GLdouble 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 envolve totalmente o objeto.
- A biblioteca GLUT possui funções para desenhar outros objetos 3D.
- As listas a seguir não são exaustivas.

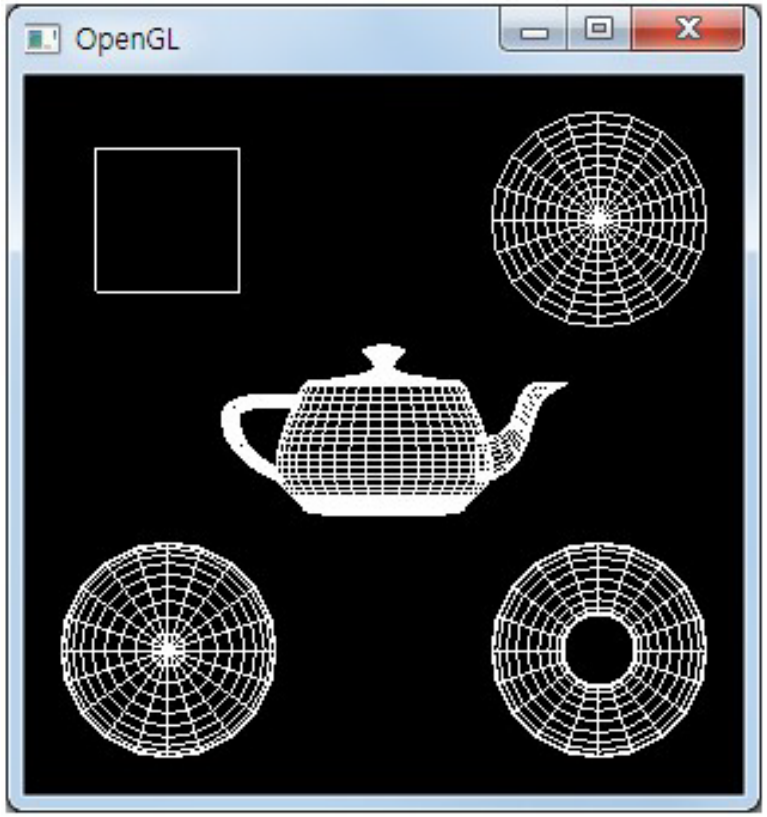
Primitivas 3D do GLUT

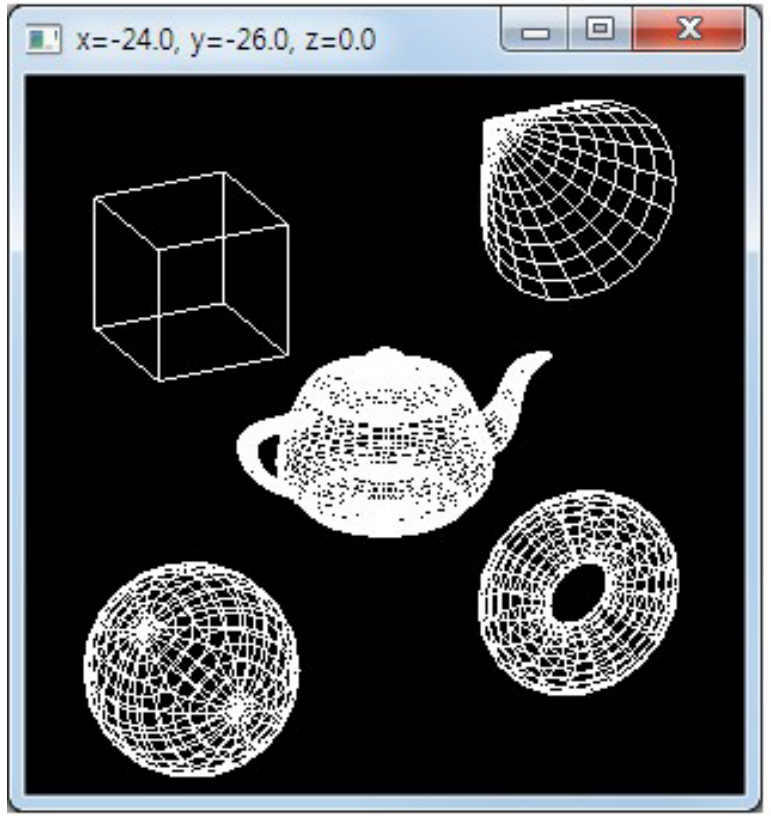
Wire-frame:

- `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(void);`

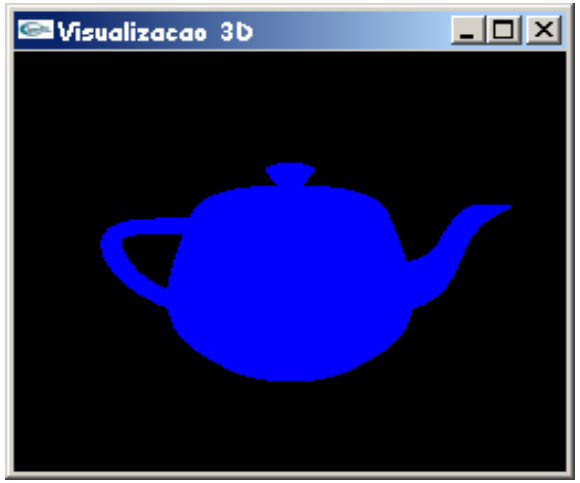
Primitivas 3D do GLUT

Wire-frame:





Primitivas 3D do 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(void);`

Computação Gráfica (IME 04-10842)
2022.2

Modelagem e Visualização no OpenGL

Gilson. A. O. P. Costa (IME/UERJ)

gilson.costa@ime.uerj.br