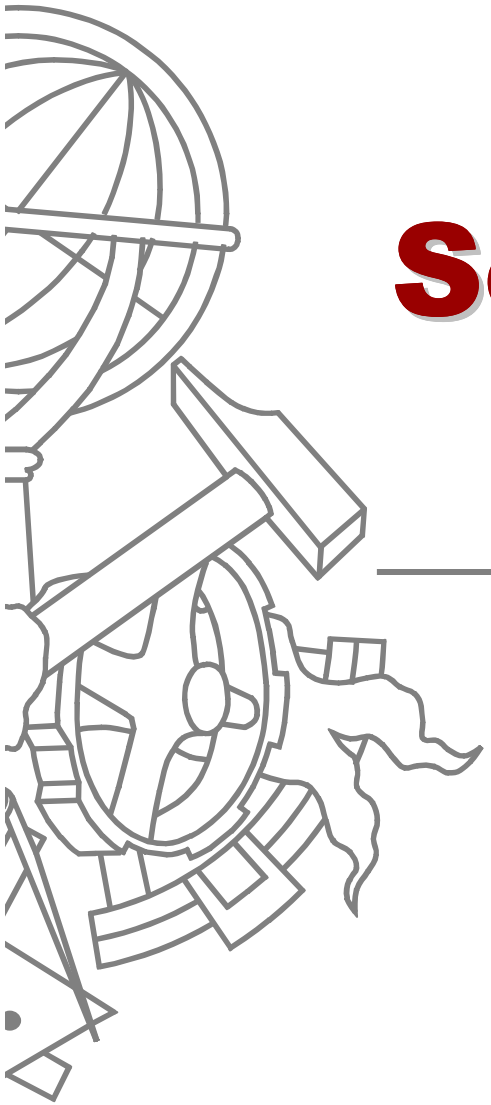


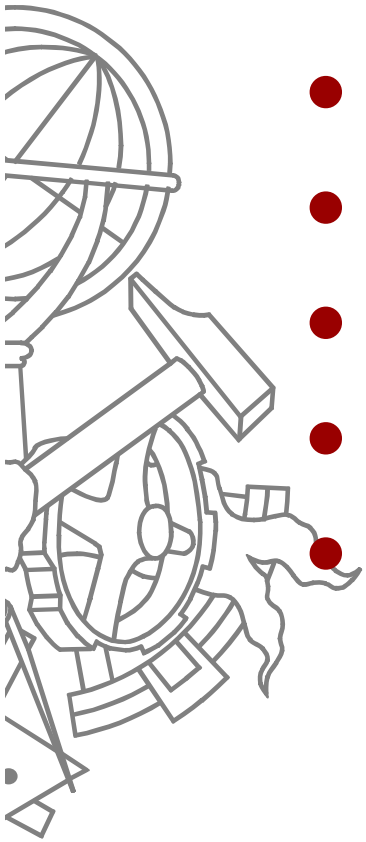
Sólidos "Primitivos" e *Display lists*

Aula 3

Sistemas Gráficos e Interactivos
Instituto Superior de Engenharia do Porto

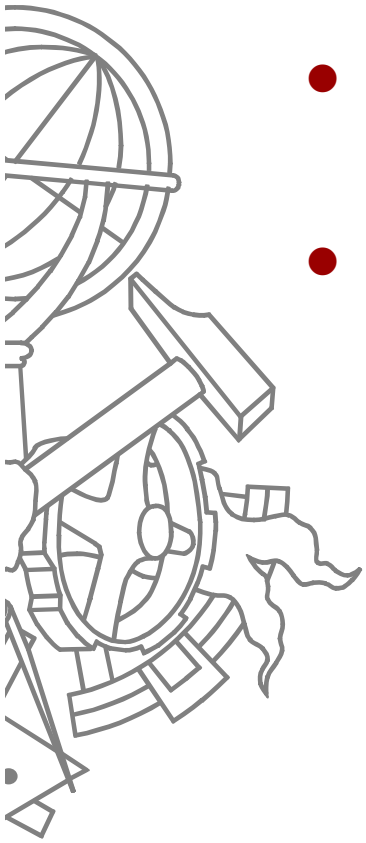
Paulo Gandra de Sousa
psousa@dei.isep.ipp.pt

Conteúdo



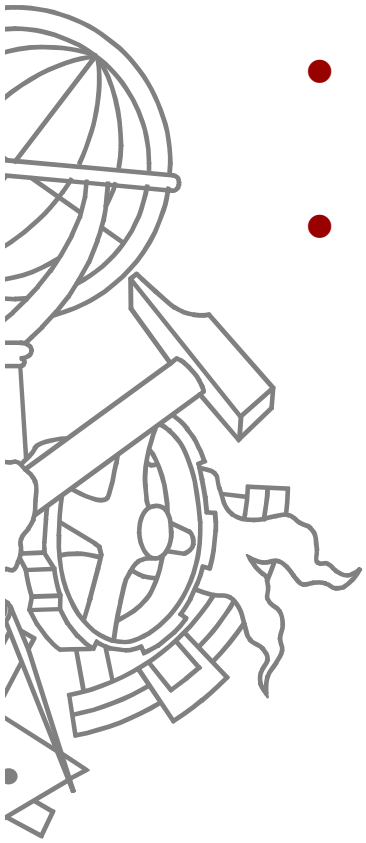
- Sólidos primitivos do GLUT
- Sólidos primitivos do GLU
- Iluminação básica
- *Display lists*
- Animações
  - Utilização de *double buffer*
  - glutIdleFunc, glutTimerFunc

Esferas

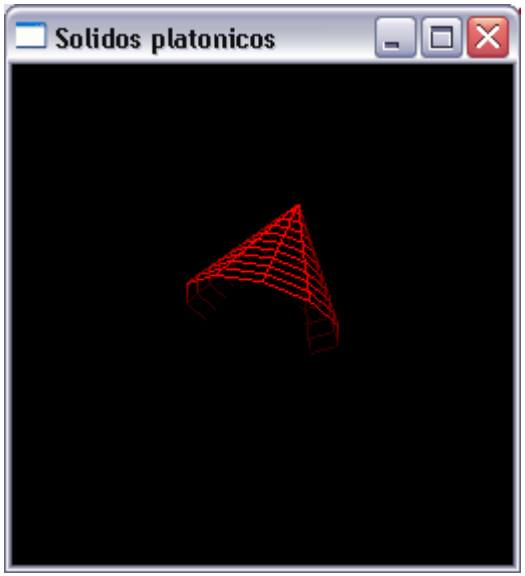


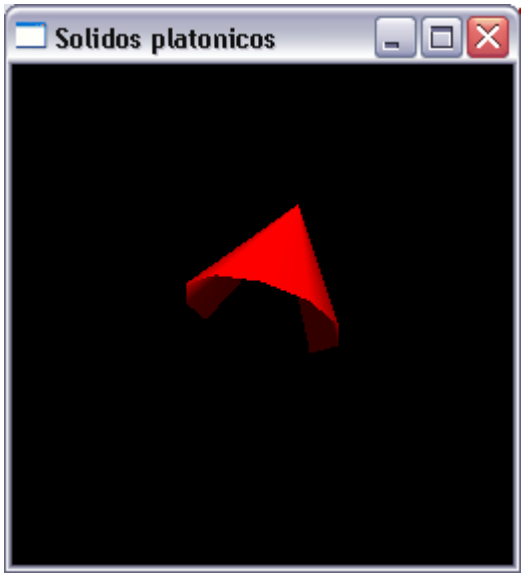
- `glutWireSphere(GLdouble radius, GLint slices, GLint stacks);`
- `glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);`

Cones

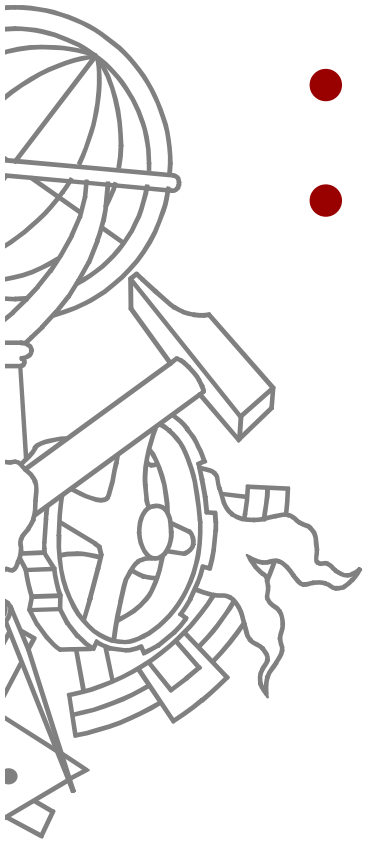


- `glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);`
- `glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);`

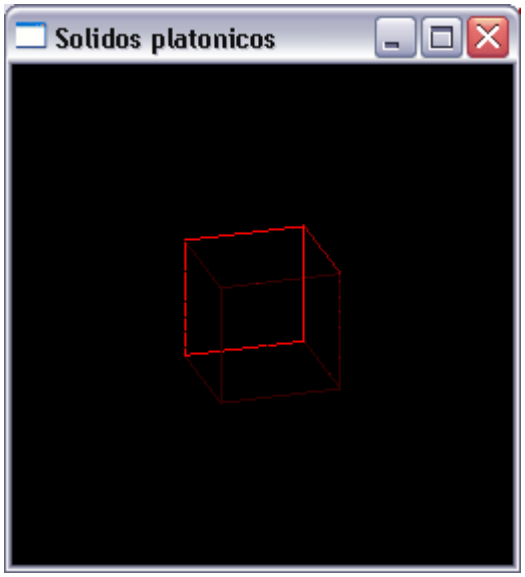




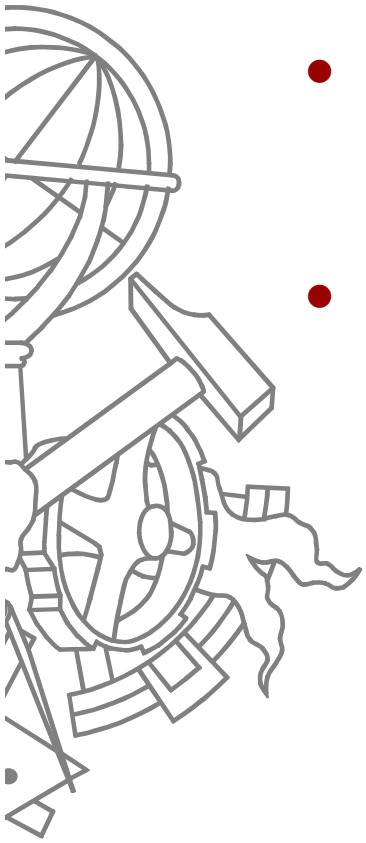
Cubos



- `glutWireCube(GLdouble size);`
- `glutSolidCube(GLdouble size);`

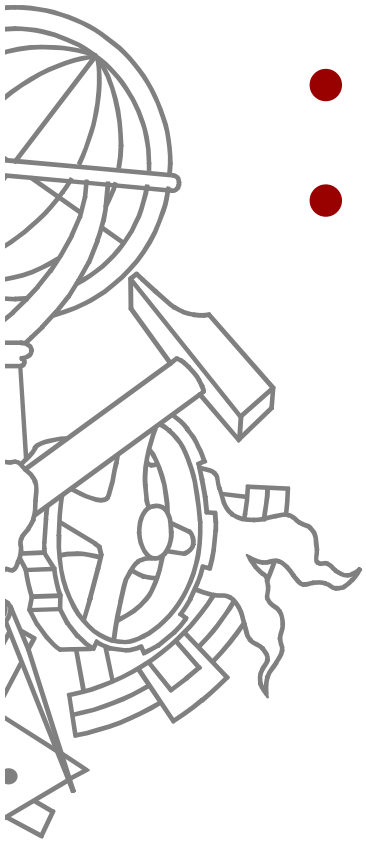


Torus



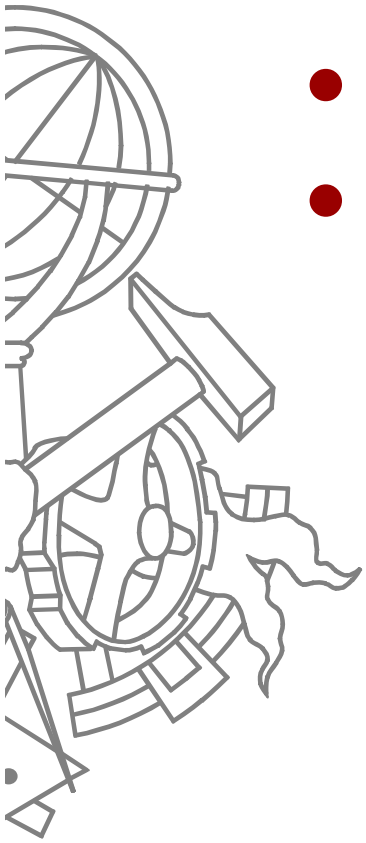
- `glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);`
- `glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);`

Dodecaedro

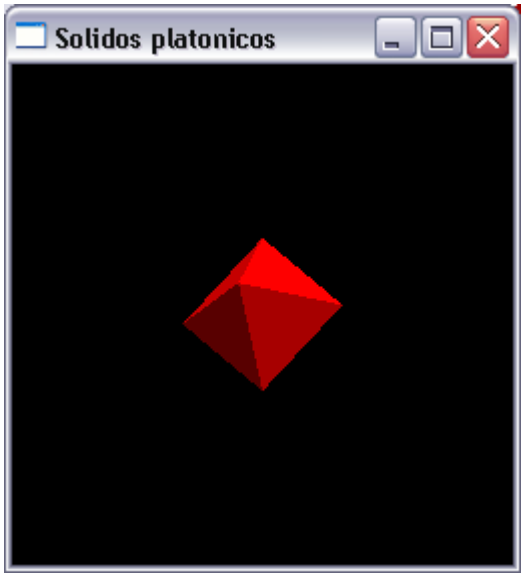


- `glutWireDodecahedron()`
- `glutSolidDodecahedron()`

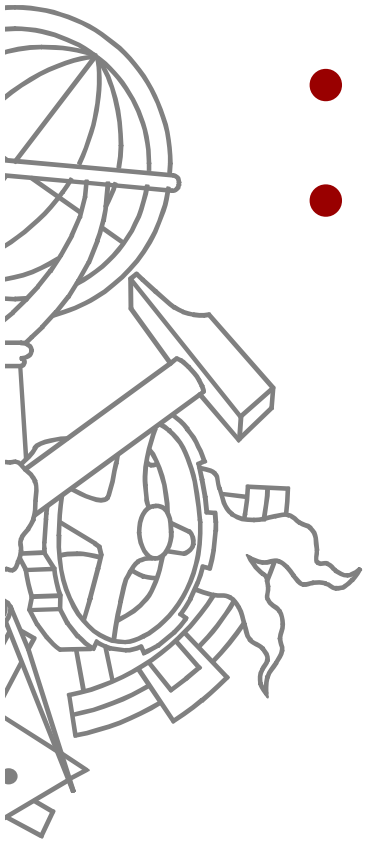
Octaedro



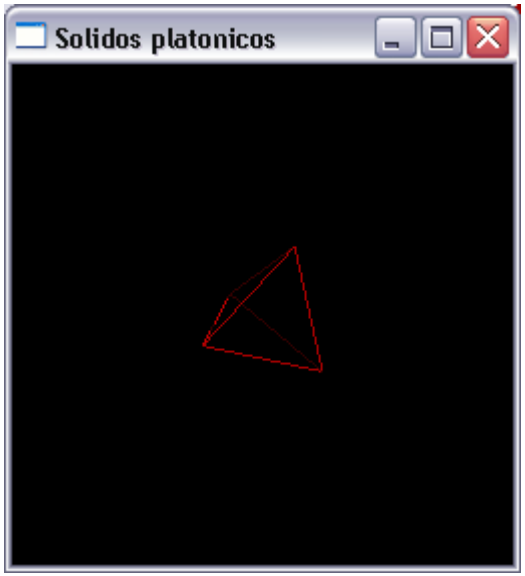
- `glutWireOctahedron()`
- `glutSolidOctahedron()`



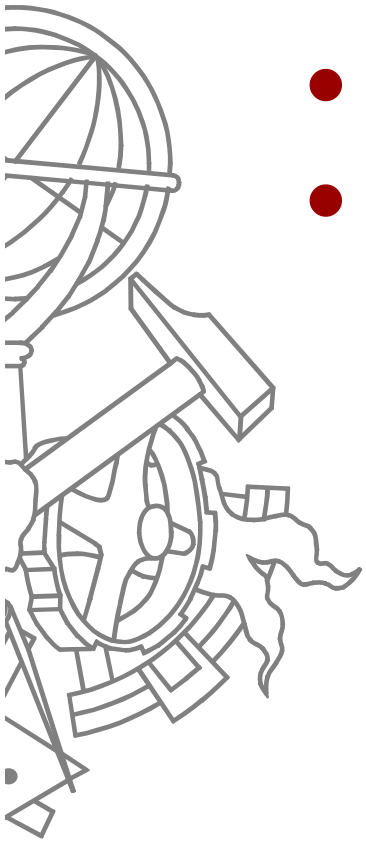
Tetraedro



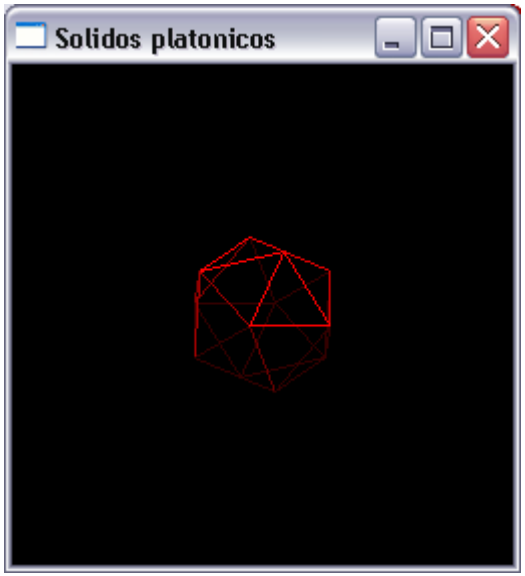
- `glutWireTetrahedron()`
- `glutSolidTetrahedron()`



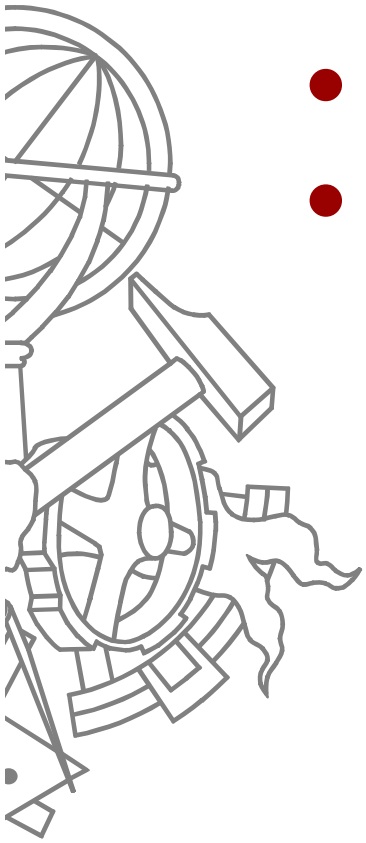
Icosaedro



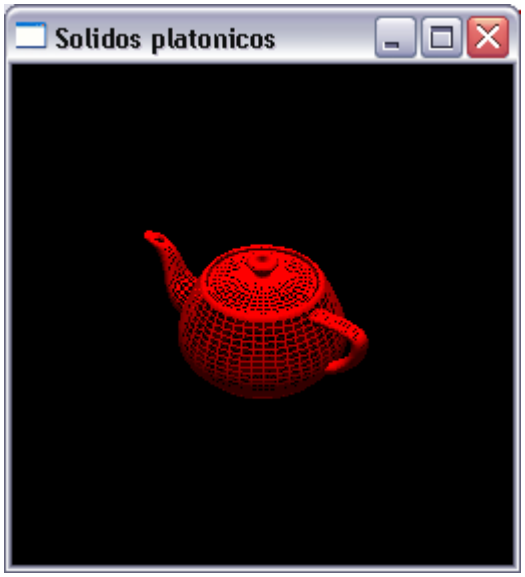
- `glutWireIcosahedron()`
- `glutSolidIcosahedron()`



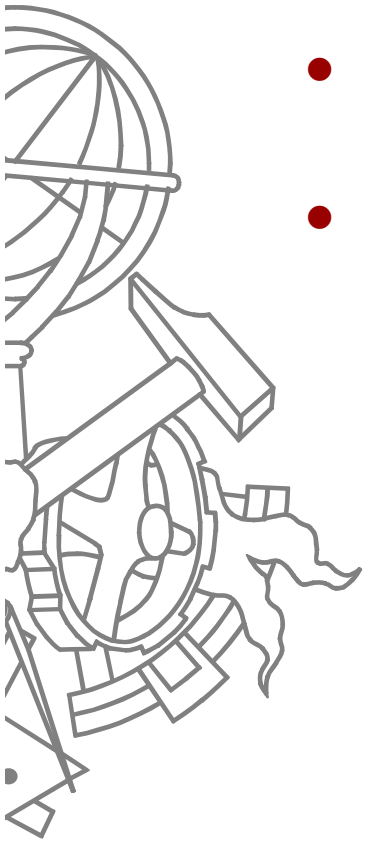
Bule de chá



- `glutWireTeapot(GLdouble size)`
- `glutSolidTeapot(GLdouble size)`



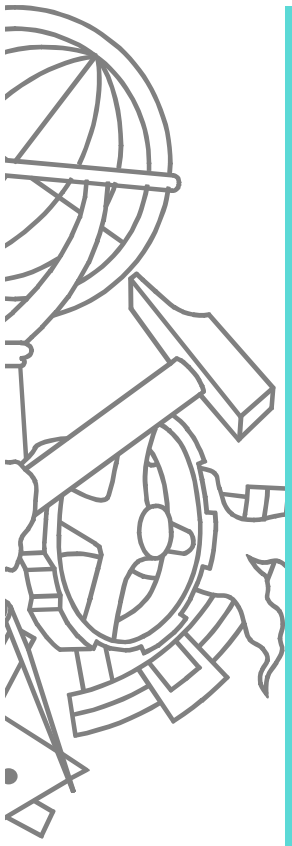
GLU quadrics



- Objectos descritos por uma equação quadrática

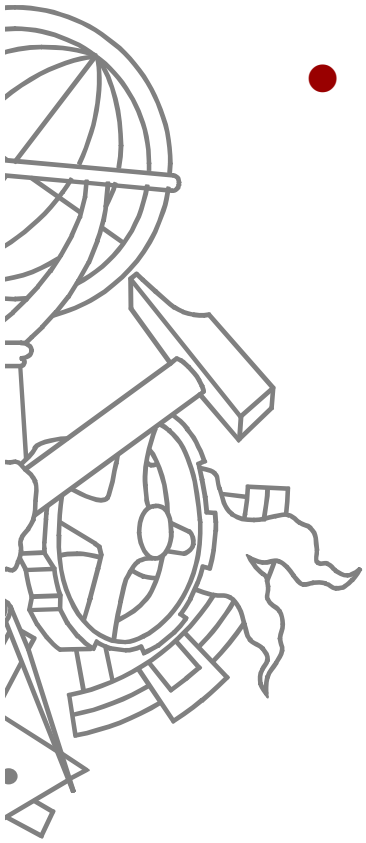
- Utilização da interface GLU quadrics:
  1. Criar um objecto **`gluNewQuadric().`**
  2. Especificar atributos de desenho:
     1. **`gluQuadricOrientation()`** to control the winding direction and differentiate the interior from the exterior.
     2. **`gluQuadricDrawStyle()`** to choose between rendering the object as points, lines, or filled polygons.
     3. **`gluQuadricNormals()`** to specify one normal per vertex or one normal per face. The default is that no normals are generated at all.
     4. **`gluQuadricTexture()`** to generate texture coordinates.
  3. Registar *callback* de erros em **`gluQuadricCallback().`**
  4. Construir o objecto **`gluSphere(), gluCylinder(), gluDisk()`**, ou **`gluPartialDisk().`**

GLU quadrics



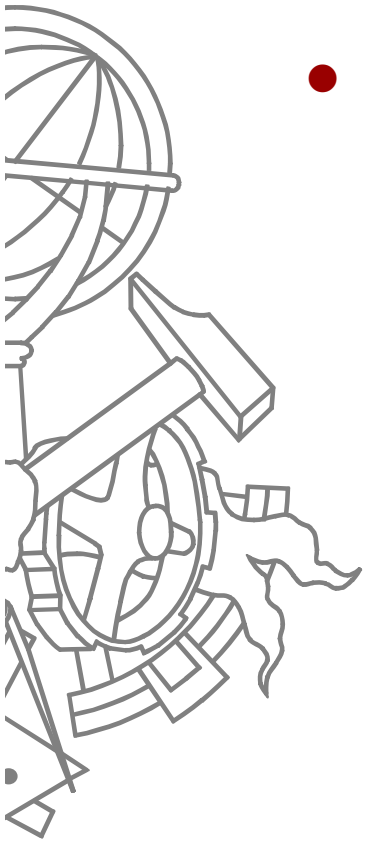
```
// criar e destruir objectos quadric não é o mais eficiente
// o objecto deveria ser colocado numa display list e
// reutilizado quando necessário

void cylinder(GLenum mode)
{
   GLUquadricObj* qobj = gluNewQuadric();
   gluQuadricDrawStyle(qobj, mode); //GLU_LINE //GLU_FILL
   gluQuadricNormals(qobj, GLU_SMOOTH);
   gluCylinder(qobj, 1.5, 1.5, 2, slices, stacks);
   gluDeleteQuadric(qobj);
}
```

GLU Sphere



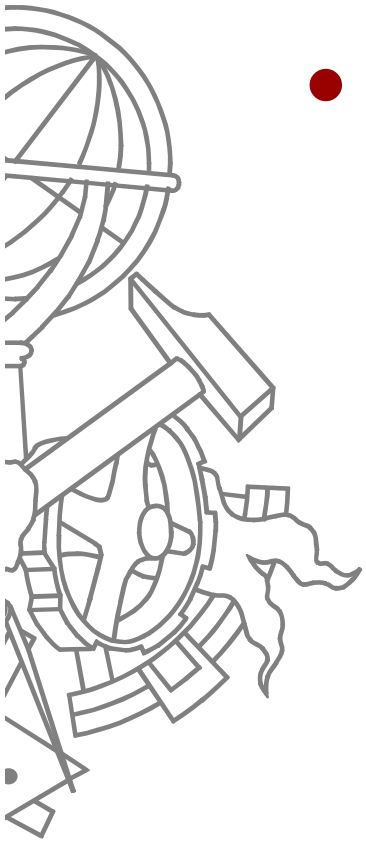
- `void gluSphere (GLUquadric *qobj, GLdouble radius, GLint slices, GLint stacks);`

GLU Cylinder



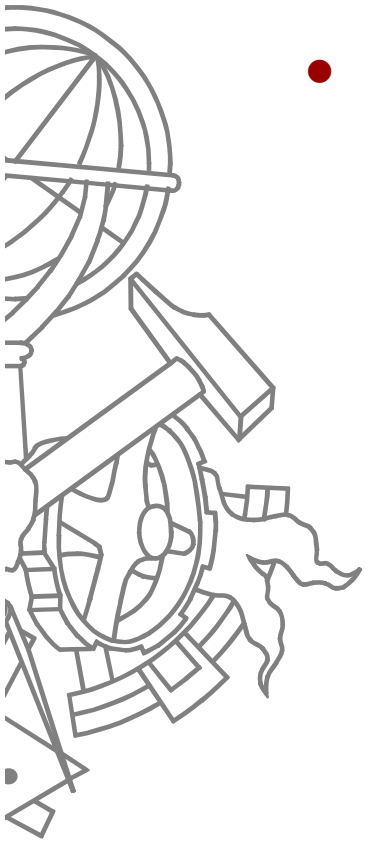
- `void gluCylinder(GLUquadric *qobj, GLdouble baseRadius, GLdouble topRadius, GLdouble height, GLint slices, GLint stacks)`

GLU Disk



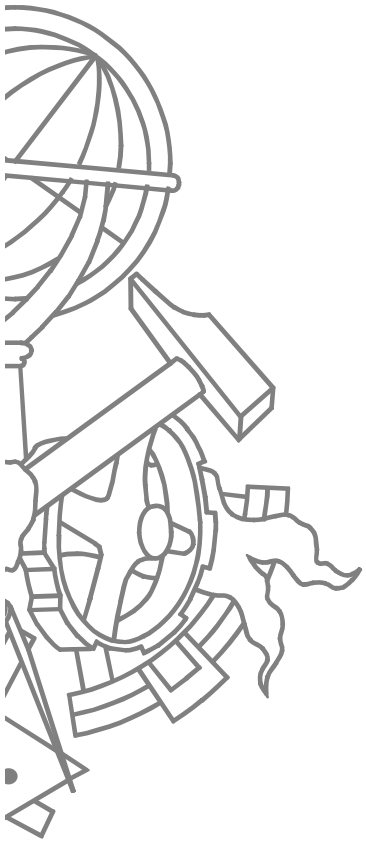
- `void gluDisk(GLUquadric *qobj, GLdouble innerRadius, GLdouble outerRadius, GLint slices, GLint loops);`

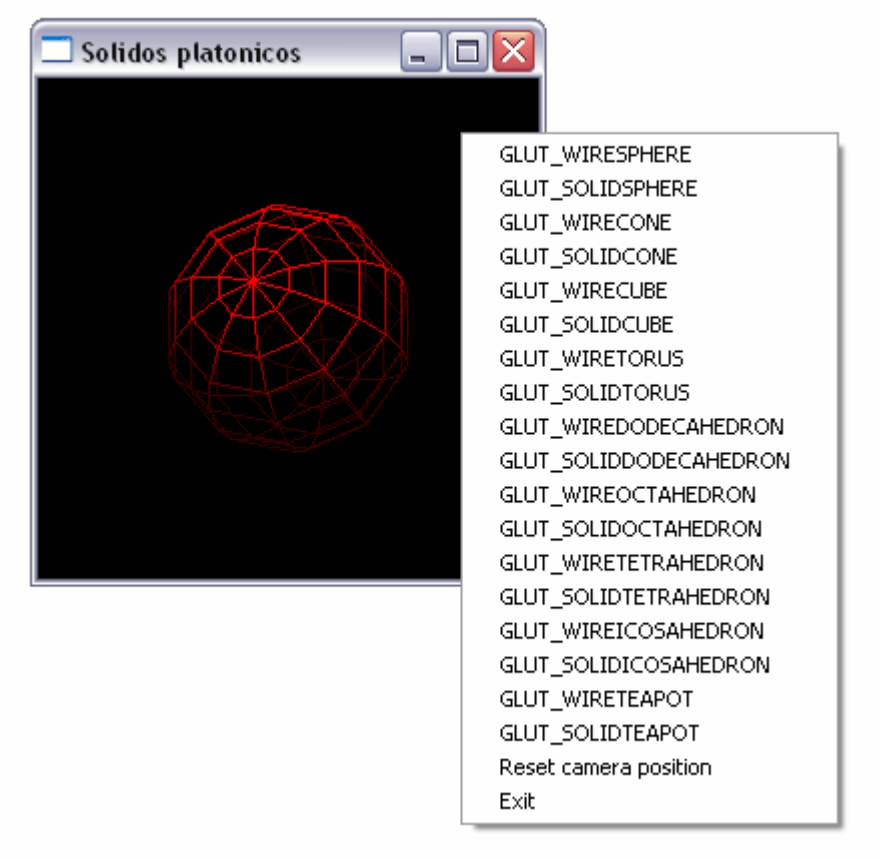
GLU Partial disk



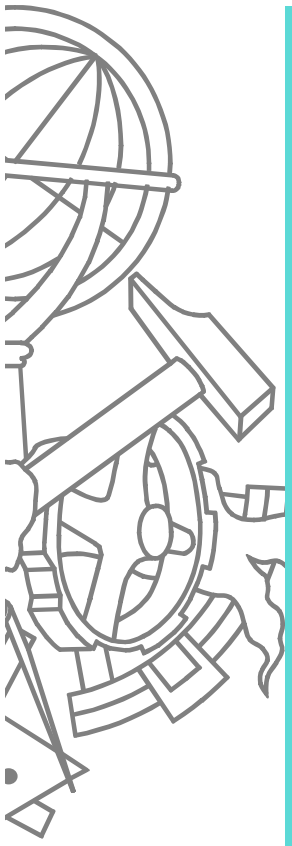
- `void gluPartialDisk(GLUquadric *qobj, GLdouble innerRadius, GLdouble outerRadius, GLint slices, GLint loops, GLdouble startAngle, GLdouble sweepAngle);`

Demo




Solidos platonicos
GLUT_WIRESPHERE
GLUT_SOLIDSPHERE
GLUT_WIRECONE
GLUT_SOLIDCONE
GLUT_WIRECUBE
GLUT_SOLIDCUBE
GLUT_WIRETORUS
GLUT_SOLIDTORUS
GLUT_WIREDODECAHEDRON
GLUT_SOLIDDODECAHEDRON
GLUT_WIREOCTAHEDRON
GLUT_SOLIDOCTAHEDRON
GLUT_WIRETETRAHEDRON
GLUT_SOLIDTETRAHEDRON
GLUT_WIREICOSAHEDRON
GLUT_SOLIDICOSAHEDRON
GLUT_WIRETEAPOT
GLUT_SOLIDTEAPOT
Reset camera position
Exit

Iluminação básica

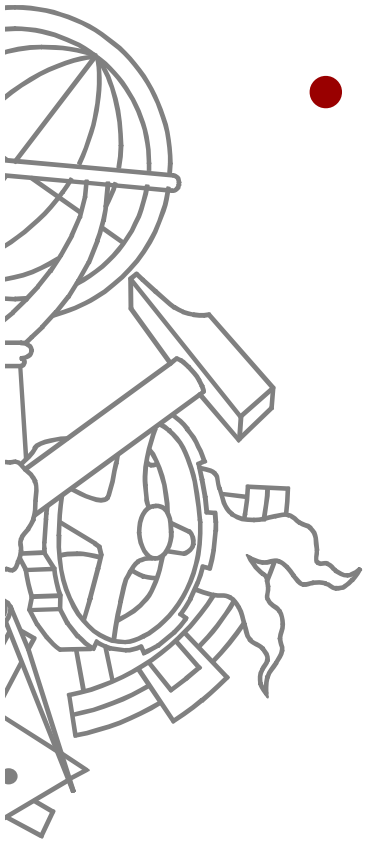


```
void init()
{
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_DEPTH_TEST);

   glFrontFace(GL_CW); // problema nas normais do teapot
   ...
}

void display()
{
   GLfloat mat[] = {0.6, 0.6, 0.6, 1.0};
   glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   ...

   glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE,
   mat);
   ...
}
```

"ligar" luz 0 (branca por omissão)
e teste de profundidade

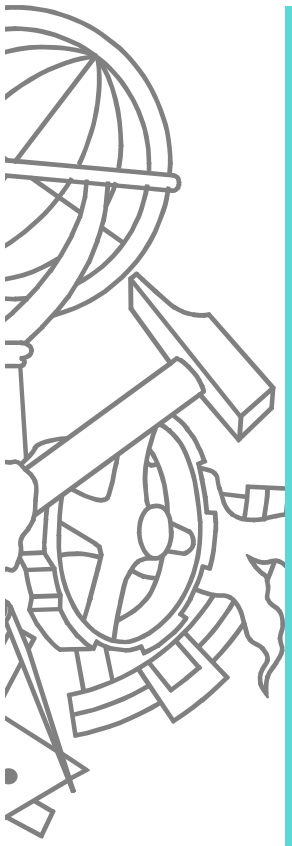
Definir material da superfície
(em vez de glColor)

Display lists



- Sequências pré-compiladas de comandos OpenGL
  - Melhoram performance ao evitar cálculos repetitivos
  - Facilitam leitura de código
  - Guardam os valores calculados dos parâmetros
  - Não podem ser alteradas após criação

Exemplo



```
void torus(int numc, int numt) {
   int i, j, k;
   double s, t, x, y, z, twopi;
   twopi = 2 * (double)M_PI;
   for (i = 0; i < numc; i++) {
      glBegin(GL_QUAD_STRIP);
      for (j = 0; j <= numt; j++)
      {
         for (k = 1; k >= 0; k--)
         {
            s = (i + k) % numc + 0.5;
            t = j % numt;
            x = (1+.1*cos(s*twopi/numc))*cos(t*twopi/numt);
            y = (1+.1*cos(s*twopi/numc))*sin(t*twopi/numt);
            z = .1 * sin(s * twopi / numc);
            glVertex3f(x, y, z);
         }
      }
      glEnd();
   }
}
```

Cálculos complexos de cada vez que desenha um *torus*

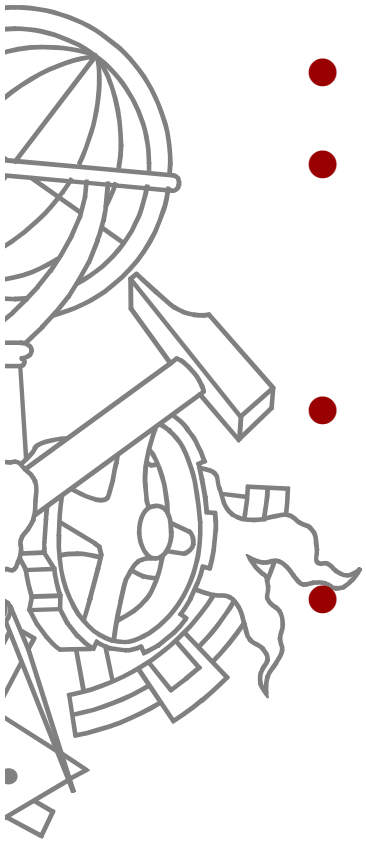
Exemplo

```
GLuint theTorus;

void init(void)
{
   theTorus = glGenLists(1);
   glNewList(theTorus, GL_COMPILE);
        torus(8, 25);
   glEndList();
   ...
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glCallList(theTorus);
   glFlush();
}
```

Comandos OpenGL são guardados na *display list*

Display list pode ser invocada quantas vezes quiser. Cálculos só são efectuados uma vez

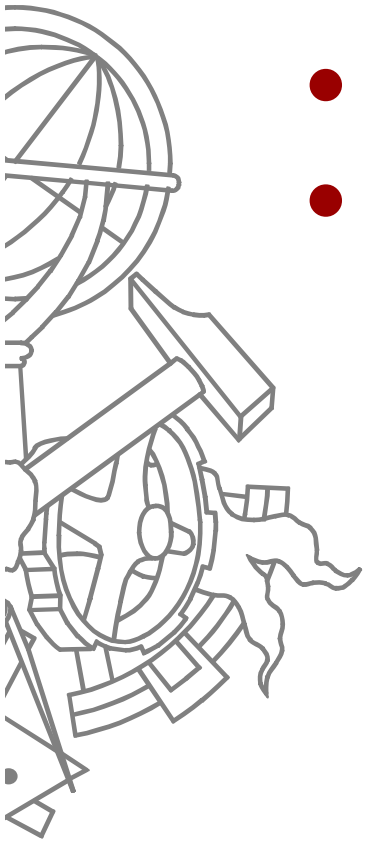
Criar lista



- **`GLuint glGenLists(`*`quantas`*`)`**
- **`void glNewList(`*`list, mode`*`)`**
  - **`GL_COMPILE`**
  - **`GL_COMPILE_AND_EXECUTE`**
- **`void glEndList()`**

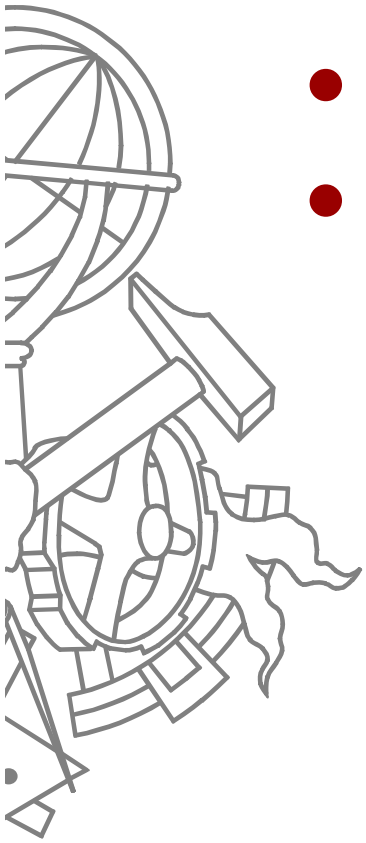
- Que instruções se podem usar
  - glBegin, glEnd e todas as possíveis aí “dentro” (glColor, glVertex, ...)
  - Transformações, projecções
  - Execução de outras *display lists*

Utilizar lista



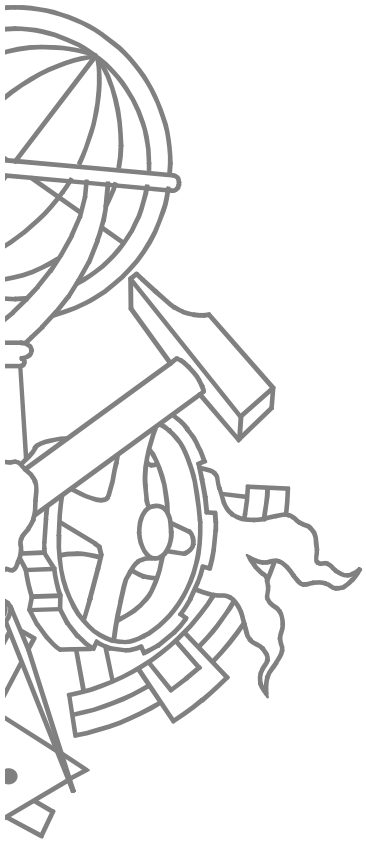
- `glCallList(list)`
- `glCallLists(GLsizei n, GLenum type, const GLvoid *lists)`

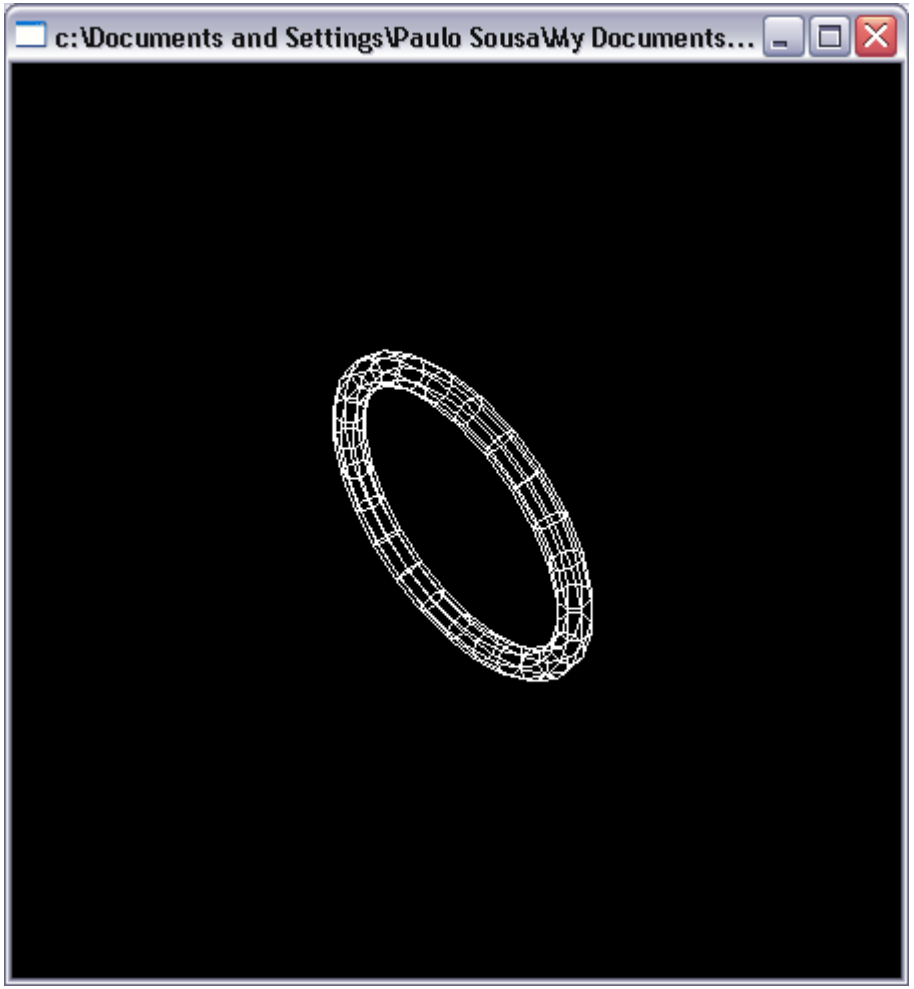
Outras operações



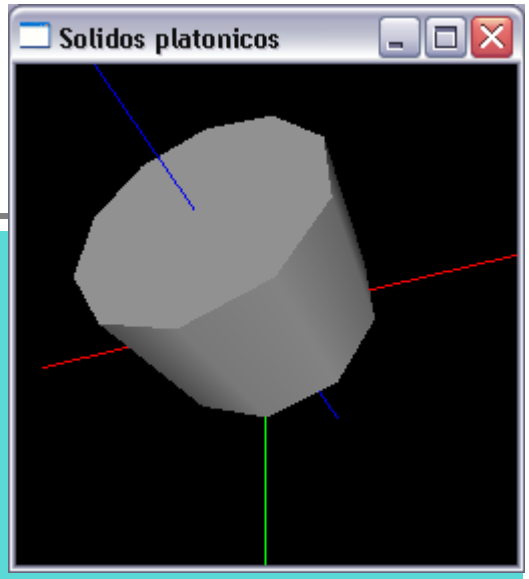
- `glIsList(GLuint i)`
- `glDeleteLists(GLuint list, GLsizei range)`

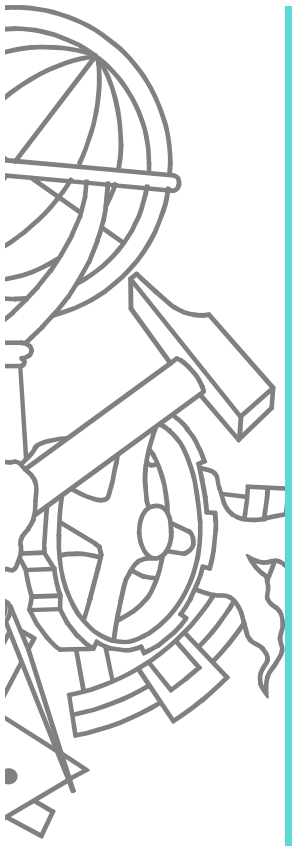
Demo





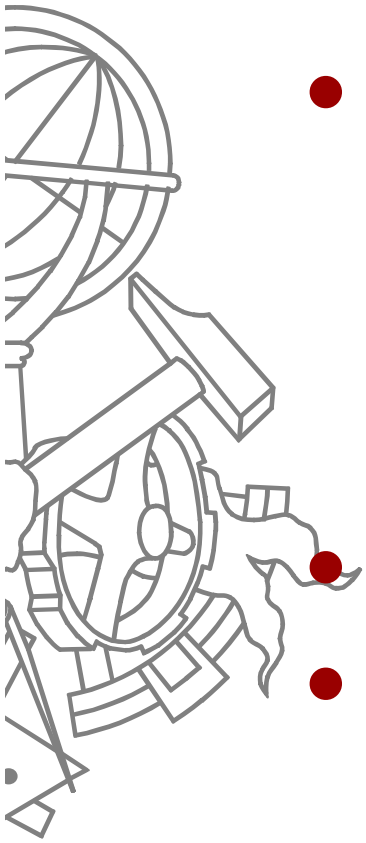
Cilindro com topo e fundo





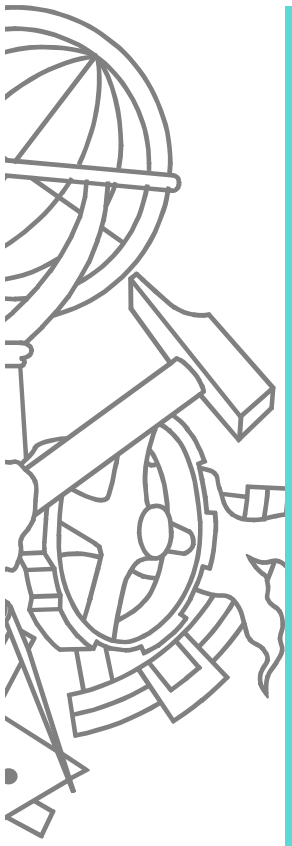
```
int cylinderWithTopAndBottom(GLenum mode)
{
   int list = glGenLists(2);
   GLUquadricObj* qobj = gluNewQuadric();
   gluQuadricDrawStyle(qobj, mode);
   gluQuadricNormals(qobj, GLU_SMOOTH);

   glNewList(list+1, GL_COMPILE);
        gluDisk(qobj, 0, 1.5, stacks, slices);
   glEndList();
   glNewList(list, GL_COMPILE);
        glCallList(list+1);
        gluCylinder(qobj, 1.5, 1.5, 2, slices, stacks);
        glTranslatef(0, 0, +2);
        glCallList(list+1);
   glEndList();

   gluDeleteQuadric(qobj);

   return list;
}
```

Animações



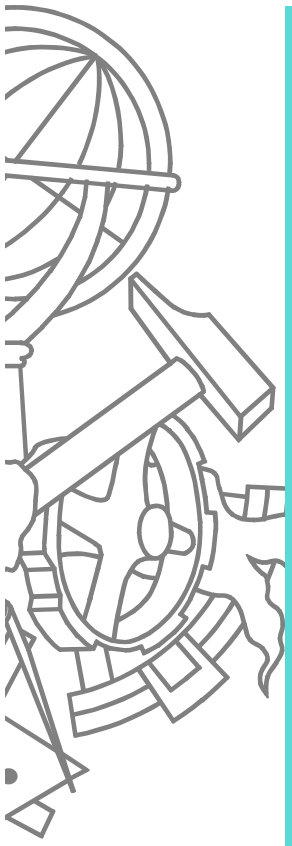
- *Double buffer*
  - Desenhar próximo *frame* num *buffer* escondido e não no *buffer* de ecrã
  - Quando a cena estiver completa, trocar o *buffer* de ecrã pelo *buffer* escondido
- **`glutInit(GLUT_DOUBLE)`**
- **`glutSwapBuffers()`** em vez de **`glFlush()`** na *callback* de *display*

Animação usando GLUT



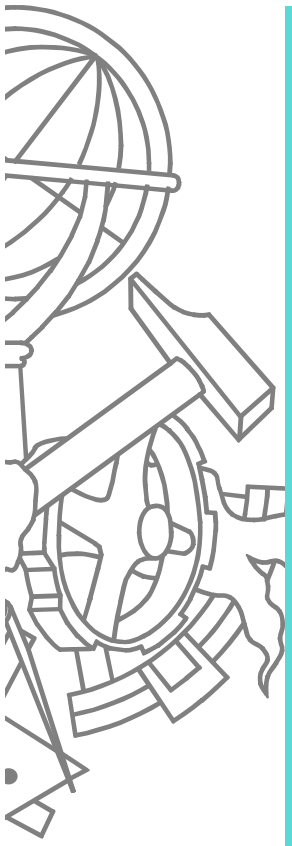
```
#include <GL/glut.h>

static GLfloat spin = 0.0;

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize (250, 250);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel (GL_FLAT);
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMouseFunc(mouse);
   glutMainLoop();
   return 0;
}
```

Animação usando GLUT



```
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glBegin(GL_QUADS);
   glVertex2f(25*cos(RAD(spin)), 25*sin(RAD(spin)));
   glVertex2f(25*cos(RAD(spin+90)), 25*sin(RAD(spin+90)));
   glVertex2f(25*cos(RAD(spin+180)), 25*sin(RAD(spin+180)));
   glVertex2f(25*cos(RAD(spin+270)), 25*sin(RAD(spin+270)));
   glEnd();
   glutSwapBuffers();
}
```

Animação usando GLUT



```
void mouse(int button, int state, int x, int y)  {
   switch (button) {
      case GLUT_LEFT_BUTTON:
         if (state == GLUT_DOWN) glutIdleFunc(spinDisplay);
         break;
      case GLUT_MIDDLE_BUTTON:
         if (state == GLUT_DOWN) glutIdleFunc(NULL);
         break;
      default:
         break;
   }
}

void spinDisplay(void) {
   spin = spin + 2.0;
   if (spin > 360.0)
        spin = spin - 360.0;
   glutPostRedisplay();
}
```

Usando temporizadores

```
...

void main(int argc, char** argv)
{
    ...
   glutTimerFunc(10, anima, 1);
   glutMainLoop();
}

void anima(int v)
{
   glutTimerFunc(10, anima, 1);
   spin = spin + 2.0;
   if (spin > 360.0)
        spin = spin - 360.0;
   glutPostRedisplay();
}
```

Demo