

Otros archivos \*.cpp para computación grafica, sacados del internet.

Copia cada uno a un block de notas y guardalos como se recomienda al inicio. No Olvidar quitarle el GL o *sino crear un directorio GL y ahí dentro colocar los directorios: include y lib.*

```
#include <stdio.h> //luces.cpp
#include <gl/openglut.h>
enum LIGHT_TYPE { AMBIENT, DIFFUSE, SPECULAR };
static int ModifyLight = AMBIENT;
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shininess[] = {50.0};
GLfloat ambient_value[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat diffuse_value[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat specular_value[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
GLfloat Height;
GLfloat Width;
GLfloat inc;
void resetLights(void)
{
    ambient_value[0] = 0.0;
    ambient_value[1] = 0.0;
    ambient_value[2] = 0.0;
    ambient_value[3] = 1.0;
    diffuse_value[0] = 1.0;
    diffuse_value[1] = 1.0;
    diffuse_value[2] = 1.0;
    diffuse_value[3] = 1.0;
    specular_value[0] = 1.0;
    specular_value[1] = 1.0;
    specular_value[2] = 1.0;
    specular_value[3] = 1.0;
    light_position[0] = 1.0;
    light_position[1] = 1.0;
    light_position[2] = 1.0;
    light_position[3] = 0.0;
}
void init(void)
{
    glClearColor ( 0.0, 0.0, 0.0, 0.0 );
    glShadeModel(GL_SMOOTH);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular );
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_value );
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse_value );
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular_value );
    glLightfv(GL_LIGHT0, GL_POSITION, light_position );
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
}
void setLightValue( int direction )
{
    if (direction == GLUT_KEY_UP | GLUT_KEY_DOWN)
    {
        if (direction == GLUT_KEY_UP)
            inc = 0.1;
        else inc = -0.1;
        if ( ModifyLight == AMBIENT )
        {
            ambient_value[0] += inc;
        }
    }
}
```

```

        ambient_value[1] += inc;
        ambient_value[2] += inc;
        printf("ambient %f %f %f\n", ambient_value[0],
            ambient_value[1], ambient_value[2] );
    }
    else if (ModifyLight == DIFFUSE )
    {
        diffuse_value[0] += inc;
        diffuse_value[1] += inc;
        diffuse_value[2] += inc;
        printf("diffuse %f %f %f\n", diffuse_value[0],
            diffuse_value[1], diffuse_value[2] );
    }
    else // ModifyLight == Specular )
    {
        specular_value[0] += inc;
        specular_value[1] += inc;
        specular_value[2] += inc;
        printf("specular %f %f %f\n",
specular_value[0],
            specular_value[1], specular_value[2] );
    }
}
}
void specialkeys( int key, int x, int y )
{
    switch (key) {
        case GLUT_KEY_DOWN:
            setLightValue( GLUT_KEY_DOWN );
            break;
        case GLUT_KEY_UP:
            setLightValue( GLUT_KEY_UP );
            break;
        case GLUT_KEY_HOME:    resetLights(); break;
        case GLUT_KEY_END:    exit(0);    break;
        default: break;
    }
    glutPostRedisplay();
}

void myKeyboard(unsigned char key, int x, int y)
{
    switch(key) {
        case 'a': ModifyLight = AMBIENT; break;
        case 'd': ModifyLight = DIFFUSE; break;
        case 's': ModifyLight = SPECULAR; break;
        default:
            break;
    }
    glutPostRedisplay();
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_value );
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse_value );
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular_value );
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glutSolidSphere(1.0, 40, 80);
    glutSwapBuffers();
}

```

```

}

void reshape( int w, int h)
{
    Height = h;
    Width = w;
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode( GL_PROJECTION);
    glLoadIdentity();
    if (w<=h)
        glOrtho(-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,
                1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
    else
        glOrtho(-1.5*(GLfloat)w/(GLfloat)h,
                1.5*(GLfloat)w/(GLfloat)h, -10.0,
10.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(1,1,10, 0,0,0,0,1,0);
}

void main(void)
{
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Testando iluminacao");
    init();
    glutSpecialFunc(specialkeys);
    glutKeyboardFunc(myKeyboard);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
}

```

```

.....
#include <stdio.h> //movluc.es.cpp
#include <gl/openglut.h>
/*
    Example taken from a computer graphics course authored by
    XXXXXX
    modified by Carla Freitas, november 2003
    -- The light position is translated with the mouse cursor
        using the callback set by glutMotionFunc;
        light position is reinitialized to (0.0, 0.0, 10.0)
        by a click on right button captured by glutMouseFunc callback.

    These comments are from the original slides:
    Light source positions are modified through
    the glLightfv() GL_POSITION parameter name value.
    The location of the source is provided in the light position
    parameter field which provides a homogeneous coordinate (x,y, z, w ).
    An infinite light source or one that is considered infinitely far away
    is specified by setting w=0. A local light source or one that is
    considered to be local to the scene is specified by setting w=1. The
    default value for a light is (0.0, 0.0, 1.0, 0.0) (i.e, from 1 on the
    z-axis and infinite).

```

A Light source may move with the eyepoint or remain fixed in a scene and not move as the view moves. A light source that moves with the eyepoint could simulate the effect that a miner's helmet has with a movable light source

To create a light source that moves with your viewpoint, bind the light BEFORE your viewing transformation.

To create a Light source that is fixed in a position in your scene, bind the light AFTER your viewing transformation

```
*/

GLfloat light_position_x = 0.0;
GLfloat light_position_y = 0.0;
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shininess[] = { 50.0 };
// luz no infinito = direcional (w=0.0)
GLfloat light_position[] = { 0.0, 0.0, 10.0, 0.0 };
// GLfloat light_position2[] = { -1.0, -1.0, -1.0, 0.0 };
GLfloat Height;
GLfloat Width;

void init(void)
{
    glClearColor ( 0.0, 0.0, 0.0, 0.0 );
    glShadeModel(GL_SMOOTH);

    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position );

    glEnable(GL_LIGHTING); // habilita iluminação

    glEnable(GL_LIGHT0); // habilita luz 0
    glEnable(GL_DEPTH_TEST);
}

/*
The mouse action call back is written such that
the x, y position of the mouse cursor is mapped
to view space x, y coordinates.
These coordinates are plugged back into the glLightfv() function
in the display callback function resulting
in the direction of the light source moving with the mouse cursor.
Note that a right mouse click repositions the light source
along a new x,y coordinate. A left mouse click will also use
a z coordinate to relocate the source of LIGHT0.
*/

void mousefunc(int button, int state, int x, int y )
{
    /* versão original
    light_position_x = x - Width/2;
    light_position_y = (Height/2) - y ;

    if ( button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN )
    {
```

```

        light_position[0] = light_position_x;
        light_position[1] = light_position_y;
    }
    if ( button == GLUT_LEFT_BUTTON && state == GLUT_DOWN )
    {
        light_position[0] = -light_position_x;
        light_position[1] = -light_position_y;
        light_position[2] = -light_position_y;
    }
    glutPostRedisplay();
    return;
*/

// new version - click on right button resets light position
if ( button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN )
{
    light_position_x = 0.0;
    light_position_y = 0.0;
}
light_position[0] = light_position_x;
light_position[1] = light_position_y;
glutPostRedisplay();

}

void mouseMovingFunc(int x, int y )
{
    /* moving the mouse either with the right or left button pressed
       moves the light position
       Notice that the movement sometimes is "jumpy"
       because I'm not taking the actual translation of the mouse*/
    light_position_x = x - Width/2;
    light_position_y = (Height/2) - y ;

    light_position[0] = light_position_x;
    light_position[1] = light_position_y;

    glutPostRedisplay();
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glutSolidSphere(1.0, 40, 80);
    glutSwapBuffers();
    glFlush();
}

void reshape( int w, int h)
{
    Height = h;
    Width = w;
    printf("w = %d, h = %d\n", w, h );
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode( GL_PROJECTION);
    glLoadIdentity();

```

```

    if (w<=h)
        // redimensiona em y
        glOrtho(-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,
                1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
    else
        // se largura maior que altura, redimensiona em x
        glOrtho(-1.5*(GLfloat)w/(GLfloat)h,
                1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void main(void)
{
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Moving lighth example");
    init();
    glutDisplayFunc(display);
    glutMouseFunc(mousefunc);
    glutMotionFunc(mouseMovingFunc); // read the GLUT spec!!
    glutReshapeFunc(reshape);

    glutMainLoop();
}

```

---

```

#include <stdio.h> //atenuacionluz.cpp
#include <stdlib.h> //se adiciono
#include <gl/openglut.h>
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shininess[] = { 50.0 };
GLfloat light_position[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat Height;
GLfloat Width;
static GLdouble spin= 0.0;
GLdouble eyex=0.0, eyey=0.0, eyez=5.0;
void setEyePoint(void)
{
    eyex=0.0; eyey=0.0; eyez = 5.0; return;
}

void init(void)
{
    glClearColor ( 0.0, 0.0, 0.0, 0.0 );
    glShadeModel(GL_SMOOTH);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position );
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
}

void display(void)
{

```

```

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glPushMatrix();
        gluLookAt( eyex, eyey, eyez, 0.0, 0.0, 0.0, 0, 1, 0 );
        glutSolidTorus( 0.275, 0.85, 20, 40);
    glPopMatrix();
    glutSwapBuffers();
    glFlush();
}

void idlefunc(void)
{
    glutPostRedisplay();
}

void reshape( int w, int h)
{
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode( GL_PROJECTION);
    glLoadIdentity();
    gluPerspective( 45.0, w/h, 1.0, 100.0 );
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLoadIdentity();
}

void specialkeys( int key, int x, int y )
{
    switch (key) {
    case GLUT_KEY_LEFT:      eyex--; break;
    case GLUT_KEY_RIGHT:    eyex++; break;
    case GLUT_KEY_DOWN:     eyey--; break;
    case GLUT_KEY_UP:       eyey++; break;
    case GLUT_KEY_PAGE_UP:  eyez++; break;
    case GLUT_KEY_PAGE_DOWN: eyez--; break;
    case GLUT_KEY_HOME:     setEyePoint(); break;
    case GLUT_KEY_END:      exit(0); break;
    default:
        break;
    }
    glutPostRedisplay();
}

void myKeyboard(unsigned char key, int x, int y)
{
    switch(key) {
    case 'x': eyex=5.0; eyey=0.0; eyez=0.0; break;
    case 'y': eyex=0.05; eyey=5.0; eyez=0.0; break;
    case 'z': eyex=0.0; eyey=0.0; eyez=5.0; break;
    case 'c':
        glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
        glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.0);
        glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.0 );
        break;
    case 'l':
        glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.0);
        glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.25);
        glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.0 );
        break;
    case 'q':
        glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.0);

```

```

        glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.0);
        glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.25 );
        break;
    case 'o':
        glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
        glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.25);
        glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.25 );
        break;
    default:
        break;
    }
    glutPostRedisplay();
}

void main(void)
{

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Light Atenuation Example");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc( idlefunc );
    glutSpecialFunc( specialkeys );
    glutKeyboardFunc( myKeyboard );
    glutMainLoop();

}

```

---

```

#include <stdio.h> //propiedadesmateriales.cpp
#include <stdlib.h> //SE ADICIONO
#include <openglut.h>
GLdouble eyex=0.0, eyey=0.0, eyez=15.0;
static int StopFlag = 1;

void setEyePoint(void)
{
    eyex=0.0; eyey=0.0; eyez = 15.0; return;
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glEnable( GL_DEPTH_TEST );
    glEnable( GL_LIGHT0 );
    //glShadeModel(GL_FLAT);
}

void display (void)
{
    static GLdouble rotation=0.0;

    GLfloat redAmbient[]= { 0.3, 0.1, 0.1, 1.0 };
    GLfloat redDiffuse[] = { 1.0, 0.0, 0.0, 1.0 };
}

```

```

GLfloat blueAmbient[] = { 0.1, 0.1, 0.3, 1.0 };
GLfloat blueDiffuse[] = { 0.0, 0.0, 1.0, 1.0 };
GLfloat dark [] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat yellowDiffuse[] = { 1.0, 1.0, 0.0, 1.0};
GLfloat yellowEmission[] = { 0.6, 0.6, 0.0, 1.0 };
GLfloat yellowSpecular[] = { 0.6, 0.6, 0.0, 1.0 };
GLfloat defaultEmission[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat whiteSpecular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat greenAmbient[] = { 0.0, 1.0, 0.0, 1.0 };
GLfloat greenDiffuse[] = { 0.0, 1.0, 0.0, 1.0 };
GLfloat greenSpecular[] = { 0.0, 1.0, 0.0, 1.0 };
GLfloat defaultSpecular[] = { 0.0, 0.0, 0.0, 1.0 };
glClearColor(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glColor3f(0.0, 0.0, 0.0);
glLoadIdentity();
// Set up the world to view map with variable eye vector,
// a look at point at the origin and and up vector that points
// in the positive y axis direction.
gluLookAt(eyex, eyey, eyez, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
glEnable( GL_LIGHTING );
glPushMatrix();
    glMaterialfv( GL_FRONT, GL_EMISSION, defaultEmission);
    glMaterialfv( GL_FRONT, GL_AMBIENT, redAmbient );
    glMaterialfv(GL_FRONT, GL_DIFFUSE, yellowDiffuse );
    glMaterialfv(GL_FRONT, GL_SPECULAR, whiteSpecular);
    glMaterialf(GL_FRONT, GL_SHININESS, 128.0);
    glRotatef(90.0, 1.0, 0.0, 0.0); // Adjust tilt in vertical
    glRotatef(rotation++/2, 0.0, 0.0, 1.0); // Sun's rotation
    glutSolidSphere(2.0, 50, 80); // Draw Sun
    glPopMatrix();
    glMaterialfv( GL_FRONT, GL_AMBIENT, blueAmbient );
    glMaterialfv( GL_FRONT, GL_DIFFUSE, blueDiffuse );
    glMaterialf( GL_FRONT, GL_SHININESS, 20.0);

    glPushMatrix(); // Matrix for Saturn's planetary system
    glRotatef(70.0, 1.0, 0.0, 0.0); // Saturn's planetary
system tilts
    glRotatef(1 - rotation/5, 0.0, 0.0, 1.0); //revolutions
around the Sun
    glTranslatef(5.0, 0.0, 0.0 );
    glPushMatrix(); // Individual attributes for Saturn
    glMaterialfv( GL_FRONT, GL_AMBIENT, blueAmbient );
    glMaterialfv( GL_FRONT, GL_DIFFUSE, blueDiffuse );
    glMaterialf( GL_FRONT, GL_SHININESS, 20.0);
    glRotatef(rotation*8, 0.0, 0.0, 1.0); // Saturns rotation
    glutSolidSphere( 0.8, 24, 48 ); // Draw Saturn
    glMaterialfv( GL_FRONT, GL_DIFFUSE, dark );

    glutWireTorus(0.2, 0.9, 12, 12); // Draw Saturn's
rings

    glMaterialfv( GL_FRONT, GL_DIFFUSE, redDiffuse );
    glutWireTorus(0.2, 1.3, 12, 12); // Draw Saturn's rings

    glPopMatrix();
// Draw an equatorial moon
glPushMatrix();
    glMaterialfv( GL_FRONT, GL_AMBIENT, redAmbient );
    glMaterialfv( GL_FRONT, GL_DIFFUSE, redDiffuse );
    glMaterialf( GL_FRONT, GL_SHININESS, 200.0);
    glRotatef(rotation, 0.0, 0.0, 1.0); // equatorial orbit
    glTranslatef(2.0, 0.0, 0.0 );

```

```

        glRotatef(rotation*2, 0.0, 0.0, 1.0); // Lunar rotation
        glutSolidSphere(0.5, 16, 32);
        glPopMatrix();
// Draw a polar moon
        glPushMatrix();
            glMaterialfv( GL_FRONT, GL_AMBIENT, greenAmbient );
            glMaterialfv(GL_FRONT, GL_DIFFUSE, greenDiffuse );
            glMaterialfv(GL_FRONT, GL_SPECULAR, greenSpecular);
            glMaterialf(GL_FRONT, GL_SHININESS, 128.0);
        glRotatef(rotation, 1.0, 0.0, 0.0); // polar orbit around
Saturn
            glTranslatef(0.0, 0.0, 2.75);
            glRotatef(90.0, 1.0, 0.0, 0.0); // adjust moon's
tilt in vertical
            glRotatef(rotation*20, 0.0, 0.0, 1.0); // Lunar
rotation
            glutSolidSphere(0.25, 16, 32);
            glPopMatrix();
        glPopMatrix();
        glFlush();
        glutSwapBuffers();
    }
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, 100.0);
    glMatrixMode(GL_MODELVIEW);
}
void idle()
{
    if (StopFlag%2) glutPostRedisplay();
}
void specialkeys( int key, int x, int y )
{
    switch (key) {
        case GLUT_KEY_LEFT:    eyex--; break;
        case GLUT_KEY_RIGHT:  eyex++; break;
        case GLUT_KEY_DOWN:   eyey--; break;
        case GLUT_KEY_UP:     eyey++; break;
        case GLUT_KEY_PAGE_UP: eyez++; break;
        case GLUT_KEY_PAGE_DOWN: eyez--; break;
        case GLUT_KEY_HOME:   setEyePoint(); break;
        case GLUT_KEY_END:    exit(0); break;
        default:
            break;
    }
    glutPostRedisplay();
}
void myKeyboard(unsigned char key, int x, int y)
{
    switch(key) {
        case 'x': eyex=12.0; eyey=0.0; eyez=0.0; break;
        case 'y': eyex=0.05; eyey=12.0; eyez=0.0; break;
        case 'z': eyex=0.0; eyey=0.0; eyez=12.0; break;
        case 's': StopFlag++; break;
        default:
            break;
    }
}

```

```

        glutPostRedisplay();
    }
void main(void)
{
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH );
    glutInitWindowSize(500, 500);
    glutInitWindowPosition( 100, 100);
    glutCreateWindow("Demo propriedades");
    init();
    glutDisplayFunc(display);
    glutSpecialFunc( specialkeys );
    glutKeyboardFunc( myKeyboard );
    glutReshapeFunc( reshape);
    glutIdleFunc( idle );
    glutMainLoop();
}

```

---

```

// Iluminacao.c //iluminacion.cpp
// Autora: Isabel Harb Manssour (2001)
// Modificado por Carla Freitas (2002)

// Visualização de objetos 3D com a inserção de uma fonte de luz
// Baseado nos exemplos do livro OpenGL SuperBible, 2nd Edition
// Richard S. Wright Jr.

#include <openglut.h>

GLfloat angle=45;
GLfloat fAspect;

// variaveis para controle da rotacao com o mouse

static bool rotacionaX = false;
static bool rotacionaY = false;

// Parâmetros de iluminação
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, 80.0, 200.0, 1.0};

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

// Função callback de desenho - chamada a cada re-exibição da janela

void RenderScene(void)
{
    // Limpa a janela e o depth buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}

```

```

// Troca cor corrente para azul
glColor3f(0.0f, 0.0f, 1.0f);

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

// Execução dos comandos de desenho
glutSwapBuffers();

}

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

    // Define a refletância do material
    // Usa apenas propriedade especular
    glMaterialfv(GL_FRONT, GL_SPECULAR, especularidade);

    // Define a concentração do brilho da superfície
    glMateriali(GL_FRONT, GL_SHININESS, especMaterial);

    // Habilita a definição da cor do material
    // a partir da cor corrente
    // Experimente colocar em comentário a linha abaixo ...
    glEnable(GL_COLOR_MATERIAL);

    // 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 o uso de iluminação
    glEnable(GL_LIGHTING);

    // Habilita a luz de número 0
    glEnable(GL_LIGHT0);

    // GL_FLAT = Habilita o modelo de tonalização constante (uma
    cor por face)

```

```

glShadeModel(GL_FLAT);

// GL_SMOOTH = Habilita o modelo de tonalização Gouraud
(interpolação de cores dos vertices)
// glShadeModel(GL_SMOOTH);

// Habilita o depth-buffering
glEnable(GL_DEPTH_TEST);

}

// Função usada para especificar o volume de visualização
void Viewing(void)
{
// Inicializa sistema de coordenadas de projeção

glMatrixMode(GL_PROJECTION);
glLoadIdentity();

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

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

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

}

// Chamada pela GLUT quando a janela é redimensionada
void ChangeSize(GLsizei w, GLsizei h)
{
// Para evitar 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;

Viewing();
}

```

```

    }

// Idle function
// chamada quando OpenGL nao tem nada a fazer
// pode ser usada para implementar animacao

void idleFunc()

{
    if ( rotacionaX ) {
        glRotatef( 0.7f, 1.0f, 0.0f, 0.0f );
        glutPostRedisplay();
    } else if ( rotacionaY ) {
        glRotatef( 0.7f, 0.0f, 1.0f, 0.0f );
        glutPostRedisplay();
    }
}

// trata eventos do mouse
// botao esquerdo - rotaciona em x
// botao direito - rotaciona em y

void HandleMouse ( int button, int state, int x, int y )

{
    if ( state == GLUT_DOWN ) {
        if (button == GLUT_LEFT_BUTTON )
            rotacionaX = true;
        else if (button == GLUT_RIGHT_BUTTON )
            rotacionaY = true;
    }
    else if ( state == GLUT_UP ) {
        rotacionaX = false;
        rotacionaY = false;
    }
}

// Programa Principal

void main(void)

{

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(400,350);
    glutCreateWindow("Visualizacao 3D");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    glutMouseFunc(HandleMouse);
    glutIdleFunc(idleFunc);
    SetupRC();
    glutMainLoop();
}

```

