Orientações AO PROFESSOR

TÉCNICAS DE COMPUTAÇÃO GRÁFICA

Orientações gerais

A Computação Gráfica (CG) é uma arte que vem evoluindo constantemente ao longo dos últimos anos. Podemos ver filmes, desenhos animados e jogos com uma qualidade indiscutível, em alguns momentos não é possível dizer se o desenho é realmente verdadeiro, ou seja, não é possível afirmar se é uma foto ou simplesmente uma criação.

O objetivo original do livro é abordar os conceitos em nível de programação na parte da síntese de imagem. Isso quer dizer, que queremos mostrar como criar objetos com os conceitos da CG. É claro que trabalhar com uma ferramenta de modelagem é mais simples (Blender, 3-D Max, Maia, Photoshop, e outros) e permite construir um determinado objeto mais rápido do que em um ambiente de programação.

Porém, a maioria das ferramentas de modelagem nos permite melhorar ou corrigir possíveis problemas com o uso de bibliotecas adicionais desenvolvidas pelo usuário. Essas bibliotecas podem ser incorporadas para melhorar um determinado problema que surgiu durante a modelagem e é com esta visão que o livro *Técnicas de computação gráfica* foi criado.

Para que consigamos trabalhar e ajustar nossas ferramentas de modelagem, é necessário conhecer como as funções e métodos utilizados internamente pela ferramenta funcionam. Sendo assim, a abordagem natural foi utilizar a OpenGL que é uma biblioteca gráfica livre, a qual pode ser utilizada sem restrições por qualquer pessoa. Essa API gráfica permite criar objetos sintéticos a partir de primitivas geométricas básicas.

Optou-se pelo uso da ferramenta de programação C ou C++, pois é possível utilizar os mesmos conceitos em diferentes sistemas operacionais e em conjunto o ambiente de desenvolvimento Codeblocks que também está disponível para outros sistemas operacionais. Isso permite que o aluno mude de sistema operacional, mas não precise mudar de linguagem de programação e de ambiente de desenvolvimento.

Objetivos do material didático

- Compreender os principais comando OpenGL.
- Criar objeto em 2-D.
- Criar objetos em 3-D.
- Aplicar os conceitos de iluminação.
- Criar objetos com realismo com o uso de texturas.
- Animar objetos por meio dos comandos OpenGl.

Princípios pedagógicos

O objetivo é que o aluno possa trabalhar os conceitos apresentados na prática, implementando cada etapa e com isso permitindo-o a buscar novas soluções que possibilitem aumentar a sua criatividade e seu raciocínio lógico.

Articulação do conteúdo

Como o conteúdo envolve a criação e a articulação dos pensamentos e ideias, é necessário que o docente esteja integrado com seus colegas de disciplinas, como programação, artes, modelagem 2D e 3D e matemática.

Atividades complementares

Desenvolvimento de atividades práticas que permitam o aluno extrapolar os conhecimentos adquiridos com o uso de pesquisa em outros materiais, como internet e livros para a resolução dos problemas já apresentados, porém com novas técnicas.

Sugestão de leitura

Shreiner, Dave; Sellers, Graham; Kessenich, John and Licea-Kane, Bill. **OpenGL Programming Guide: The Official Guide to Learning OpenGL**. Addison-Wesley Professional. 8 edition. 2013.

Sellers, Graham; Wright, Richard S. and Haemel, Nicholas. **OpenGL® SuperBible**. Addison-Wesley Professional. 6 edition. 2013.

Azevedo, Eduardo e Conci, **Aura. Computação Gráfica – Teoria e Prática**. Elsevier. 2003.

Sugestão de Planejamento

Este manual foi elaborado para dar suporte ao livro *Computação gráfica*. O livro foi estruturado para ser trabalhado em sala de aula, em 80 horas e essas horas podem ser divididas em dois semestres. Quanto à sequência do conteúdo não há uma regra rígida que deva ser seguida, embora seja interessante dar continuidade aos assuntos expostos para melhor aprendizagem.

Semestre 1

Primeiro bimestre

Capítulo 1– Introdução à computação gráfica

Capítulo 2 – Iniciando os trabalhos com OpenGL

Capítulo 3 – Primitivas geométricas 3-D

Objetivos

- Introduzir computação gráfica: visão geral; code::Blocks e OpenGL.
- Iniciar os trabalhos com OpenGL.
- Compreender primitivas geométricas em 3-D.

Atividades

Iniciar o aluno nos conceitos de computação gráfica, e permitir a compreensão de como instalar e configurar o ambiente de trabalho Codeblocks, necessário para o desenvolvimento das atividades. Outro ponto fundamental é compreender o funcionamento da OpenGL.

Segundo bimestre

Capítulo 4 – Como alterar as características do objeto

Capítulo 5 – Curvas e superfícies

Capítulo 6 – Como iluminar os objetos e o ambiente

Objetivos

- Aprender como alterar as características do objeto: translação; escala; rotação; reflexão e visão em perspectiva.
- Saber sobre curvas e superfícies como: manipular, definir e criar.

Atividades

Auxiliar o aluno a compreender os tipos de primitivas existentes em domínio 3-D. Fazer com que o aluno aplique os conceitos básicos para manipulação de objetos. Permitir o aluno compreender as diferenças entre curvas e superfícies e o seus comportamentos.

Semestre 2

Primeiro bimestre

Capítulo 7 – Criação e manipulação de sólidos

Capítulo 8 – Como tornar as coisas reais – textura

Objetivos

- Entender criação e manipulação de sólidos: criar cubo sólido, criar objeto mais complexo.
- Saber como tornar as coisas reais – texturas.

Atividades

Possibilitar o aluno aplicar efeitos de iluminação em diferentes modos sobre a cena em desenvolvimento. Aplicar os conceitos de modelagem 2-D e 3-D para criação de objetos. Permitir o uso de textura para modelar objetos com características reais.

Segundo bimestre

Capítulo 9 – Renderização

Capítulo 10 – Animação

Objetivos

- Aprender renderização
- Trabalhar animação.

Atividades

Possibilitar a criação de cenas com vários objetos que representem o mundo real. Permitir ao aluno compreender como um determinado objeto pode se deslocar no tempo, ou percorrer um determinado percurso. Aplicar todos os conceitos aprendidos durante o curso.

Todos os capítulos têm alguma atividade a ser resolvida, porém todas já possuem respostas. O ideal é que o docente auxilie na busca e construção de novas soluções para o mesmo problema. Isso permitirá ao aluno melhor fixação do conteúdo apresentado, além de exercitar a capacidade individual de cada um.

Respostas didáticas e respostas das atividades

Capítulo 1

Orientações

Nesse capítulo é apresentado de forma abrangente, o que é computação gráfica, e para isso o docente deve apresentar imagens das diferentes formas da computação gráfica e suas aplicações, como: síntese de imagens, processamento de imagens e análise de imagens.

Respostas – página 20

1) Método glutWireSphere, assim:

   determina o raio da esfera;

   número de divisões horizontas no objeto;

   número de divisões verticais no objeto.

2) Método glutWireCube(p1):

   Determina o tamanho do lado ou tamanho do vértice do cubo.

Capítulo 2

Orientações

Mostrar as etapas de instalação do ambiente de programação Codeblocks, isso é necessário, pois os alunos podem ter trabalhado com outro ambiente de desenvolvimento. Após essa etapa é ideal fazer com que os alunos façam o *download* dos arquivos para instalação e configuração da OpenGL, dessa forma, é necessário a orientação para a correta instalação dos arquivos. Fazer testes com a ferramenta para mostrar o seu correto funcionamento.

No caso das primitivas geométricas básica em 2-D, é ideal que os conceitos antes de ser implementados, sejam apresentados em quadro ou Datashow, para que os alunos tenham noção do que é uma primitiva geométrica e como ela funciona. Caso seja de interesse do docente, este pode relacionar a parte matemática da criação de uma determinada primitiva geométrica 2-D.

Respostas – página 37

1) Usando o comando lColor3d:

   indica a cor vermelha variando de 0 até 1 com passo 0;

   indica a cor verde variando de 0 até 1 com passo 0.1;

   indica a cor azul variando de 0 até 1 com passo 0.1.

   Neste caso, para identificar a cor usa-se o valor 0 (zero) para informar que não há cor ou um valor no intervalo de 0.1 até 1, variando em 0.1. No caso da cor vermelha pura, então é colocado o valor 1. Nos demais casos, há tonalidades de vermelho diferentes, como o valor, 0.1 ou o valor 0.2 ou o valor 0.3.

2) O método glutWireCone possui quatro parâmetros, ou seja, glutWireCone(p1, p2, p3, p4), assim:

   determina o tamanho da base do cone;

   determina a altura do cone;

   determina a quantidade de divisões no eixo vertical;

   determina a quantidade de divisões no eixo horizontal.

3)

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


static void resize(int width, int height)
{

    glViewport(0, 0, width, height);

}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(0,1,0);

    int centroX = glutGet(GLUT_WINDOW_WIDTH) / 2;
    int centroY = glutGet(GLUT_WINDOW_HEIGHT) / 2;

    glBegin(GL_LINES);
        glVertex2f(-0.2,0);
        glVertex2f(0.2,0);
        glVertex2f(0,0.2);
        glVertex2f(0,-0.2);
    glEnd();
    glutSwapBuffers();
}


int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("GLUT Shapes");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);


    glutMainLoop();
}
```

4)

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


static void resize(int width, int height)
{

    glViewport(0, 0, width, height);

}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(0,1,0);

    glutWireSphere(1,10,5);
    glutSwapBuffers();
}


int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("GLUT Shapes");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);


    glutMainLoop();
}
```

5)

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


static void resize(int width, int height)
{

    glViewport(0, 0, width, height);

}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,0,0);
    /*
        parametros do glutWireCone
        raio = 1;
        altura = 1;
        numero divisões eixo x = 15;
        numero divisões exio y = 5;
    */
    glutWireCone(1,1, 15, 5);
    glutSwapBuffers();
}


int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("GLUT Shapes");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);


    glutMainLoop();
}
```

Capítulo 3

Orientações

No caso das primitivas geométricas 3-D, é ideal que os conceitos antes de ser implementados, sejam apresentados em quadro ou Datashow, para que os alunos tenham noção do que é uma primitiva geométrica e como ela funciona. Caso seja de interesse do docente, este pode relacionar a parte matemática da criação de uma determinada primitiva geométrica 3-D.

Respostas – página 51

1)

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


static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,1,1);
    glPointSize(5.0f);

    glBegin(GL_POINTS);
        glVertex2f(0,0);
        glVertex2f(0.45,0);
        glVertex2f(0.4,0.25);
        glVertex2f(0.25,0.4);

        glVertex2f(0,0.5);
        glVertex2f(-0.4,0.25);
        glVertex2f(-0.25,0.4);

        glVertex2f(-0.45,0);
        glVertex2f(-0.4,-0.25);
        glVertex2f(-0.25,-0.4);

        glVertex2f(0,-0.5);
        glVertex2f(0.4,-0.25);
        glVertex2f(0.25,-0.4);
    glEnd();
    glutSwapBuffers();
}


int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Janela Principal");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);


    glutMainLoop();
}
```

2)

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

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
}

static void display(void)
{

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glColor3d(0,0,1);

    glutWireCube(0.1);
    glutWireCube(0.2);
    glutWireCube(0.3);
    glutWireCube(0.4);
    glutWireCube(0.5);
    glutWireCube(0.6);

    glutSwapBuffers();
}


int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(640,480);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Janela Principal");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);

    glutMainLoop();
}
```

3) A resposta é a critério do aluno.

Capítulo 4

Orientações

As transformações geométricas alteram alguma característica de um objeto ou cena, então é necessário que o docente apresente de forma física o que ocorre com objeto ao fazer uma determinada transformação. Pois isso, nessa etapa, os alunos não têm uma clara visão de um objeto 3-D no espaço e como ocorre esta transformação.

Respostas – página 64

1)

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

static void resize(int width, int height)
{
        // determinação da área de uso
 glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-10,10,-10,10,0,50);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity() ;

}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,1,1);
    glPointSize(5.0f);

    // criação da esfera
    glPushMatrix();
        glRotated(90,1,0,0); // girando para a posição desejada
        glutWireSphere(3,15,6); // desenhando a esfera
 glPopMatrix();
    // criação do cone
    glPushMatrix();
        glColor3d(0,0,1);
        glTranslatef(0,1.5,0); // reposicionamento do cone
        glRotated(-90,1,0,0); // girando para a posição desejada
        glutWireCone(3,5,15,6); // desenhando o cone
 glPopMatrix();

    glutSwapBuffers();
}


int main(int argc, char *argv[])
{
 glutInit(&argc, argv);
 glutInitWindowSize(800,600);
 glutInitWindowPosition(10,10);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

 glutCreateWindow("Exercicios");

 glutReshapeFunc(resize);
 glutDisplayFunc(display);

 glutMainLoop();
}
```

2)

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

static void resize(int width, int height)
{
        // determinação da área de uso
 glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-10,10,-10,10,0,50);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity() ;

}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,1,1);
    glPointSize(5.0f);

    // criação da esfera
    glPushMatrix();
        glRotated(90,1,0,0); // girando para a posição desejada
        glutSolidSphere(3,15,6); // desenhando a esfera
 glPopMatrix();
    // criação do cone
    glPushMatrix();
        glColor3d(0,0,1);
        glTranslatef(0,1.5,0); // reposicionamento do cone
        glRotated(-90,1,0,0); // girando para a posição desejada
        glutSolidCone(3,5,15,6); // desenhando o cone
 glPopMatrix();

    glutSwapBuffers();
}


int main(int argc, char *argv[])
{
 glutInit(&argc, argv);
 glutInitWindowSize(800,600);
 glutInitWindowPosition(10,10);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

 glutCreateWindow("Exercicios");

 glutReshapeFunc(resize);
 glutDisplayFunc(display);

 glutMainLoop();
}
```

3)

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


static void resize(int width, int height)
{
        // determinação da área de uso
 glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-10,10,-10,10,0,50);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity() ;

}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,1,1);
    glPointSize(5.0f);

    // criação da esfera
    glPushMatrix();
        glRotated(90,1,0,0);
        glutSolidSphere(3,15,6);
 glPopMatrix();

    // adicionado o nariz
    glPushMatrix();
        glColor3d(0,1,0);
        glRotated(90,1,0,0);
        glutSolidSphere(0.5,15,6);
 glPopMatrix();

    // adicionado o olhos direito
    glPushMatrix();
        glColor3d(0,1,1);
        glTranslatef(-0.8,1.2,0);
        glRotated(90,1,0,0);
        glutSolidSphere(0.8,15,6);
 glPopMatrix();
    // adicionado o olhos esquerdo
    glPushMatrix();
        glColor3d(0,1,1);
        glTranslatef(0.8,1.2,0);
        glRotated(90,1,0,0);
        glutSolidSphere(0.8,15,6);
 glPopMatrix();

    // adicionado a boca
    glPushMatrix();
        glColor3d(1,0,0);
        glTranslatef(0,-1.6,0);
```

```
            glRotated(90,1,0,0);
            glScalef(1,0.1,0.1);
            glutSolidSphere(0.8,15,6);
    glPopMatrix();

        // criação do cone
        glPushMatrix();
            glColor3d(0,0,1);
            glTranslatef(0,1.5,0); // reposicionamento do cone
            glRotated(-90,1,0,0); // girando para a posição desejada
            glutSolidCone(3,5,15,6); // desenhando o cone
    glPopMatrix();

        glutSwapBuffers();
    }


        int main(int argc, char *argv[])
        {
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Exercicios");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);

    glutMainLoop();
        }
```

4) A resposta é a critério do aluno.

5) A resposta é a critério do aluno.

Capítulo 5

Orientações

As curvas e superfícies permitem modelar objetos com uma complexidade maior do que as primitivas geométricas básicas. Para isso, é necessário utilizar inicialmente alguma ferramenta gráfica que permita desenhar uma curva ou superfície e o que acontece quando há uma alteração do formato da curva ou superfície.

Respostas – página 79

1)

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

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0,100,0,100,0,100);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,1,1);


        glPushMatrix();
            glTranslated(x,x,0);
            glutWireCube(4);
        glPopMatrix();

    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Janela Principal");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);

    glutMainLoop();
}
```

2)

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

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-10,10,-10,10,0,10);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

}

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

    glScalef(1.5,2.5,0);
    glPushMatrix();
        glColor3d(0,0,1);
        glutWireCube(4);
    glPopMatrix();

    glPushMatrix();
        glTranslated(0,2,0);
        glBegin(GL_TRIANGLE_STRIP);
            glColor3d(0,0,1);
            glVertex3d(-2,0,0);
            glVertex3d(2,0,0);
            glVertex3d(0,2,0);
        glEnd();
    glPopMatrix();

    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Janela Principal");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);

    glutMainLoop();
}
```

3)

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

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-10,10,-10,10,0,10);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

}

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

    for (int rot = 10;rot <= 180;rot+=10){
        glPushMatrix();
            glColor3d(1,0,0);
            glRotated(rot,0,0,1);
            glutWireCube(4);
        glPopMatrix();
    }
    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Janela Principal");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);

    glutMainLoop();
}
```

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

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-20,20,-20,20,0,20);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

}

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

    glPushMatrix();
        glTranslated(5,0,0);
        glScalef(1.5,2.5,0);
        glPushMatrix();
            glColor3d(0,0,1);
            glutWireCube(4);
        glPopMatrix();

        glPushMatrix();
            glTranslated(0,2,0);
            glBegin(GL_TRIANGLE_STRIP);
                glColor3d(0,0,1);
                glVertex3d(-2,0,0);
                glVertex3d(2,0,0);
                glVertex3d(0,2,0);
            glEnd();
        glPopMatrix();
    glPopMatrix();

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // reflexão
    glPushMatrix();
        glTranslated(-5,0,0);
        glScalef(1.5,2.5,0);
        glPushMatrix();
            glColor3d(0,0,1);
            glutWireCube(4);
        glPopMatrix();

        glPushMatrix();
            glTranslated(0,2,0);
            glBegin(GL_TRIANGLE_STRIP);
                glColor3d(0,0,1);
                glVertex3d(-2,0,0);
                glVertex3d(2,0,0);
```

```
                glVertex3d(0,2,0);
            glEnd();
        glPopMatrix();
    glPopMatrix();

    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Janela Principal");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);

    glutMainLoop();
}
```

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

int rotsol = 0, rotterra = 0;

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0,((GLfloat)width/ (GLfloat)
height),0.0,60.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,1,1);
    glPushMatrix();
        glTranslated(0,0,-30);
        glRotated(10,1,1,1);
        glutWireSphere(3,10,10);
    glPopMatrix();

    glPushMatrix();
        glLoadIdentity();
        glRotated(rotsol,0,1,0);
```

```
            glPushMatrix();
                glTranslated(3,0,-30);
                glRotated(rotterra,0,1,0);
                glutWireSphere(2,10,10);
            glPopMatrix();
        glPopMatrix();

        rotterra +=2;
        rotsol +=1;

        glutSwapBuffers();
    }

    static void idle(void)
    {
        glutPostRedisplay();
    }


    int main(int argc, char *argv[])
    {
        glutInit(&argc, argv);
        glutInitWindowSize(800,600);
        glutInitWindowPosition(10,10);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

        glutCreateWindow("Janela Principal");

        glutReshapeFunc(resize);
        glutDisplayFunc(display);
        glutIdleFunc(idle);

        glutMainLoop();
    }
```

Capítulo 6

Orientações

Nesse capítulo é apresentado o tipo de luzes existentes, o ideal neste caso, é fazer a correlação com os tipos de luzes que utilizamos no dia a dia e como isso afeta o nosso ambiente. Isso será semelhante ao que ocorre com o cenário e com os objetos quando inserido algum tipo de luminosidade. Uma boa forma é ter uma caixa com um bocal de lâmpada e que possa ser trocada para orientar o funcionamento da propagação de luz no ambiente.

Respostas – página 91

1)

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

const int n_pts_controle = 8;

GLfloat pontos_controle[n_pts_controle][3] = {
   {1.0, 1.0,  0.0},
   {0.0, 4.0,  0.0},
   {2.0, 6.0,  0.0},
   {4.0, 8.0,  0.0},
   {6.0, 7.0,  0.0},
   {8.0, 5.0,  0.0},
   {7.0, 0.0,  0.0},
   {7.0, 1.0,  0.0}
};

int numerosegm = 60;

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0,50,0,24,0,1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,1,1);
    glLineWidth(2.0);

    glEvalMesh1(GL_LINE, 0, numerosegm);

    glutSwapBuffers();
}

static void idle(void)
{
    glutPostRedisplay();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
```

```
    glutCreateWindow("Janela Principal");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);
    glutIdleFunc(idle);

    glMap1f(GL_MAP1_VERTEX_3,
            0.0, 1.0,
            3,
            n_pts_controle,
            &pontos_controle[0][0]);
    glEnable(GL_MAP1_VERTEX_3);

    glMapGrid1d(numerosegm, 0.0, 1.0);

    glutMainLoop();
}
```

2)

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

const int n_pts_controle = 4;

GLfloat pontos_controle[n_pts_controle][3] = {
   {-2.0, -1.0,  0.0},
   {-1.0,  3.0,  0.0},
   { 1.0, -3.0,  0.0},
   { 2.0,  1.0,  0.0}
};

int numerosegm = 20;

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-5,5,-5,5,0,1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,1,1);
    glLineWidth(2.0);

    glEvalMesh1(GL_LINE, 0, numerosegm);

    glutSwapBuffers();
}
```

```
static void idle(void)
{
    glutPostRedisplay();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Janela Principal");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);
    glutIdleFunc(idle);

    // Necessário para a curva de Bézier
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3,
            n_pts_controle, &pontos_controle[0][0]);
    glEnable(GL_MAP1_VERTEX_3);
    glMapGrid1d(numerosegm, 0.0, 1.0);

    glutMainLoop();
}
```

3)

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

GLfloat ctlpoints[4][4][3];
int showPoints = 0;

GLUnurbsObj *theNurb;

void init_surface(void)
{
   int u, v;
   for (u = 0; u < 4; u++) {
      for (v = 0; v < 4; v++) {
         ctlpoints[u][v][0] = 2.0*((GLfloat)u - 1.5);
         ctlpoints[u][v][1] = 2.0*((GLfloat)v - 1.5);

         if ( (u == 1 || u == 2) && (v == 1 || v == 2))
            ctlpoints[u][v][2] = 3.0;
         else
            ctlpoints[u][v][2] = -3.0;
      }
   }
}

void display(void)
```

```
{
   GLfloat knots[8] = {0.0, 0.0, 1.0, 1.0, 1.5, 1.5, 2., 2.};
   int i, j;

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glPushMatrix();
   glTranslatef (0.0, 0.0, -5.0);
   glRotatef(330.0, 0.,1.,0.);
   glScalef (0.5, 0.5, 0.5);

   gluBeginSurface(theNurb);
   gluNurbsSurface(theNurb,
                   8, knots, 8, knots,
                   4 * 3, 3, &ctlpoints[0][0][0],
                   4, 4, GL_MAP2_VERTEX_3);
   gluEndSurface(theNurb);

   glPopMatrix();
   glutSwapBuffers();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective (45.0, (GLdouble)w/(GLdouble)h, 0.0, 8.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();

}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize (800,600);
   glutInitWindowPosition (100, 100);
   glutCreateWindow("Janela Principal");

   glutReshapeFunc(reshape);
   glutDisplayFunc(display);

   glClearColor (0.0, 0.0, 0.0, 0.0);
   init_surface();
   theNurb = gluNewNurbsRenderer();

   glutMainLoop();
}
```

Capítulo 7

Orientações

A criação de um objeto sólido a partir de primitivas geométricas possibilita uma visão mais ampla do que se pode construir. Por exemplo, no caso da construção de um veículo, ou mesmo da cabeça de uma pessoa, o ideal é fazê-los com elementos triangulares ou quadráticos, pois estes permitem um melhor arranjo de seus pontos para construção do objeto desejado.

Porém, sempre parte-se de uma pequena parte, ou de um objeto geométrico simples, e aumenta-se gradativamente até obter um objeto mais complexo. No exemplo do silo, o qual faz parte da resolução no livro, o ideal é utilizar as primitivas geométricas como triângulo e quadrilátero, pois com esses dois elementos podemos criar o teto (triângulos) e as paredes (quadriláteros), e, dessa forma, temos um objeto tridimensional.

Respostas – página 103

1)

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

using namespace std;

static void resize(int width, int height)
{
    const float ar = (float) width / (float) height;

    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(90, width/height, 0.0, 100.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity() ;
}

void montaLateral(int nquads, float x, float y, float z){
        glBegin(GL_QUAD_STRIP);
            glVertex3f( x, y, z);
            glVertex3f( x, y+1, z);

            for(int i = 0; i <= nquads; i++) {
                x++;
                glVertex3f(x, y, z);
                glVertex3f(x, y+1, z);

            }
        glEnd();
}
```

```
void montaLat_ED(int nquads, float x, float y, float z){
        glBegin(GL_QUAD_STRIP);
            glVertex3f( x, y, z); // lado inferior esquerdo
            glVertex3f( x, y+1, z); // lado superior esquerdo

            for(int i = 0; i < nquads; i++) {
                z--;
                glVertex3f(x, y, z); // lado inferior direito
                glVertex3f(x, y+1, z); // lado superior direito

            }
        glEnd();
}

void montaBase(int nquads, float x, float y, float z){
        glBegin(GL_QUAD_STRIP);
            float z_aux = z;
            for (int j = 0; j <= nquads; j++){
                glVertex3f( x, y, z); // lado inferior esquerdo
                glVertex3f( x+1, y, z); // lado superior esquer-
do
                for(int i = 0; i <= nquads; i++) {
                    glVertex3f(x, y, z); // lado inferior di-
reito
                    glVertex3f(x+1, y, z); // lado superior di-
reito
                    z--;
                }
                z = z_aux;
                x++;
            }
        glEnd();
}

void montaPeMesa(float x, float y, float z){
        glBegin(GL_QUAD_STRIP);
            for (int j = 0; j <= 10; j++){
                glVertex3f( x, y, z);
                glVertex3f( x, y-1, z);

                glVertex3f(x+1, y, z);
                glVertex3f(x+1,y-1, z);

                glVertex3f(x+1, y,z-1);
                glVertex3f(x+1,y-1,z-1);

                glVertex3f(x, y, z-1);
                glVertex3f(x, y-1, z-1);

                glVertex3f( x, y, z);
                glVertex3f( x, y-1, z);

                y--;
            }
        glEnd();
}

static void display(void)
{
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,1,1);

    int quads = 10; // numero de quadrados

    gluLookAt(0,0,10,0,0,0,0,1,0);
    //glTranslated(0,0,-4);
    glRotatef(45,1,1,0);
    //  monta base da mesa
    glPushMatrix();

        glColor3f(0.2,0.2,0.8);
        montaLateral(quads, 0, 0, -3); // monta base frontal da
mesa
        glColor3f(0.3,0.3,0.8);
        montaLateral(quads, 0, 0, -13); // monta base traseira
da mesa
        glColor3f(0.3,0.4,0.8);
        montaLat_ED(quads, 0, 0, -3); // monta base lateral esq
da mesa
        glColor3f(0.3,0.6,0.8);
        montaLat_ED(quads, quads+1, 0, -3); // monta base lat-
eral dir da mesa
        glColor3f(0.6,0.6,0.8);
        montaBase(quads, 0, 0, -3); // tampa inferior da mesa
        glColor3f(0.9,0.6,0.8);
        montaBase(quads, 0, 1, -3); // tampa inferior da mesa
        // cria os pés da mesa
        montaPeMesa(0,0,-3);
        montaPeMesa(quads,0,-3);
        montaPeMesa(quads,0,-12);
        montaPeMesa(0,0,-12);
    glPopMatrix();

    glutSwapBuffers();
}


static void key(unsigned char key, int x, int y)
{
    switch (key)
    {
        case 27 :
        case 'q':
            exit(0);
            break;

    }

    glutPostRedisplay();
}

static void idle(void)
{
    glutPostRedisplay();
}

const GLfloat light_ambient[]  = { 0.0f, 0.0f, 0.0f, 1.0f };
const GLfloat light_diffuse[]  = { 1.0f, 1.0f, 1.0f, 1.0f };
```

```
const GLfloat light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat light_position[] = { 2.0f, 5.0f, 5.0f, 0.0f };

const GLfloat mat_ambient[]    = { 0.7f, 0.7f, 0.7f, 1.0f };
const GLfloat mat_diffuse[]    = { 0.8f, 0.8f, 0.8f, 1.0f };
const GLfloat mat_specular[]   = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat high_shininess[] = { 100.0f };

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("GLUT Shapes");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);
//    glutKeyboardFunc(key);
//    glutIdleFunc(idle);

    glClearColor(0,0,0,1);
//    glEnable(GL_CULL_FACE);
//    glCullFace(GL_BACK);
//
//    glEnable(GL_DEPTH_TEST);
//    glDepthFunc(GL_LESS);

    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);

    glLightfv(GL_LIGHT0, GL_AMBIENT,  light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE,  light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    glMaterialfv(GL_FRONT, GL_AMBIENT,   mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE,   mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR,  mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);

    glutMainLoop();

    return EXIT_SUCCESS;
```

2)

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

using namespace std;

static void resize(int width, int height)
{
    const float ar = (float) width / (float) height;

    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(90, width/height, 0.0, 200.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity() ;
}

void paredeCasa(float x1, float x2, float y1, float y2, float prox,
float longe){
        glBegin(GL_QUADS);
                glVertex3f( x1, y1, prox);
                glVertex3f( x1, y2, prox);
                glVertex3f( x2, y2,longe);
                glVertex3f( x2, y1,longe);
        glEnd();
}

void paredeSuperiorTriangular(float topox, float topoy, float prof,
float x, float y){
    glBegin(GL_TRIANGLE_FAN);
        glVertex3f(topox,topoy,prof);
        glVertex3f(x,y,prof);
        glVertex3f(x+2,y,prof);
        glVertex3f(x+4,y,prof);
    glEnd();
}

void telhado(float x1, float x2, float y1, float y2, float prox, float
longe){
    glBegin(GL_QUADS);
        glVertex3f( x1, y1, prox);
        glVertex3f( x1, y1, longe);
        glVertex3f( x2, y2,longe);
        glVertex3f( x2, y2,prox);
    glEnd();
}
static void display(void)
{
```

```
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3d(1,1,1);

        int quads = 10; // numero de quadrados

        //gluLookAt(10,2,8,0,0,0,1,1,0);
        glTranslated(0,0,-10);
        glRotatef(180,0,1,0);
        //  monta base da mesa
        glPushMatrix();
            glColor3f(0.2,0.2,0.8);

            paredeCasa(0,4,0,4,-1,-1); // frente da casa
            glPushMatrix();
                // frente superior triangular frente
                paredeSuperiorTriangular(2,6,-1,0,4);
            glPopMatrix();

            glColor3f(0.1,0.9,0.8);
            paredeCasa(0,4,0,4,-4,-4); // fundos da casa
            glPushMatrix();
                // frente superior triangular fundos
                paredeSuperiorTriangular(2,6,-4,0,4);
            glPopMatrix();

            glColor3f(0.2,0.2,0.1);
            paredeCasa(0,0,0,4,-1,-4); // lateral esquerda da casa
            paredeCasa(4,4,0,4,-1,-4); // lateral direita da casa

            glColor3f(0.6,0.6,0.6);
            telhado(0.,2.,4.,6.,-1,-4);
            telhado(2.,4.,6.,4.,-1,-4);
        glPopMatrix();

        glutSwapBuffers();
    }

    int main(int argc, char *argv[])
    {
        glutInit(&argc, argv);
        glutInitWindowSize(800,600);
        glutInitWindowPosition(10,10);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

        glutCreateWindow("GLUT Shapes");

        glutReshapeFunc(resize);
        glutDisplayFunc(display);

        glClearColor(0,0,0,1);

        glutMainLoop();

        return EXIT_SUCCESS;
    }
```

Capítulo 8

Orientações

A textura permite diminuir a quantidade de polígonos em um objeto, o que reduz drasticamente o tempo computacional. Para tanto, é possível utilizar qualquer tipo de imagem, contudo, no livro foi utilizado o algoritmo para abertura e leitura de arquivos .bmp que são mais fáceis, porém nada impede o docente de buscar outras bibliotecas para manipulação de arquivos. Isso irá permitir o uso de outros formatos de imagem, como .tiff, .jpeg e .png.

Foi utilizada a criação de uma janela (no livro), usando um conjunto de primitivas geométricas, pode-se, entretanto, criar um único objeto retangular e então aplicar a textura de uma janela, mas isso fica como exemplo para que o docente possa utilizar como uma tarefa adicional.

Respostas – página 115

1)

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

float _angle = -70.0f;

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective (45.0, (GLdouble) width/(GLdouble)height,
1.0, 200.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

static void initLight(){
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
}

static void display(void)
{
    initLight();


 GLfloat ambientColor[] = {0.2f, 0.2f, 0.2f, 1.0f};
 glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientColor);

 GLfloat light[] = {0., 1., 0., 0.};
 glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, light);
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glPushMatrix();
        glTranslated(0,0,-10);
        glScalef(0.5,0.5,0.5);
        glutSolidSphere(5,25,25);
    glPopMatrix();

    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(150,150);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Janela Principal");

    glClearColor(1.0, 1.0, 1.0, 0.0);
    glutReshapeFunc(resize);
    glutDisplayFunc(display);

    glutMainLoop();
}
```

2)

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

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective (45.0, (GLdouble) width/(GLdouble)height,
1.0, 200.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

static void initLight(){
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
}

static void display(void)
{
    initLight();
```

```
GLfloat ambientColor[] = {0.2f, 0.2f, 0.2f, 1.0f};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientColor);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    GLfloat light[] = {0., 0., 0., 0.};;

    glPushMatrix();
        light = {0., 0., 1., 0.};
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, light);
        glTranslated(0,0,-20);
        glScalef(0.5,0.5,0.5);
        glutSolidSphere(5,25,25);
    glPopMatrix();

    glPushMatrix ();
        light = {1., 1., 0., 0.};
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, light);
        glTranslated(-5,0,-20);
        glScalef(0.5,0.5,0.5);
        glutSolidSphere(5,25,25);
    glPopMatrix();

    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(150,150);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Janela Principal");

    glClearColor(1.0, 1.0, 1.0, 0.0);
    glutReshapeFunc(resize);
    glutDisplayFunc(display);

    glutMainLoop();
}
```

3)

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

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective (45.0, (GLdouble) width/(GLdouble)height,
1.0, 200.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

```
static void initLight(){
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
}

static void display(void)
{
    initLight();

 GLfloat ambientColor[] = {0.2f, 0.2f, 0.2f, 1.0f};
 glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientColor);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    GLfloat light[] = {0., 0., 0., 0.};;

    glPushMatrix();
        light = {0., 0., 1., 0.};
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, light);
        glTranslated(0,0,-20);
        glScalef(0.5,0.5,0.5);
        glutSolidSphere(5,25,25);
    glPopMatrix();

    glPushMatrix ();
        light = {1., 1., 0., 0.};
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, light);
        glTranslated(-5,0,-20);
        glScalef(0.5,0.5,0.5);
        glutSolidSphere(5,25,25);
    glPopMatrix();

    glPushMatrix ();
        light = {.8, .5, 0.2, 1.};
        glMaterialfv(GL_FRONT, GL_SPECULAR, light);
        GLfloat lightPos[] = {-6.0, 1.0, 4.0, .0};
        glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
        glTranslated(5,0,-20);
        glScalef(0.5,0.5,0.5);
        glutSolidSphere(5,25,25);
    glPopMatrix();

    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
```

```
        glutInitWindowPosition(150,150);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

        glutCreateWindow("Janela Principal");

        glClearColor(1.0, 1.0, 1.0, 0.0);
        glutReshapeFunc(resize);
        glutDisplayFunc(display);

        glutMainLoop();
    }
```

Capítulo 9

Orientações

A renderização permite incorporar outros elementos como a transparência. A transparência possibilita criar novos contrastes entre objetos, e até mesmo permitir que outros objetos de nosso cotidiano sejam criados, por exemplo, o vidro e o espelho.

O vidro é um elemento mais simples de descrever e apresentar ao aluno, pois já estão bem acostumados com tal objeto. Agora o espelho não há como fazê-lo diretamente e para isso é importante ressaltar que são necessários dois objetos iguais, desenhados em lados opostos, justamente para caracterizar a reflexão no espelho.

Respostas – página 131

1)

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

GLuint  texture_id[1];
int posicaoluz = 0;
float ang = 0.0;

int  LoadBMP(char *filename)
 {
    #define SAIR        {fclose(fp_arquivo); return -1;}
    #define CTOI(C)     (*(int*)&C)

    GLubyte     *image;
    GLubyte     Header[0x54];
    GLuint      DataPos, imageSize;
    GLsizei     Width,Height;

    int nb = 0;
```

```
        FILE * fp_arquivo = fopen(filename,"rb");
        if (!fp_arquivo){
            std::cout << "erro de leitura!!!";
            return -1;
        }
        if (fread(Header,1,0x36,fp_arquivo)!=0x36){
            std::cout << "erro de leitura!!!";
            SAIR;
        }
        if (Header[0]!='B' || Header[1]!='M'){
            std::cout << "erro de leitura!!!";
            SAIR;
        }
        if (CTOI(Header[0x1E])!=0){
            std::cout << "erro de leitura!!!";
            SAIR;
        }
        if (CTOI(Header[0x1C])!=24){
            std::cout << "erro de leitura!!!";
            SAIR;
        }

        // Recupera a informação dos atributos de
        // altura e largura da imagem

        Width   = CTOI(Header[0x12]);
        Height  = CTOI(Header[0x16]);
        ( CTOI(Header[0x0A]) == 0 ) ? ( DataPos=0x36 ) : ( DataPos =
    CTOI(Header[0x0A]) );

        imageSize=Width*Height*3;

        // Efetura a Carga da Imagem
        image = (GLubyte *) malloc ( imageSize );
        int retorno;
        retorno = fread(image,1,imageSize,fp_arquivo);

        if (retorno !=imageSize)
         {
            free (image);
            SAIR;
         }

        // Inverte os valores de R e B
        int t, i;

        for ( i = 0; i < imageSize; i += 3 )
         {
```

```
            t = image[i];
            image[i] = image[i+2];
            image[i+2] = t;
         }

        // Tratamento da textura para o OpenGL

        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LIN-
    EAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LIN-
    EAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);

        glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE
    );
        // Faz a geraçao da textura na memória
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, Width, Height, 0, GL_
    RGB, GL_UNSIGNED_BYTE, image);

        fclose (fp_arquivo);
        free (image);
        return 1;
     }

    static void resize(int width, int height)
    {
        glViewport(0, 0, width, height);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(90,width / height,1, 1000);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity() ;
    }

    void desenhaCubo(float tam){
           glBegin(GL_QUADS);
                // face frontal
                glColor3f(1.,1.,1.);
                glTexCoord2f(0.,0.);glVertex3f(0.0,0.0,0.0);
                glTexCoord2f(1.,0.);glVertex3f(tam,0.0,0.0);
                glTexCoord2f(1.,1.);glVertex3f(tam,tam,0.0);
                glTexCoord2f(0.,1.);glVertex3f(0.0,tam,0.0);
                // face lateral direita
                glColor3f(1.,1.,1.);
                glTexCoord2f(0.,0.);glVertex3f(tam,0.0,0.0);
                glTexCoord2f(1.,0.);glVertex3f(tam,0.0,tam);
                glTexCoord2f(1.,1.);glVertex3f(tam,tam,tam);
                glTexCoord2f(0.,1.);glVertex3f(tam,tam,0.0);
```

```
            // face lateral esquerda
            glColor3f(1.0,0.6,0.6);
            glTexCoord2f(0.,0.);glVertex3f(0.0,0.0,0.0);
            glTexCoord2f(1.,0.);glVertex3f(0.0,0.0,tam);
            glTexCoord2f(1.,1.);glVertex3f(0.0,tam,tam);
            glTexCoord2f(0.,1.);glVertex3f(0.0,tam,0.0);
            // face inferior
            glColor3f(0.4,0.4,0.0);
            glTexCoord2f(0.,0.);glVertex3f(0.0,0.0,0.0);
            glTexCoord2f(1.,0.);glVertex3f(tam,0.0,0.0);
            glTexCoord2f(1.,1.);glVertex3f(tam,0.0,tam);
            glTexCoord2f(0.,1.);glVertex3f(0.0,0.0,tam);
            // face supeiror
            glColor3f(0.9,0.9,0.9);
            glTexCoord2f(0.,0.);glVertex3f(0.0,tam,0.0);
            glTexCoord2f(1.,0.);glVertex3f(tam,tam,0.0);
            glTexCoord2f(1.,1.);glVertex3f(tam,tam,tam);
            glTexCoord2f(0.,1.);glVertex3f(0.0,tam,tam);
            // face fundo
            glColor3f(1.0,0,1.);
            glTexCoord2f(0.,0.);glVertex3f(0.0,0.0,tam);
            glTexCoord2f(1.,0.);glVertex3f(tam,0.0,tam);
            glTexCoord2f(1.,1.);glVertex3f(tam,tam,tam);
            glTexCoord2f(0.,1.);glVertex3f(0.0,tam,tam);
        glEnd();
}

static void display(void)
{

    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);

    glEnable ( GL_TEXTURE_2D );
    glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
    glGenTextures ( 1, texture_id );
    glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
    LoadBMP ("F://Editora_Livro_Comp_Grafica//imagens//aula9_3.
bmp");

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);

    //gluLookAt(-10,10,15,0,0,0,1,1,0);

    GLfloat semespecular[4]={0.0,0.0,0.0,1.0};
    GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };
```

```
        GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };

        glPushMatrix () ;
            glRotated ((GLdouble) posicaoluz, 1.0, 0.0, 0.0);
            glLightfv (GL_LIGHT0, GL_POSITION, posicao);
        glPopMatrix();

       glPushMatrix();

            glMaterialfv(GL_FRONT,GL_SPECULAR, semespecular);
            glMateriali(GL_FRONT,GL_SHININESS,20);
            glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
            glTranslatef(-2.5,-2.5,-10);
            glRotated(ang,1,1,0);
            desenhaCubo(5);
        glPopMatrix();
        glutSwapBuffers();
    }


    static void key(unsigned char key, int x, int y)
    {
        switch (key)
        {
            case 27 :
            case 'q':
                exit(0);
                break;
        }

        glutPostRedisplay();
    }

    void idle(int value)
    {
        ang += 10.0;
        glutPostRedisplay();
        glutTimerFunc(200,idle,1);
    }

    int main(int argc, char *argv[])
    {
        glutInit(&argc, argv);
        glutInitWindowSize(640,480);
        glutInitWindowPosition(10,10);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

        glutCreateWindow("Cubo com Textura");

        glutReshapeFunc(resize);
```

```
        glutDisplayFunc(display);
        glutKeyboardFunc(key);
        glutTimerFunc(200,idle,1);

        glClearColor(0,0,0,0);

        glutMainLoop();

        return EXIT_SUCCESS;
    }
```

2)

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

GLuint  texture_id[5];
int posicaoluz = 0;
float ang = 0.0;

int  LoadBMP(char *filename)
 {
    #define SAIR        {fclose(fp_arquivo); return -1;}
    #define CTOI(C)     (*(int*)&C)

    GLubyte     *image;
    GLubyte     Header[0x54];
    GLuint      DataPos, imageSize;
    GLsizei     Width,Height;

    int nb = 0;
    FILE * fp_arquivo = fopen(filename,"rb");
    if (!fp_arquivo){
        std::cout << "erro de leitura!!!";
        return -1;
    }
    if (fread(Header,1,0x36,fp_arquivo)!=0x36){
        std::cout << "erro de leitura!!!";
        SAIR;
    }
    if (Header[0]!='B' || Header[1]!='M'){
        std::cout << "erro de leitura!!!";
        SAIR;
    }
    if (CTOI(Header[0x1E])!=0){
        std::cout << "erro de leitura!!!";
        SAIR;
    }
```

```
    if (CTOI(Header[0x1C])!=24){
        std::cout << "erro de leitura!!!";
        SAIR;
    }

    // Recupera a informação dos atributos de
    // altura e largura da imagem

    Width   = CTOI(Header[0x12]);
    Height  = CTOI(Header[0x16]);
    ( CTOI(Header[0x0A]) == 0 ) ? ( DataPos=0x36 ) : ( DataPos =
CTOI(Header[0x0A]) );

    imageSize=Width*Height*3;

    // Efetura a Carga da Imagem
    image = (GLubyte *) malloc ( imageSize );
    int retorno;
    retorno = fread(image,1,imageSize,fp_arquivo);

    if (retorno !=imageSize)
     {
        free (image);
        SAIR;
     }

    // Inverte os valores de R e B
    int t, i;

    for ( i = 0; i < imageSize; i += 3 )
     {
        t = image[i];
        image[i] = image[i+2];
        image[i+2] = t;
     }

    // Tratamento da textura para o OpenGL

    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LIN-
EAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LIN-
EAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);

    glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE
);
    // Faz a geraçao da textura na memória
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, Width, Height, 0, GL_
RGB, GL_UNSIGNED_BYTE, image);
```

```
    fclose (fp_arquivo);
    free (image);
    return 1;
 }

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(90,width / height,1, 1000);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity() ;
}

void desenhaCubo(float tam){

       glColor3f(1.,1.,1.);
       LoadBMP ("F://Editora_Livro_Comp_Grafica//imagens//
aula9_3.bmp");
       glBegin(GL_QUADS);
            // face frontal
            glTexCoord2f(0.,0.);glVertex3f(0.0,0.0,0.0);
            glTexCoord2f(1.,0.);glVertex3f(tam,0.0,0.0);
            glTexCoord2f(1.,1.);glVertex3f(tam,tam,0.0);
            glTexCoord2f(0.,1.);glVertex3f(0.0,tam,0.0);
        glEnd();
        LoadBMP ("F://Editora_Livro_Comp_Grafica//imagens//tex-
tura_ladrilho.bmp");
        glBegin(GL_QUADS);
            // face lateral direita
            glTexCoord2f(0.,0.);glVertex3f(tam,0.0,0.0);
            glTexCoord2f(1.,0.);glVertex3f(tam,0.0,tam);
            glTexCoord2f(1.,1.);glVertex3f(tam,tam,tam);
            glTexCoord2f(0.,1.);glVertex3f(tam,tam,0.0);
        glEnd();
        LoadBMP ("F://Editora_Livro_Comp_Grafica//imagens//tex-
tura_ladrilho1.bmp");
        glBegin(GL_QUADS);
            // face lateral esquerda
            glTexCoord2f(0.,0.);glVertex3f(0.0,0.0,0.0);
            glTexCoord2f(1.,0.);glVertex3f(0.0,0.0,tam);
            glTexCoord2f(1.,1.);glVertex3f(0.0,tam,tam);
            glTexCoord2f(0.,1.);glVertex3f(0.0,tam,0.0);
        glEnd();
        LoadBMP ("F://Editora_Livro_Comp_Grafica//imagens//tex-
tura_lajota.bmp");
        glBegin(GL_QUADS);
            // face inferior
```

```
            glTexCoord2f(0.,0.);glVertex3f(0.0,0.0,0.0);
            glTexCoord2f(1.,0.);glVertex3f(tam,0.0,0.0);
            glTexCoord2f(1.,1.);glVertex3f(tam,0.0,tam);
            glTexCoord2f(0.,1.);glVertex3f(0.0,0.0,tam);
        glEnd();
        LoadBMP ("F://Editora_Livro_Comp_Grafica//imagens//tex-
tura_tijolo.bmp");
        glBegin(GL_QUADS);
            // face supeiror
            glTexCoord2f(0.,0.);glVertex3f(0.0,tam,0.0);
            glTexCoord2f(1.,0.);glVertex3f(tam,tam,0.0);
            glTexCoord2f(1.,1.);glVertex3f(tam,tam,tam);
            glTexCoord2f(0.,1.);glVertex3f(0.0,tam,tam);
            // face fundo
            glTexCoord2f(0.,0.);glVertex3f(0.0,0.0,tam);
            glTexCoord2f(1.,0.);glVertex3f(tam,0.0,tam);
            glTexCoord2f(1.,1.);glVertex3f(tam,tam,tam);
            glTexCoord2f(0.,1.);glVertex3f(0.0,tam,tam);
        glEnd();
}

static void display(void)
{

    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);

    glEnable ( GL_TEXTURE_2D );


    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);

    //gluLookAt(-10,10,15,0,0,0,1,1,0);

    GLfloat semespecular[4]={0.0,0.0,0.0,1.0};
    GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };

    GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };

    glPushMatrix () ;
        glRotated ((GLdouble) posicaoluz, 1.0, 0.0, 0.0);
        glLightfv (GL_LIGHT0, GL_POSITION, posicao);
    glPopMatrix();

    glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
    glGenTextures ( 1, texture_id );
    glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
```

```
        glPushMatrix();

            glMaterialfv(GL_FRONT,GL_SPECULAR, semespecular);
            glMateriali(GL_FRONT,GL_SHININESS,20);
            glTranslatef(-2.5,-2.5,-10);
            glRotated(ang,1,1,0);
            desenhaCubo(5);
        glPopMatrix();
        glutSwapBuffers();
    }


    static void key(unsigned char key, int x, int y)
    {
        switch (key)
        {
            case 27 :
            case 'q':
                exit(0);
                break;
        }

        glutPostRedisplay();
    }

    void idle(int value)
    {
        ang += 10.0;
        glutPostRedisplay();
        glutTimerFunc(200,idle,1);
    }

    int main(int argc, char *argv[])
    {
        glutInit(&argc, argv);
        glutInitWindowSize(640,480);
        glutInitWindowPosition(10,10);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

        glutCreateWindow("Cubo com Textura");

        glutReshapeFunc(resize);
        glutDisplayFunc(display);
        glutKeyboardFunc(key);
        glutTimerFunc(200,idle,1);

        glClearColor(0,0,0,0);

        glutMainLoop();

        return EXIT_SUCCESS;
    }
```

Capítulo 10

Orientações

A animação pode ser apresentada pelo docente usando uma filmadora digital. Nesse caso, filma-se uma bola em movimento e depois mostra o arquivo quadro a quadro da filmagem. A ideia é poder separar os quadros para mostrar que o movimento ocorre com a junção de todos eles (quadros) e o mesmo deverá ser feito pelos alunos na programação.

Respostas – página 143

1)

```
#include <GL/glut.h>
#include <string>
#include <stdlib.h>
#include <iostream>
#include <stdio.h>

GLuint texture_id[2];

static void resize(int width, int height)
{
    const float ar = (float) width / (float) height;

    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60,width/height,1.,100.);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity() ;
}

int  LoadBMP(const char *filename)  {
    #define SAIR        {fclose(fp_arquivo); return -1;}
    #define CTOI(C)     (*(int*)&C)

    GLubyte     *image;
    GLubyte     Header[0x54];
    GLuint      DataPos, imageSize;
    GLsizei     Width,Height;

    int nb = 0;

    FILE * fp_arquivo = fopen(filename,"rb");
    if (!fp_arquivo){
        std::cout << "erro de leitura!!!+\n";
        return -1;
    }
    if (fread(Header,1,0x36,fp_arquivo)!=0x36){
        std::cout << "erro de leitura!!!-\n";
        SAIR;
    }
```

```
        if (Header[0]!='B' || Header[1]!='M'){
            std::cout << "erro de leitura!!!*\n";
            SAIR;
        }
        if (CTOI(Header[0x1E])!=0){
            std::cout << "erro de leitura!!!/\n";
            SAIR;
        }
        if (CTOI(Header[0x1C])!=24){
            std::cout << "erro de leitura!!!\\\n";
            SAIR;
        }

        // Recupera a informação dos atributos de
        // altura e largura da imagem
        Width   = CTOI(Header[0x12]);
        Height  = CTOI(Header[0x16]);
        ( CTOI(Header[0x0A]) == 0 ) ? ( DataPos=0x36 ) : ( DataPos =
    CTOI(Header[0x0A]) );

        imageSize=Width*Height*3;

        // Efetura a Carga da Imagem
        image = (GLubyte *) malloc ( imageSize );
        int retorno;
        retorno = fread(image,1,imageSize,fp_arquivo);

        if (retorno !=imageSize)
         {
            free (image);
            SAIR;
         }

        // Inverte os valores de R e B
        int t, i;

        for ( i = 0; i < imageSize; i += 3 )
         {
            t = image[i];
            image[i] = image[i+2];
            image[i+2] = t;
         }

        // Tratamento da textura para o OpenGL

        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LIN-
    EAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LIN-
    EAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
```

```
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);

        glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE
    );
        // Faz a geracao da textura na memoria
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, Width, Height, 0, GL_
    RGB, GL_UNSIGNED_BYTE, image);

        fclose (fp_arquivo);
        free (image);
        return 1;
     }


    static void initLight(){
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_COLOR_MATERIAL);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_NORMALIZE);
    }


    static void loadLight() {
        GLfloat ambientColor[] = {0.5, 0.5, 0.5, 1};
        // determina o modelo de luz a ser utilizada
        glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientColor);
        glColorMaterial ( GL_FRONT, GL_AMBIENT_AND_DIFFUSE );

        glPushMatrix();
            glRotated(90,0,1,0);
            // determina o tipo de luz a ser utilizada
            GLfloat diffuseLight[] = { 0.7f, 0.7f, 0.4f, 1.0f };
            glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
            // aplica a luz na posicao desejada
            GLfloat lightPos[] = {0.0f, 0.0f, -3.0f, 1.0f};
            glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
        glPopMatrix();
    }

    void desenhaParedes(float i) {

        /// Paredes
        // Aplica textra as paredes
        glBindTexture ( GL_TEXTURE_2D, texture_id[1] );
        // parede lateral
        glBegin(GL_POLYGON);
            glTexCoord2d(1,1);glVertex3d(i,0,0);
            glTexCoord2d(1,0);glVertex3d(i,i,0);
            glTexCoord2d(0,0);glVertex3d(0,i,0);
```

```
            glTexCoord2d(0,1);glVertex3d(0,0,0);
        glEnd();
        // parede de fundo com recorte para a janela
        glBegin(GL_QUADS);
            glColor4d(1,1,1,1);
            glTexCoord2d(0,1);glVertex3d(0,i,0);
            glTexCoord2d(0.66,1);glVertex3d(0,i,0.66*i);
            glTexCoord2d(0.66,0);glVertex3d(0,0,0.66*i);
            glTexCoord2d(0,0);glVertex3d(0,0,0);

            glTexCoord2d(0,1);glVertex3d(0,i,i*2);
            glTexCoord2d(0.33,1);glVertex3d(0,i,i*2-(0.66*i));
            glTexCoord2d(0.33,0);glVertex3d(0,0,i*2-(0.66*i));
            glTexCoord2d(0,0);glVertex3d(0,0,i*2);

            glTexCoord2d(0.33,1);glVertex3d(0,i     ,i*2-(0.66*i));
            glTexCoord2d(0.66,1);glVertex3d(0,i     ,0.66*i);
            glTexCoord2d(0.66,0.66);glVertex3d(0,0.66*i,0.66*i);
            glTexCoord2d(0.33,0.66);glVertex3d(0,0.66*i,i*2-
    (0.66*i));

            glTexCoord2d(0.33,0);glVertex3d(0,0       ,i*2-
    (0.66*i));
            glTexCoord2d(0.66,0);glVertex3d(0,0       ,0.66*i);
            glTexCoord2d(0.66,0.33);glVertex3d(0,i-0.66*i,0.66*i);
            glTexCoord2d(0.33,0.33);glVertex3d(0,i-0.66*i,i*2-
    (0.66*i));

        glEnd();
        // parede lateral
        glColor3d(1,1,1);
        glBegin(GL_POLYGON);
            glTexCoord2d(1,1);glVertex3d(i,0,i*2);
            glTexCoord2d(1,0);glVertex3d(i,i,i*2);
            glTexCoord2d(0,0);glVertex3d(0,i,i*2);
            glTexCoord2d(0,1);glVertex3d(0,0,i*2);
        glEnd();
    }

    void desenhaChao(float tam){
        /// Chão
        // aplica textura no chao
        glBindTexture ( GL_TEXTURE_2D, texture_id[0]);
        // cria o chao
        glBegin(GL_QUADS);
            glTexCoord2d(1,1);glVertex3d(0,0,tam*2);
            glTexCoord2d(1,0);glVertex3d(tam,0,tam*2);
            glTexCoord2d(0,0);glVertex3d(tam,0,0);
            glTexCoord2d(0,1);glVertex3d(0,0,0);
```

```
        glEnd();
    }

    void desenhaJanela(float tam){
        glBegin(GL_QUADS);
            // cor com transparencia
            glColor4d(0.7,0.7,1,0.5);
            // desenhando a janela no centro da parede
            glVertex3d(0,tam*0.66  ,tam*2-(0.66*tam));
            glVertex3d(0,tam*0.66  ,0.66*tam);
            glVertex3d(0,tam-0.66*tam,0.66*tam);
            glVertex3d(0,tam-0.66*tam,tam*2-(0.66*tam));
        glEnd();
    }

    static void display(void)
    {
        initLight();
        loadLight();

        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3d(1,1,1);

        //gluLookAt(10,5,0,0,0,0,0,1,0);

        glPushMatrix();
            glTranslated(0,.35f,-5);
            glutSolidCube(0.3);
        glPopMatrix();
        glPushMatrix();
            glTranslated(0,-0.40,-4);
            glRotated(15,1,0,0);
            glRotated(-90,0,1,0);

            glPushMatrix();
                glTranslated(0,0,-1.5);
                glColor3d(1,1,1);
                desenhaParedes(1.5);
                desenhaChao(1.5);
                desenhaJanela(1.5);
            glPopMatrix();
        glPopMatrix();

        glutSwapBuffers();
    }

    int main(int argc, char *argv[])
```

```
{
    glutInit(&argc, argv);
    glutInitWindowSize(640,480);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Sala");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);

    glClearColor(1,1,1,0.3);
    glCullFace(GL_BACK);

    glEnable ( GL_TEXTURE_2D );
    glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
    glGenTextures ( 1, texture_id );

    glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
    LoadBMP ("F:\\lixo\\sala3d_w\\piso.bmp");

    glBindTexture ( GL_TEXTURE_2D, texture_id[1] );
    LoadBMP ("F:\\lixo\\sala3d_w\\parede.bmp");

    glutMainLoop();

    return EXIT_SUCCESS;
}
```

2)

```
#include <GL/glut.h>
#include <string>
#include <stdlib.h>
#include <iostream>
#include <stdio.h>

GLuint texture_id[2];

static void resize(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60,width/height,1.,100.);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity() ;
}

int  LoadBMP(const char *filename)  {
    #define SAIR        {fclose(fp_arquivo); return -1;}
    #define CTOI(C)     (*(int*)&C)
```

```
        GLubyte     *image;
        GLubyte     Header[0x54];
        GLuint      DataPos, imageSize;
        GLsizei     Width,Height;

        int nb = 0;

        FILE * fp_arquivo = fopen(filename,"rb");
        if (!fp_arquivo){
            std::cout << "erro de leitura!!!+\n";
            return -1;
        }
        if (fread(Header,1,0x36,fp_arquivo)!=0x36){
            std::cout << "erro de leitura!!!-\n";
            SAIR;
        }
        if (Header[0]!='B' || Header[1]!='M'){
            std::cout << "erro de leitura!!!*\n";
            SAIR;
        }
        if (CTOI(Header[0x1E])!=0){
            std::cout << "erro de leitura!!!/\n";
            SAIR;
        }
        if (CTOI(Header[0x1C])!=24){
            std::cout << "erro de leitura!!!\\\n";
            SAIR;
        }

        // Recupera a informação dos atributos de
        // altura e largura da imagem
        Width   = CTOI(Header[0x12]);
        Height  = CTOI(Header[0x16]);
        ( CTOI(Header[0x0A]) == 0 ) ? ( DataPos=0x36 ) : ( DataPos =
    CTOI(Header[0x0A]) );

        imageSize=Width*Height*3;

        // Efetura a Carga da Imagem
        image = (GLubyte *) malloc ( imageSize );
        int retorno;
        retorno = fread(image,1,imageSize,fp_arquivo);

        if (retorno !=imageSize)
         {
            free (image);
            SAIR;
         }

        // Inverte os valores de R e B
```

```
    int t, i;

    for ( i = 0; i < imageSize; i += 3 )
     {
        t = image[i];
        image[i] = image[i+2];
        image[i+2] = t;
     }

    // Tratamento da textura para o OpenGL
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LIN-
EAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LIN-
EAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);

    glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE
);
    // Faz a geracao da textura na memoria
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, Width, Height, 0, GL_
RGB, GL_UNSIGNED_BYTE, image);

    fclose (fp_arquivo);
    free (image);
    return 1;
 }


static void initLight(){
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
}


static void loadLight() {
    GLfloat ambientColor[] = {0.5, 0.5, 0.5, 1};
    // determina o modelo de luz a ser utilizada
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientColor);
    glColorMaterial ( GL_FRONT, GL_AMBIENT_AND_DIFFUSE );

    glPushMatrix();
        glRotated(90,0,1,0);
        // determina o tipo de luz a ser utilizada
        GLfloat diffuseLight[] = { 0.7f, 0.7f, 0.4f, 1.0f };
        glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
        // aplica a luz na posição desejada
        GLfloat lightPos[] = {0.0f, 0.0f, -3.0f, 1.0f};
        glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
    glPopMatrix();
```

```
}

void desenhaParedes(float i) {

    /// Paredes
    // Aplica textra as paredes
    glBindTexture ( GL_TEXTURE_2D, texture_id[1] );
    // parede lateral
    glBegin(GL_POLYGON);
        glTexCoord2d(1,1);glVertex3d(i,0,0);
        glTexCoord2d(1,0);glVertex3d(i,i,0);
        glTexCoord2d(0,0);glVertex3d(0,i,0);
        glTexCoord2d(0,1);glVertex3d(0,0,0);
    glEnd();
    // parede de fundo com recorte para a janela
    glBegin(GL_QUADS);
        glColor4d(1,1,1,1);
        glTexCoord2d(0,1);glVertex3d(0,i,0);
        glTexCoord2d(0.66,1);glVertex3d(0,i,0.66*i);
        glTexCoord2d(0.66,0);glVertex3d(0,0,0.66*i);
        glTexCoord2d(0,0);glVertex3d(0,0,0);

        glTexCoord2d(0,1);glVertex3d(0,i,i*2);
        glTexCoord2d(0.33,1);glVertex3d(0,i,i*2-(0.66*i));
        glTexCoord2d(0.33,0);glVertex3d(0,0,i*2-(0.66*i));
        glTexCoord2d(0,0);glVertex3d(0,0,i*2);

        glTexCoord2d(0.33,1);glVertex3d(0,i,i*2-(0.66*i));
        glTexCoord2d(0.66,1);glVertex3d(0,i,0.66*i);
        glTexCoord2d(0.66,0.66);glVertex3d(0,0.66*i,0.66*i);
        glTexCoord2d(0.33,0.66);glVertex3d(0,0.66*i,i*2-
(0.66*i));

        glTexCoord2d(0.33,0);glVertex3d(0,0,i*2-(0.66*i));
        glTexCoord2d(0.66,0);glVertex3d(0,0,0.66*i);
        glTexCoord2d(0.66,0.33);glVertex3d(0,i-0.66*i,0.66*i);
        glTexCoord2d(0.33,0.33);glVertex3d(0,i-0.66*i,i*2-
(0.66*i));

    glEnd();
    // parede lateral
    glColor3d(1,1,1);
    glBegin(GL_QUADS);
        glTexCoord2d(1,1);glVertex3d(i,0,i*2);
        glTexCoord2d(1,0);glVertex3d(i,i,i*2);
        glTexCoord2d(0,0);glVertex3d(i,i,0);
        glTexCoord2d(0,1);glVertex3d(i,0,0);
    glEnd();
}
```

```
void desenhaChao(float tam){
    /// Chão
    // aplica textura no chao
    glBindTexture ( GL_TEXTURE_2D, texture_id[0]);
    // cria o chao
    glBegin(GL_QUADS);
        glTexCoord2d(1,1);glVertex3d(0,0,tam*2);
        glTexCoord2d(1,0);glVertex3d(tam,0,tam*2);
        glTexCoord2d(0,0);glVertex3d(tam,0,0);
        glTexCoord2d(0,1);glVertex3d(0,0,0);
    glEnd();
}

void desenhaJanela(double tam){
    glBegin(GL_QUADS);
        // cor com transparencia
        glColor4d(0.7,0.7,1,0.5);
        // desenhando a janela no centro da parede
        glVertex3d(0,tam*0.66  ,tam*2-(0.66*tam));
        glVertex3d(0,tam*0.66  ,0.66*tam);
        glVertex3d(0,tam-0.66*tam,0.66*tam);
        glVertex3d(0,tam-0.66*tam,tam*2-(0.66*tam));
    glEnd();
}

void espelho(double tam){
    glBegin(GL_QUADS);
        glVertex3d(tam-0.01,tam*0.66  ,tam*2-(0.66*tam));
        glVertex3d(tam-0.01,tam*0.66  ,0.66*tam);
        glVertex3d(tam-0.01,tam-0.66*tam,0.66*tam);
        glVertex3d(tam-0.01,tam-0.66*tam,tam*2-(0.66*tam));

    glEnd();
}

void desenhaEspelho(){
 glEnable(GL_STENCIL_TEST); //habilita o stencil buffer

 glColorMask(0, 0, 0, 0); //desabilita o uso de cores para a
tela
 glDisable(GL_DEPTH_TEST); //desabilita o teste de profundidade
 glStencilFunc(GL_ALWAYS, 1, 1); //Faz o teste de stenil sempre
passar
                                    // determina que os pixels
no stencil buffer
                                    // ser 1 quando passar pelo
stencil
 glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);    // configurar to-
dos os pixels para
                                                 // 1 quando
```

```
forem cobertos pelo stencil
 espelho(0.5);

 glColorMask(1, 1, 1, 1); // habilita o desenho de cores na tela
 glEnable(GL_DEPTH_TEST); // habilita o teste de profundidade
 // faz o teste de stencil passar somente quando
 // o pixel for no stencil buffer
 glStencilFunc(GL_EQUAL, 1, 1);
 glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP); // Faz com que o sten-
cil buffer não mude

    // desenha o reflexo do cubo
 glPushMatrix();
        glScalef(1, -1, 1);
        glTranslated(0,.35f,-5);
        glColor3d(0,0,1);
        glutSolidCube(0.3);
 glPopMatrix();

 glDisable(GL_STENCIL_TEST); //Disable using the stencil buffer

 glEnable(GL_BLEND);

 glColor4f(1, 1, 1, 0.6f);
 espelho(0.5);
 glDisable(GL_BLEND);
}

static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STEN-
CIL_BUFFER_BIT);

    initLight();

 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();

    gluLookAt(5,1,8,0,0,0,0,1,0);

    glPushMatrix();
        glColor3d(0,0,1);
       glTranslated(0,.35f,-5);
        glutSolidCube(0.3);
    glPopMatrix();

    glPushMatrix();
        glTranslated(0,-0.40,-4);
        glRotated(15,1,0,0);
        glRotated(-90,0,1,0);
```

```
            glPushMatrix();
                glTranslated(0,0,-1.5);
                glColor3d(1,1,1);
                desenhaParedes(1.5);
                desenhaChao(1.5);
                desenhaJanela(1.5);

            glPopMatrix();
        glPopMatrix();

        desenhaEspelho();

        glutSwapBuffers();
    }

    int main(int argc, char *argv[])
    {
        glutInit(&argc, argv);
        glutInitWindowSize(640,480);
        glutInitWindowPosition(10,10);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH |
    GLUT_STENCIL);

        glutCreateWindow("Sala");

        glEnable ( GL_TEXTURE_2D );
        glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
        glGenTextures ( 1, texture_id );

        glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
        LoadBMP ("F:\\lixo\\sala3d_w\\piso.bmp");

        glBindTexture ( GL_TEXTURE_2D, texture_id[1] );
        LoadBMP ("F:\\lixo\\sala3d_w\\parede.bmp");

        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

        glutReshapeFunc(resize);
        glutDisplayFunc(display);

        glClearColor(1,1,1,1);

        glutMainLoop();

        return EXIT_SUCCESS;
    }
```