Projeto: Construção de cena

Parte 1

Sumário

Sumário

Projeto

O projeto consiste em modelar os objetos de uma cena utilizando somente as funções do OpenGL . A cena é composta por uma mesa, quadrada ou redonda, cadeiras, vaso, taças, luminária e garrafa. Cada grupo irá compor a cena com no **mínimo** 3 objetos distintos. Ainda terá um chão (piso) presente em todas as cenas. Todos esses objetos podem ser construídos através de comandos da GLUT tais como as funções glutWireCube() e glutWireTorus(),e também das primitivas do próprio OpenGL (GL_QUADS, GL_TRIANGLES, etc).

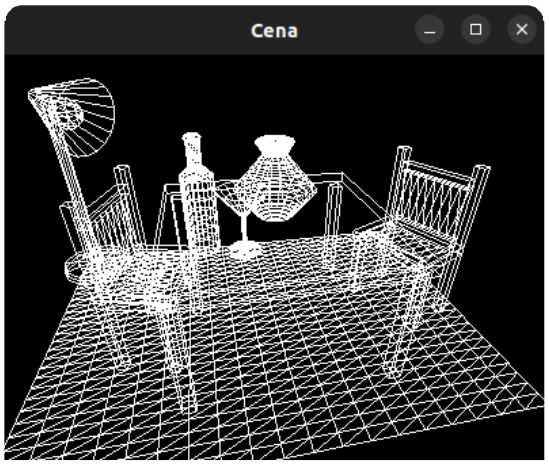


Figura 1 Cena completa com todos os objetos em wireframe

Não é necessário reproduzir fielmente a cena sugerida para cada grupo. Outras técnicas poderão ser exploradas de acordo com a criatividade do aluno para produzir, por exemplo, uma cena mais elaborada e com objetos dispostos de maneira diferente. Entretanto, a cena deve conter pelo menos os objetos sugeridos.
A série de atividades a seguir descreve como cada objeto pode ser construído e como a cena é gradativamente montada e melhorada.

- Modelagem Geométrica - Modelagem dos objetos da cena usando as primitivas do OpenGL.
- Transformações Geométrica- Composição da cena composta pelos objetos da atividade anterior.
- Projeções - Visualização da cena construída em diferentes pontos de vista.

A seguir o esqueleto dos códigos fontes que poderão ser usados para gerar a cena.
Algumas sugestões de:

- **Chão**, mesa, cadeiras e vaso
- **Chão,** mesa, cadeira e luminária
- **Chão,** mesa, cadeira e taças
- **Chão,** mesa, cadeira e garrafa
- **Chão,** mesa, taças e garrafa.
- **Chão,** mesa, taças e vaso.

- **Chão,** mesa, luminária e taça
- **Chão,** mesa, luminária e garrafa
- **Chão,** mesa, luminária e vaso
- **Chão,** mesa, vaso e garrafa
- **Chão,** luminária, cadeira e vaso
- **Chão,** cadeiras, garrafa e taça
- **Chão,** luminária, cadeira e taça
- **Chão,** cadeira, taça e vaso.

Inicialmente será modelado cada objeto que irá compor a cena e, em seguida, posicionados corretamente para obter o resultado final desejado.

Comentários Gerais:

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar;
- O trabalho é em dupla (grupo de **DOIS** alunos);
- Trabalhos copiados (FONTE) terão nota zero.
- Trabalhos entregues em atraso serão aceitos, todavia a nota atribuída ao trabalho será diminuída em 10%;

Critérios de Avaliação:

1. Atendimento aos requisitos.
2. Qualidade do código (estruturação e comentário).
3. Qualidade do programa gráfico (complexidade da cena).

O que deve ser submetido?

- Código-fonte do trabalho, com comentários e indicação dos membros do grupo no ínicio do código.

- **Data da entrega: 14/08/2024 (impreterivelmente)**

Modelagem dos objetos que compõem a cena.

Modelagem da mesa

A mesa pode ser modelada usando apenas a função glutWireCube da GLUT que gera um cubo unitário na origem. Podem ser utilizadas transformações de escala sobre diferentes cubos para criar o tampo e os pés da mesa. Por exemplo, o pés da mesa são simplesmente cubos alongados no eixo Y. O tampo da mesa é um cubo achatado no eixo Y e alongado no eixo X e Z.
Use o seguinte esqueleto de código para implementar essa tarefa:

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
#define MESA 1

void init(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

  // Inicializa display lists da mesa
  glNewList(MESA, GL_COMPILE); // Mesa

 // Use aqui as primitivas e transformações geométricas do OpenGL para modelar a mesa.

 glEndList();
}

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glLoadIdentity();

  // Chama o display list da mesa para exibi-la
  glPushMatrix();
        glCallList(MESA);
  glPopMatrix();
 // término do posicionamento da mesa

  glutSwapBuffers();
```

```
}

void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
   //projeção perspectiva
  gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 100.0);
  gluLookAt(-10.0, 30.0, 50.0,0.0, -2.0, 0.0, 0.0, 1.0, 0.0);
  //projeção ortográfica
  // glOrtho(-30.0,30.0, -30.0,30.0, 1.0,250.0);
            // vista de cima
            //gluLookAt(0.0, 250.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
            // vista frontal
            // gluLookAt(0.0, 0.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
     glMatrixMode(GL_MODELVIEW);
}

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

int main()
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(400, 300);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Mesa");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

Dicas:

- Comece com apenas um cubo unitário na origem usando a função **glutWireCube**(1.0). Esse procedimento é útil para se ter uma noção de escala entre as dimensões dos objetos da cena e o tamanho da janela de visualização.
- Para posicionar corretamente os pés sob a mesa, use o comando **glTranslate*** dentro de blocos de funções **glPushMatrix** e **glPopMatrix**. Faça o mesmo para o tampo da mesa. É recomendável que a mesa seja modelada num ***display list*** para que sua posterior exibição seja mais simples e ao mesmo tempo eficiente.
- Para verificar se os componentes de cada objeto estão posicionados corretamente (por exemplo, se o tampo da mesa está realmente repousando sobre os pés da mesa), visualize o objeto inicialmente com projeção ortográfica (use a função **glOrtho**). Esse procedimento ajuda a evitar efeitos de ilusão de óptica resultantes da distorção da projeção perspectiva e do uso de geometria em *wireframe*. Estas visualizações estão demonstradas noa figura 2.
- Desenha um cubo

  void glutSolidCube(GLDouble size), void glutWireCube(GLDouble size)

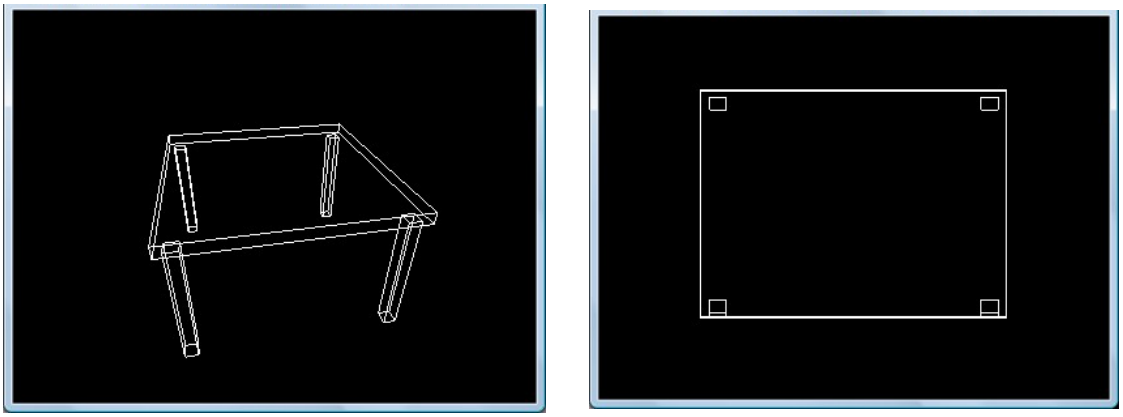
  glutSolidCube and glutWireCube gera um cub solid ou wireframe rspectivamente.

  O paramento *size* é o tamanho de cada lado do cubo

Observação: Para visualizar os objetos em *wireframe* utilize o comando glPolygonMode(GL_FRONT_AND_BACK, GL_LINE). Esse comando também habilita a visualização de polígonos que não estão defrontando o observador.

Sinta-se livre para construir um modelo diferente do sugerido.

(a) (b)



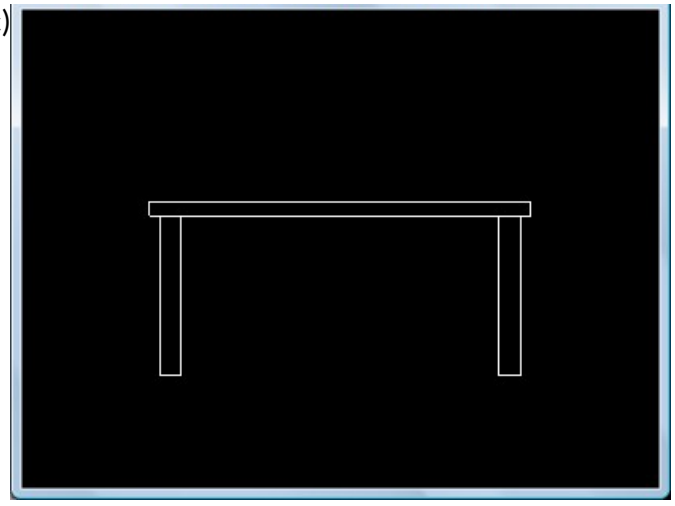
(c) 

Figura 2 (a) Projeção Perspectiva (b) Projeção ortográfica (vista de cima) (c) Projeção ortográfica (vista frontal)

Modelagem da taça

A taça foi feito com dois cilindros. O comando gluCylinder gera cilindros abertos em cima e em baixo e um cone. Para tampar um cilindro podemos utilizar o comando gluDisk, também da GLU. Esse procedimento foi adotado na modelagem da base do cálice (o cilindro achatado). Use o seguinte esqueleto de código para implementar essa tarefa:

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

#define TACA 1
GLUquadricObj *q;

void init(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  q = gluNewQuadric();

  // Inicializa display lists
  glNewList(TACA, GL_COMPILE); // taça

   // Use aqui as primitivas e transformações geométricas  do OpenGL para modelar a taça.

   glEndList();
}

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glLoadIdentity();

  // Chama o display list do taça para exibi-lo
  glPushMatrix();
          glCallList(TACA);
  glPopMatrix();
 // término do posicionamento da taça

  glutSwapBuffers();
}
```

```
void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  //projeção perspectiva
 gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 100.0);
 gluLookAt(-10.0, 30.0, 50.0,0.0, -2.0, 0.0, 0.0, 1.0, 0.0);
  //projeção ortográfica
 // glOrtho(-30.0,30.0, -30.0,30.0, 1.0,250.0);
            // vista de cima
            // gluLookAt(0.0, 250.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
             // vista frontal
             //gluLookAt(-5.0, 0.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  glMatrixMode(GL_MODELVIEW);
}

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

int main()
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(400, 300);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Taca");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

Dica:

Desenha o cilindo

- A função g**luCylinder**() possui o seguinte protótipo:

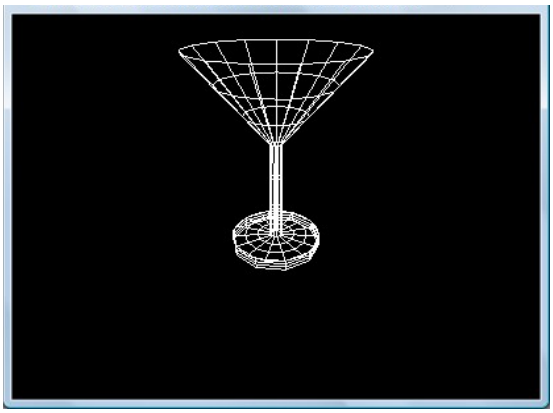
void gluCylinder(GLUquadric* quad, GLdouble base, GLdouble top, GLdouble height, GLint slices, GLint stacks);

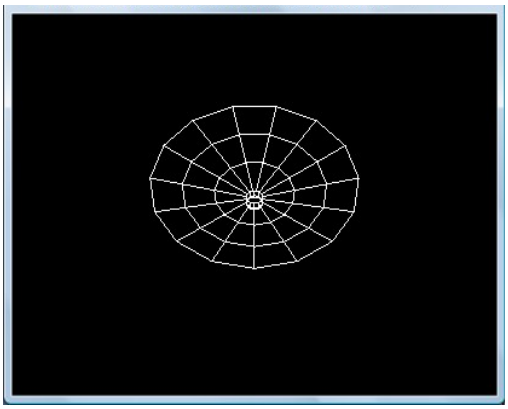
- O parâmetro *quad* é o objeto de quádrica; *base*, *top* e *height* especificam o raio da base, o raio do topo e a altura do cilindro, respectivamente; *slices stacks* especificam o número de subdivisões ao redor do eixo z e ao longo do mesmo.

- Desenha o disco
  void **gluDisk** (GLUquadricObj *qogj , GLdouble innerRadius , GLdouble outRadius , Glint slices , GLint rings);
  - O parâmetro *quad* é o objeto de quádrica, innerRadius e outerRadius especificam o raio interno do disco e o raio externo do disco, respectivamente; slices o número de fatias (como fatias de pizza) que o disco é subdivido ao longo do eixo z. Rings o número de anéis concêntrico que o disco é dividido sobre o eixo z.

- Use o comando glTranslate* para colocar um cilindro sobre o outro. Use também o comando glRotate* para colocar cada cilindro "em pé", uma vez que o comando gluCylinder gera um cilindro deitado, isto é, ao longo do eixo Z na origem. O mesmo se aplica ao disco.
- A figura 3 mostra a taça utilizando projeção perspectiva e projeção ortográfica vista de cima e vista frontal. Para obter as visualizações é preciso modificar o tipo de projeção no código fonte.

(a) **(b)**





(c)

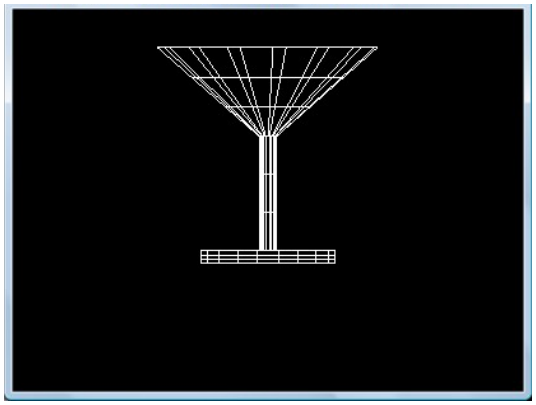


Figura 3 (a) Projeção Perspectiva (b) Projeção ortográfica (vista de cima) (c) Projeção ortográfica (vista frontal)

Modelagem da luminária

A luminária pode ser modelada a partir do modelo do cálice. Para tanto basta definirmos uma haste mais alongada e, no topo, rotacionarmos o copo de modo que sua boca fique direcionada ao centro da mesa. Além disso, a luminária contém uma esfera que simboliza a lâmpada. A esfera pode ser criada com o comando ***glutSphere*** e posicionada no local correto através de chamadas a glTranslate*. Use o seguinte esqueleto de código para implementar essa tarefa:

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

#define LUMINARIA 1
GLUquadricObj *q;

void init(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  q = gluNewQuadric();

  // Inicializa display lists
  glNewList(LUMINARIA, GL_COMPILE); // Luminaria

        // Use aqui as primitivas e transformações geométricas
        // do OpenGL para modelar a luminaria.

  glEndList();
}

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glLoadIdentity();
  // Chama o display list da luminaria para exibi-la
  glPushMatrix();
          glCallList(LUMINARIA);
  glPopMatrix();
 // término do posicionamento da luminária
  glutSwapBuffers();
}
```

```
void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  //projeção perspectiva
  gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 100.0);
  gluLookAt(-10.0, 30.0, 50.0,0.0, -2.0, 0.0, 0.0, 1.0, 0.0);
  //projeção ortográfica
  //glOrtho(-30.0,30.0, -30.0,30.0, 1.0,250.0);
            // vista de cima
            //gluLookAt(0.0, 250.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
            // vista frontal
            //gluLookAt(-5.0, 0.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  glMatrixMode(GL_MODELVIEW);
}

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

int main()
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(400, 300);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Luminaria");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

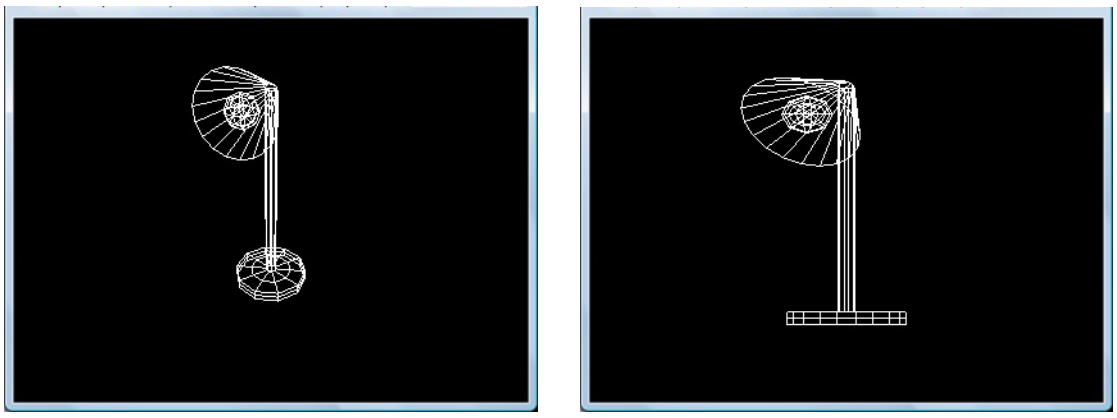
Dica:

- Desenha o cilindro
  A função gluCylinder() possui o seguinte protótipo:
  void gluCylinder(GLUquadric* quad, GLdouble base, GLdouble top, GLdouble height, GLint slices, GLint stacks);
  O parâmetro q*uad* é o objeto de quádrica; *base*, *top* e *height* especificam o raio da base, o raio do topo e a altura do cilindro, respectivamente; *slices stacks* especificam o número de subdivisões ao redor do eixo z e ao longo do mesmo.

- Desenha o disco
  void (GLUquadricObj *qogj , GLdouble innerRadius , GLdouble outRadius , Glint slices , GLint rings);
  O parâmetro *quad* é o objeto de quádrica, innerRadius e outerRadius especificam o raio interno do disco e o raio externo do disco, respectivametne; sides o número de fatias (como fatias de pizza) que o disco é subdivido ao longo do eixo z. Rings o número de anéis concêntrico que o disco é dividido sobre o eixo z.

- Use o comando glTranslate* para colocar um cilindro sobre o outro. Use também o comando glRotate* para colocar cada cilindro "em pé", uma vez que o comando gluCylinder gera um cilindro deitado, isto é, ao longo do eixo Z na origem. O mesmo se aplica ao disco.
- A figura 4 mostra a lunimária utilizando projeção perspectiva e projeção ortográfica vista de cima e frontal. Para obter as visualizações é preciso modificar o tipo de projeção no código fonte.

(a) (b)



(c)

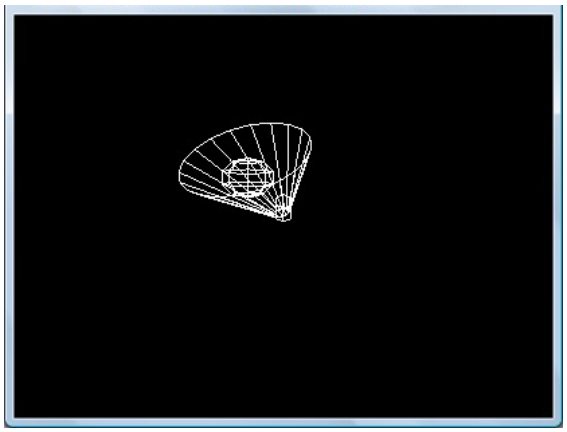


Figura 4 (a) Projeção Perspectiva (b) Projeção ortográfica (vista de cima) (c) Projeção ortográfica (vista frontal)

Modelagem das cadeiras

Na cena que queremos construir, as duas cadeiras são idênticas e apenas diferem entre si por estarem em posições diferentes. Desse modo, podemos modelar apenas uma cadeira, armazená-la num display list e chamar glCallList duas vezes para criar rapidamente as duas cadeiras. A modelagem da cadeira nesse exemplo foi feita usando o comando glutWireCube. Os cubos (distorcidos com glScale*) foram utilizados para fazer, por exemplo, os pés da cadeira.
O encosto e o assento podem ser gerados com primitiva a GL_TRIANGLE_STRIP do OpenGL.
Use o seguinte esqueleto de código para implementar essa tarefa:

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
#define CADEIRA 1

void init(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  glEnable(GL_AUTO_NORMAL);
  glEnable(GL_NORMALIZE);

  // Inicializa display lists
  glNewList(CADEIRA, GL_COMPILE); // Cadeira
          //
          // Use aqui as primitivas e transformacoes geometricas
          // do OpenGL para modelar a cadeira.
          //
  glEndList();
}

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glLoadIdentity();

  // Chama o display list da cadeira para exibi-la
  glPushMatrix();
    glCallList(CADEIRA);
  glPopMatrix();
 // término do posicionamento da cadeira
```

```
  glutSwapBuffers();
}

void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
   //projeção perspectiva
  gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 100.0);
  gluLookAt(-30.0, 30.0, 50.0,0.0, -2.0, 0.0, 0.0, 1.0, 0.0);
  //projeção ortográfica
 // glOrtho(-30.0,30.0, -30.0,30.0, 1.0,250.0);
  / vista frontal
  //gluLookAt(0.0, 0.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  glMatrixMode(GL_MODELVIEW);
}

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

int main()
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(400, 300);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Cadeira");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

A figura 5 mostra a cadeira utilizando projeção perspectiva e projeção ortográfica vista frontal. Para obter as visualizações é preciso modificar o tipo de projeção no código fonte.

(a) (b)

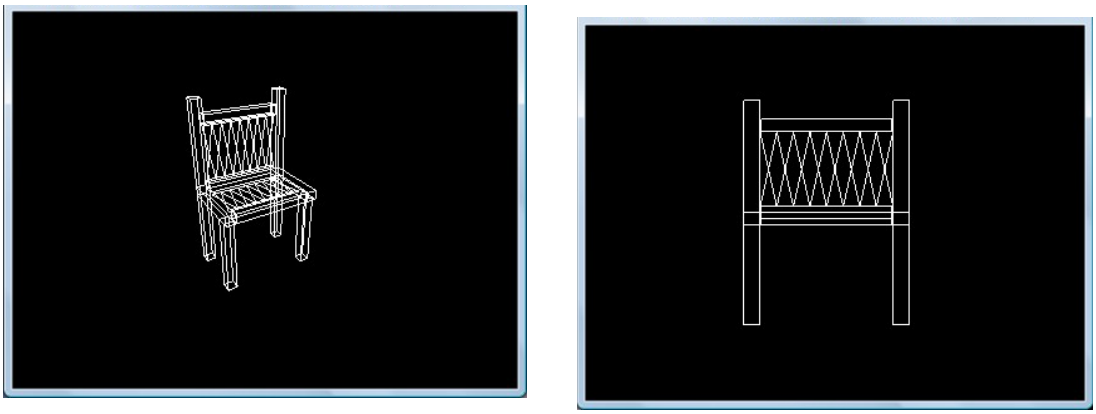


Figura 5 (a) Projeção Perspectiva (b) Projeção ortográfica (vista frontal)

Modelagem da garrafa

A modelagem da garrafa foi feita com três cilindros um sobre o outro usando o comando gluCylinder. Para gerar a tampa da garrafa foi utilizada o comando glutWireTorus.
Use o seguinte esqueleto de código para implementar essa tarefa:

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
#define GARRAFA 1
GLUquadricObj *q;
void init(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  q = gluNewQuadric();

  // Inicializa display lists
  glNewList(GARRAFA, GL_COMPILE); // GARRAFA

          // Use aqui as primitivas e transformacoes geometricas
          // do OpenGL para modelar o GARRAFA.
    glEndList();
}

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glLoadIdentity();

  // Chama o display list do vaso para exibi-lo
  glPushMatrix();
    glCallList(GARRAFA);
  glPopMatrix();
 // término do posicionamento da garrafa

  glutSwapBuffers();
}

void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei)w, (GLsizei)h);
```

```
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();

  //projeção perspectiva
 gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 100.0);
 gluLookAt(-10.0, 30.0, 50.0,0.0, -2.0, 0.0, 0.0, 1.0, 0.0);
 //projeção ortográfica
 //glOrtho(-30.0,30.0, -30.0,30.0, 1.0,250.0);
          // vista de cima
           //  gluLookAt(0.0, 250.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
           // vista frontal
           //gluLookAt(-5.0, 0.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  glMatrixMode(GL_MODELVIEW);
}

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

int main()
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
 glutInitWindowSize(400, 300);
 glutInitWindowPosition(100, 100);
 glutCreateWindow("Garrafa");
 init();
 glutDisplayFunc(display);
 glutReshapeFunc(reshape);
 glutKeyboardFunc(keyboard);
 glutMainLoop();
 return 0;
}
```

Dica:

- Use o comando glTranslate* para colocar um cilindro sobre o outro. Use também o comando glRotate* para colocar cada cilindro "em pé", uma vez que o comando gluCylinder gera um cilindro deitado, isto é, ao longo do eixo Z na origem. O mesmo se aplica ao toróide.

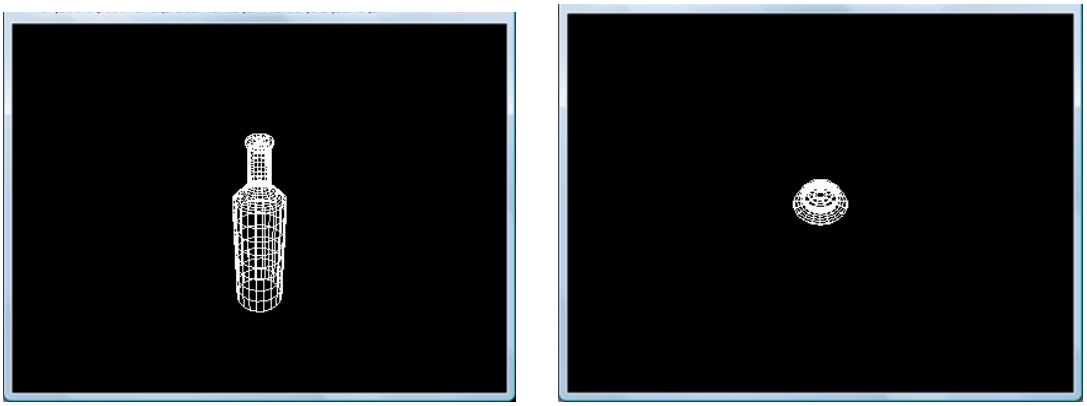
Através das imagens abaixo pode-se ter uma idéia das dimensões do modelo. Sinta-se livre para construir um modelo diferente do sugerido.

- Desenha o cilindro
  A função gluCylinder() possui o seguinte protótipo:
  void gluCylinder(GLUquadric* quad, GLdouble base, GLdouble top, GLdouble height, GLint slices, GLint stacks);
  O parâmetro *quad* é o objeto de quádrica; *base*, *top* e *height* especificam o raio da base, o raio do topo e a altura do cilindro, respectivamente; *slices stacks* especificam o número de subdivisões ao redor do eixo z e ao longo do mesmo.

- Desenha o toróide
  A função glutSolidTorus() ou glutWireTorus() possui o seguinte protótipo:
  void **glutSolidTorus** (GLdouble innerRadius,GLdouble outerRadius,Glint nsides,Glint rings)
  void **glutWireTorus** (GLdouble innerRadius,GLdouble outerRadius,Glint nsides,Glint rings)
- Os parâmetros innerRadius e outerRadius especificam o raio interno do toróide e o raio externo do toróide, respectivametne; nsides o número de lados para cada seção radial e rings o número de subdivisões radiais do toróide.

- A figura 6 mostra a garrafa utilizando projeção perspectiva e projeção ortográfica vista de cima e frontal. Para obter as visualizações é preciso modificar o tipo de projeção no código fonte.

(a) (b)



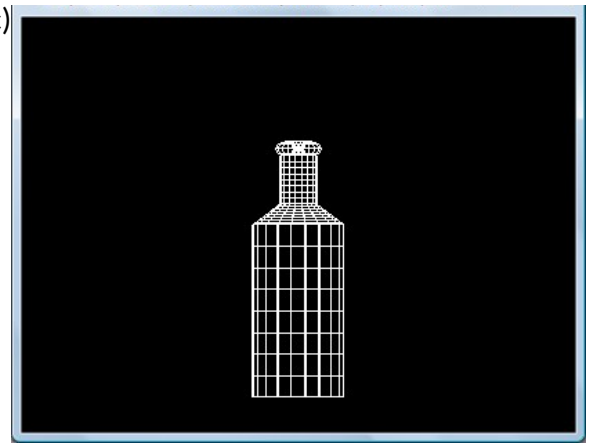
(c) 

Figura 6 (a)Projeção Perspectiva (b)Projeção ortográfica(vista de cima)(c) Projeção ortográfica (vista frontal)

Modelagem do chão

O chão pode ser uma grade retangular gerada com primitivas GL_TRIANGLES, GL_QUADS, GL_TRIANGLE_STRIP ou GL_QUAD_STRIP do OpenGL ou atém mesmo um cubo gerado por glutWireCube. Infelizmente, se for necessário gerar o chão por malhas triangulares, não existem funções da GLUT ou da GLU pra isto. Por outro lado, a criação dessa malha usando as primitivas citadas é relativamente simples e pode ser feita com apenas dois laços aninhados (uma interação em X e outra interação em Z). O chão também pode ser armazenado num *display list* para facilitar seu posicionamento posterior na cena
Use o seguinte esqueleto de código para implementar essa tarefa:

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

#define CHAO 1
void init(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

  // Inicializa display lists
  glNewList(CHAO, GL_COMPILE); // Chao
    glPushMatrix();

          // Use aqui duas estruturas de repetição aninhadas para gerar uma malha de
          // triângulos, quadrados (ou outras primitivas) para modelar o chão.
          // OU
          // Use aqui as primitivas e transformações geométricas do OpenGL para modelar o
chão

    glPopMatrix();
  glEndList();
}

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glLoadIdentity();

  // Chama o display list do chao para exibi-lo
  glPushMatrix();
```

```
   glCallList(CHAO);
  glPopMatrix();
 // término do posicionamento do chão

  glutSwapBuffers();
}

void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
   //projeção perspectiva
  gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 100.0);
  gluLookAt(-10.0, 30.0, 50.0,0.0, -2.0, 0.0, 0.0, 1.0, 0.0);
  //projeção ortográfica
  //glOrtho(-30.0,30.0, -30.0,30.0, 1.0,250.0);
  // vista de cima
  // gluLookAt(0.0, 150.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  glMatrixMode(GL_MODELVIEW);
}

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

int main()
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(400, 300);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Chao");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

A figura 7 mostra o chão gerado por malhas triangulares utilizando projeção perspectiva e projeção ortográfica vista de cima. Para obter as visualizações é preciso modificar o tipo de projeção no código fonte. A figura 8 mostra o chão gerada através do objeto cubo.

(a) (b)

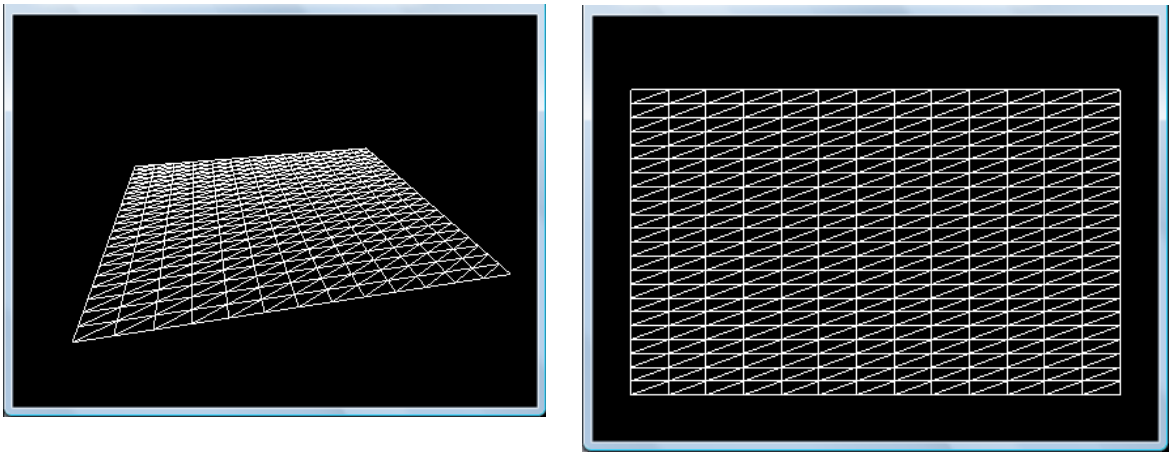


Figura 7 (a) Projeção Perspectiva (b) Projeção ortográfica (vista de cima)

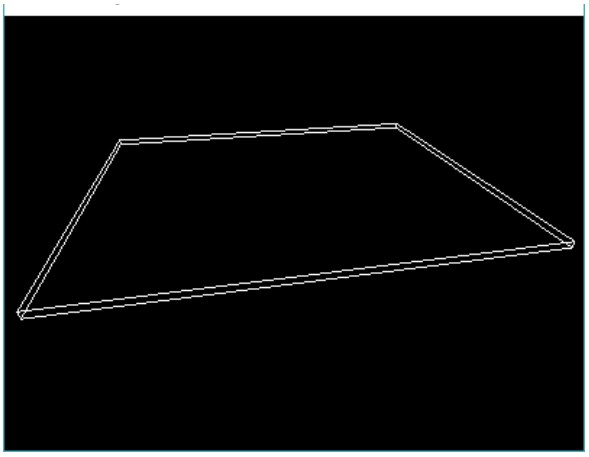


Figura 8: Chão gerado com glutWireCube

Modelagem do vaso

O vaso mostrado abaixo foi construído apenas com funções da GLUT e da GLU. O corpo do vaso foi feito com três cilindros um sobre o outro usando o comando gluCylinder com diâmetros diferentes para o início e fim de cada cilindro. Finalmente, a boca do vaso foi finalizada com um toróide gerado através do comando glutWireTorus. É importante que o vaso seja definido num *display list*, pois essa estratégia vai facilitar seu posicionamento sobre a mesa no próximo projeto. Use o seguinte esqueleto de código para implementar essa tarefa:

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
#define VASE 2
//criação o objeto quádrica
GLUquadricObj *q;

void init(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  q = gluNewQuadric();

  // Inicializa display lists
  glNewList(VASE, GL_COMPILE); // Vaso

    // Use aqui as primitivas e transformacoes geométricas  do OpenGL para modelar o vaso.

   glEndList();

}
void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glLoadIdentity();

  // Chama o display list do vaso para exibi-lo
  glPushMatrix();
      glCallList(VASE);
  glPopMatrix();
  glutSwapBuffers();
```

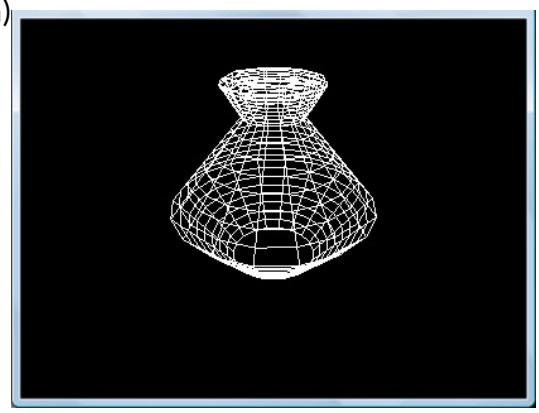
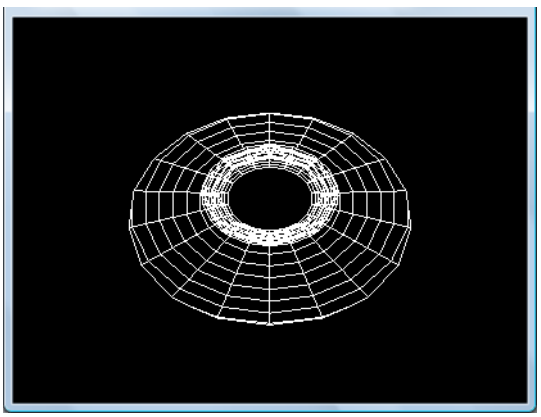
```
}
// término do posicionamento do vaso

void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
   //projeção perspectiva
     gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 100.0);
     gluLookAt(-10.0, 30.0, 50.0,0.0, -2.0, 0.0, 0.0, 1.0, 0.0);
  //projeção ortográfica
  //glOrtho(-30.0,30.0, -30.0,30.0, 1.0,250.0);
            // vista de cima
           //gluLookAt(0.0, 250.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
            // vista frontal
           //gluLookAt(-5.0, 0.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    glMatrixMode(GL_MODELVIEW);
}

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

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(400, 300);
  glutInitWindowPosition(100, 100);
  glutCreateWindow(argv[0]);
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

Dica:

- Use o comando glTranslate* para colocar um cilindro sobre o outro. Use também o comando glRotate* para colocar cada cilindro "em pé", uma vez que o comando gluCylinder gera um cilindro deitado, isto é, ao longo do eixo Z na origem. O mesmo se aplica ao toróide.
  Através das imagens abaixo pode-se ter uma ideia das dimensões do modelo. Sinta-se livre para construir um modelo diferente do sugerido.

- Desenha o cilindro
  A função gluCylinder() possui o seguinte protótipo:
  void gluCylinder(GLUquadric* quad, GLdouble base, GLdouble top, GLdouble height, GLint slices, GLint stacks);
  O parâmetro *quad* é o objeto de quádrica; *base*, *top height* especificam o raio da base, o raio do topo e a altura do cilindro, respectivamente; *slices e stacks* especificam o número de subdivisões ao redor do eixo z e ao longo do mesmo.

- Desenha o toróide
  A função glutSolidTorus() ou glutWireTorus() possui o seguinte protótipo:
  void **glutSolidTorus** (GLdouble innerRadius,GLdouble outerRadius,Glint nsides,Glint rings)
  void **glutWireTorus** (GLdouble innerRadius,GLdouble outerRadius,Glint nsides,Glint rings)
  Os parâmetros innerRadius e outerRadius especificam o raio interno do toróide e o raio externo do toróide, respectivametne; nsides o número de lados para cada seção radial e rings o número de subdivisões radiais do toróide.
- A figura 9 mostra o vaso utilizando projeção perspectiva e projeção ortográfica vista de cima e frontal. Para obter as visualizações é preciso modificar o tipo de projeção no código fonte.

(a)  (b) 

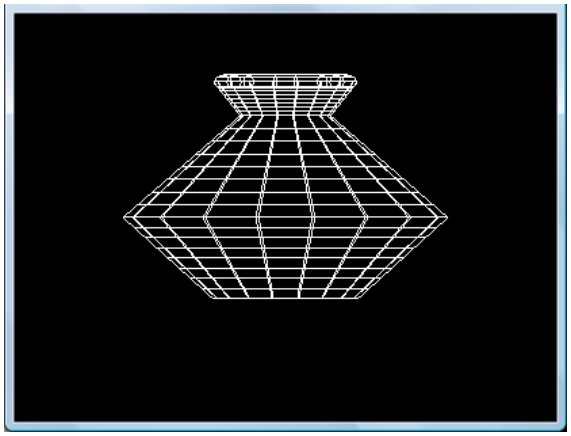
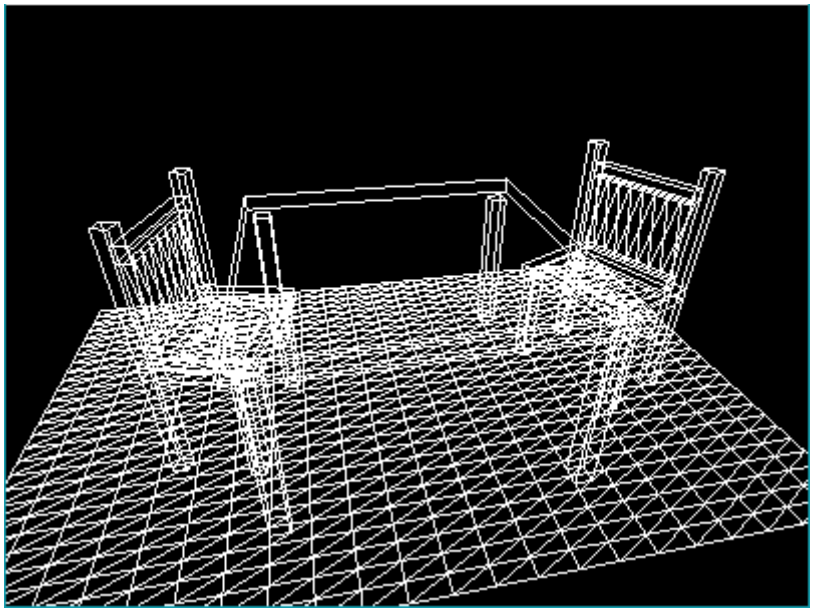
(c) 

Figura 9 (a) Projeção Perspectiva (b) Projeção ortográfica (vista de cima) (c) Projeção ortográfica (vista frontal)

Exemplo de composição da Cena

Composição da Cena - Mesa, Cadeiras e chão



Uma vez que todos os objetos da cena já foram construídos anteriormente, o objetivo dessa fase é dispor apropriadamente cada objeto no espaço de modo a compor a cena. A mesa e as cadeiras devem ficar sobre o chão. A execução dessa tarefa é bastante simples e eficiente desde que cada objeto tenha sido armazenado num *display list* como sugerido nas tarefas anteriores. Assim, basta fazer uma chamada glCallList para cada objeto dentro de um bloco glPushMatrix e glPopMatrix contendo as transformações geométricas (**através de glTranslate*, glScale* e glRotate***) que definem uma localização no espaço. Para verificar se a disposição dos objetos na cena está correta, utilize a projeção ortográfica através do comando glOrtho.
Use o seguinte esqueleto de código para implementar essa tarefa

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

#define MESA 1
#define CADEIRA 2
#define CHAO 3
GLUquadricObj *qGarrafa;
```

```
void init(void)
{
 glClearColor(0.0, 0.0, 0.0, 0.0);
 glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

 // Inicializa as display lists

 glNewList(CHAO, GL_COMPILE); // Chao
        // Use aqui dois lacos aninhados para gerar uma malha de
        // triangulos, quadrados (ou outras primitivas) para modelar o chão
    glEndList();

 //---------------------------------------------------------------------------
  glNewList(MESA, GL_COMPILE); // Mesa
        // Use aqui as primitivas e transformacoes geometricas
        // do OpenGL para modelar a mesa.
  glEndList();

 //-------------------------------------------------------------
 glNewList(CADEIRA, GL_COMPILE); // Cadeira
        // Use aqui as primitivas e transformacoes geometricas
        // do OpenGL para modelar a cadeira.
  glEndList();

 }

void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(1.0, 1.0, 1.0);
 glLoadIdentity();

 // Chama o display list do chao para exibi-lo
 glPushMatrix();
  //....
   glCallList(CHAO);
 glPopMatrix();
// término do posicionamento do chão

// Chama o display list da mesa para exibi-la
 glPushMatrix();
   //....
   glCallList(MESA);
 glPopMatrix();
// término do posicionamento da mesa
```

```
 // Chama o display list da cadeira 1 para exibi-lo
  glPushMatrix();
     // ...
     glCallList(CADEIRA);
  glPopMatrix();
 // término do posicionamento da cadeira 1

  // Chama o display list da cadeira 2 para exibi-lo
  glPushMatrix();
     //....
    glCallList(CADEIRA);
  glPopMatrix();
 // término do posicionamento da cadeira 2

   glutSwapBuffers();

}

void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
   //projeção perspectiva
  gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 100.0);
  gluLookAt(-10.0, 30.0, 50.0,0.0, -2.0, 0.0, 0.0, 1.0, 0.0);

  //projeção ortografica
  //glOrtho(-30.0,30.0, -30.0,30.0, 1.0,250.0);
  //gluLookAt(-5.0, 0.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  glMatrixMode(GL_MODELVIEW);
}

void keyboard(unsigned char key, int x, int y)
{
  switch(key) {
  case 27:
    gluDeleteQuadric(qGarrafa);
     exit(0);
     break;
  }
}

int main()
```

```
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(400, 300);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Cena");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```