BANGALORE UNIVERSITY

UNIVERSITY VISVESVARAYA COLLEGE OF ENGINEERING

K. R. CIRCLE, BENGALURU - 560001

For Quality and Excellence in Higher Education

Department of Computer Science and Engineering

A MINI PROJECT REPORT

ON

"MULTIPLE IMAGE REFLECTION"

Submitted by

CHARAN M K

(17GAEC9017) V SEM CSE

Under the guidance of

Department of Computer Science and Engineering

DECEMBER 2019/ JANUARY 2020

BANGALORE UNIVERSITY

UNIVERSITY VISVESVARAYA COLLEGE OF ENGINEERING

K. R. CIRCLE, BENGALURU - 560001

DECEMBER 2019/ JANUARY 2020

BANGALORE UNIVERSITY

UNIVERSITY VISVESVARAYA COLLEGE OF ENGINEERING

K. R. CIRCLE, BENGALURU - 560001

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the mini project entitled **"Multiple Image Reflection"** is a bona fide work carried out by **CHARAN MK (17GAEC9017)** a bona fide student of **University Visvesvaraya College of Engineering** in partial fulfillment for the mini project in Computer Graphics Laboratory, 5th semester, Computer Science & Engineering, Bangalore University, Bengaluru during the year 2019-20. It is certified that all the corrections/suggestions indicated for Internal Assessment have been incorporated in the Report. The mini Project Report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said degree.

Dept. of CSE UVCE

Dr. CHAMPA H N
Chairperson
Dept. of CSE UVCE

Examiner 1:_____

Examiner 2:_____

ACKNOWLEDGEMENT

I take this opportunity to thank our institution **University Visvesvaraya College of Engineering** for having given us an opportunity to carry out this project.

I would like to thank **Dr. Ramesh H N, Principal, UVCE**, for providing us all the facilities to work on this project. I am indebted to him for being our pillar of strength and inspiration.

I wish to place my grateful thanks to **Dr. Champa H N, Chairperson, Department of Computer Science and Engineering, UVCE,** who helped me to make my project a great success.

I am grateful to **Sampath, Department of Computer Science and Engineering**, **UVCE,** for her valuable suggestions and support, which has sustained me throughout the course of the project.

- CHARAN M K

17GAEC9017

Contents

1. INTRODUCTION

Overview of the project

It is developed in C using OpenGL and implemented in the WINDOWS platform. The graphics package designed here provides an interface for the users for handling display and manipulation of basic picture objects. The interface is user-friendly with icons, menus.

In the project 'MULTIPLE IMAGE REFLECTION', the computer graphics concepts along with the OpenGL functions are used. It consist of geometric objects such as sphere, cone, teapot etc. The "multiple image reflection", which is implemented using OpenGL, makes the viewer familiar to different viewing concepts, lighting and shading concepts and many tools available in OpenGL. This project has a view of a room with image reflection. The main intension is viewing the object in user convenient angles. Using keyboard function, the view of the room can be changed. And using menu function we can also change the rotating objects around the cone as required. And the view can be adjusted with speed of the rotating object. The room has many mirrors, a cone and a rotating object around the cone. We can see rotation of objects in mirrors. We also use coloring in OpenGL, lighting and shadows, menus etc. This project has keyboard interactions. In the program it is implemented such that whenever the specified key in the keyboard is pressed, the top view of the room is visible. Also by using some specified keys we can increase the number of mirrors in the room.

General Constraints

1. As software is being built to run on a WINDOWS platform, efficient use of the memory is very important.

2. The code should be efficient and optimal with the minimal redundancies.

3. Needless to say, the package should also be robust and fast.

Assumptions and Dependencies

1. It is assumed that the standard output device, namely the monitor, supports colors and users system is required to have the C compiler for the appropriate version.

2. The system is also expected to have a mouse connected since most of the drawing and other graphical operations implemented assume the presence of a mouse.

2. REQUIREMENTS SPECIFICATION

The requirement specifications of this project is not perfectly optimized. However the following hardware and software specifications were done to be of my best effort. Here are the specifications:

1. Hardware Requirements:

The hardware requirements given here is minimal requirements for the project to run even though the project can smoothly run on almost all i3h86 machines.

- Processor Speed -300 MHz and above
- Ram Size -64 Mb or above
- Storage Space -2MB or above

2. Software Requirements:

- Operating System -Windows Family
- Compiler - CODBLOCKS
- Graphics Library - GL/glut.h, GL/glu.h, GL/gl.h
- Programming Language -C using OpenGL

3. ABOUT OPENGL

The OpenGL specification describes an abstract for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware.

The API is defined as a number of functions which may be called by the client program, alongside a number of named integer constants. the function definition are superficially similar to those of the C programming language, they are independent. As such, OpenGl has many language bindings, some of the most noteworthy being the JavaScript binding WebGL(API based on OpenGL ES2.0);the C bindings WGL ,GLX, CGL; the C binding provided by iOS and the Java and C binding provided by the Android.

In addition to being language-independent, OpenGL is also platform independent. The specification says nothing on the subject of obtaining, managing an OpenGL context, leaving this as a detail of the underlying windowing system. For the same reason, OpenGL is purely concerned with the rendering, providing no APIs related to input, audio, or windowing.

6. IMPLEMENTATION

Rotation: Rotation concept is exhibited by rotating the 3 dimensional objects, whenever they are clicked. This is implemented by using OpenGL library functions as shown below,

```
glRotatef(-90,1,0,0,);
glRotatef(angle,0,1,0);
```

Scale: Scaling is used in the project is to change the rotating object scale in one revolution for every 10 seconds. The OpenGL function is

```
glScalef(0.18,0.18,0.18);
glScalef(m->scale[0], m->scale[1], m->scale[2]);
glScalef(1./m->scale[0], 1./m->scale[1], 1./m->scale[2]);
```

Translate: This concept is used many times in the program. It is used in positioning the objects with respect to the cone. For example,

```
glTranslatef(m->trans[0], m->trans[1], m->trans[2]);
glTranslatef(-m->trans[0], -m->trans[1], -m->trans[2]);
glTranslatef(0, -1, 0);
glTranslatef(0, -.3, 0);
glTranslatef(.6, 0, 0);
```

Coloring: The objects in the room are given different colors to improve the looks of the project. For example,

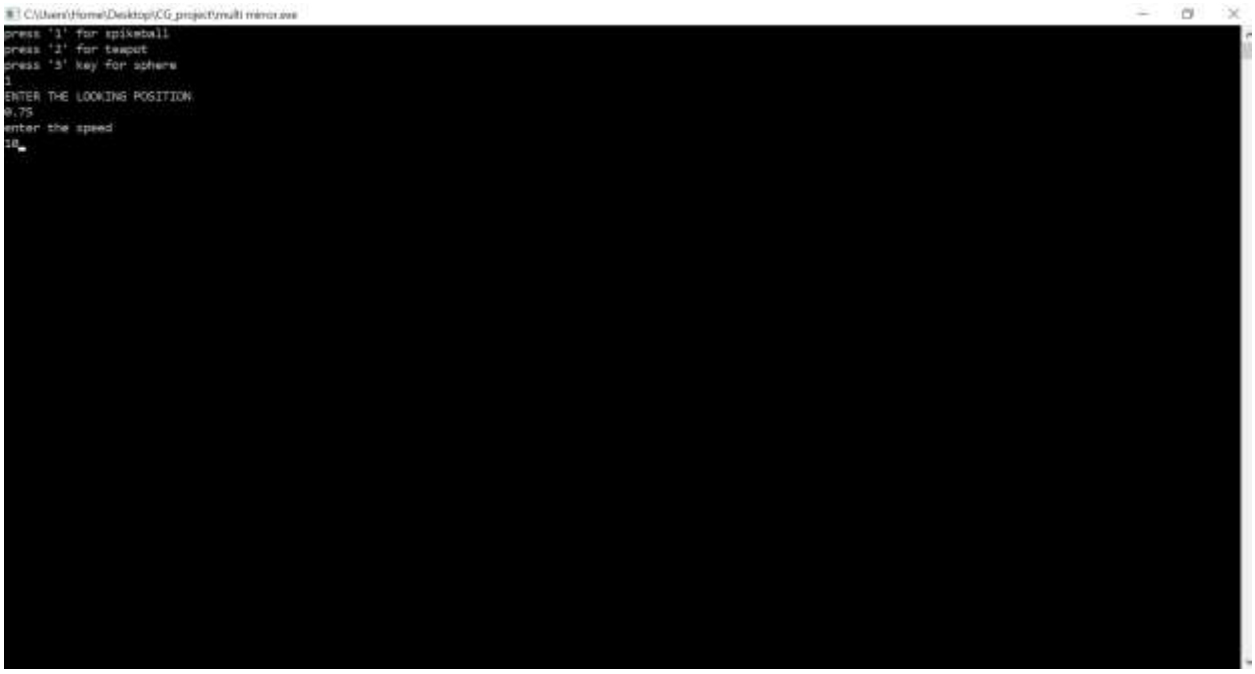
```
glColor3f(1, 0, 0);
```

Lighting and shading: The lighting and shading are applied on the objects by using the following functions.

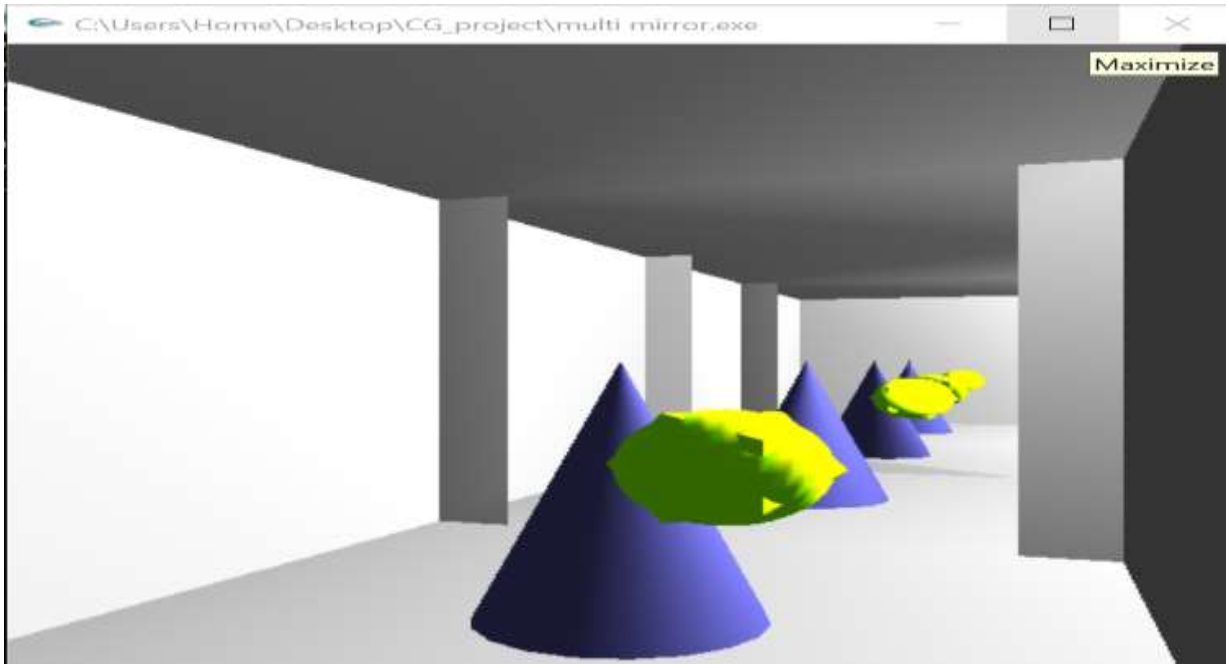
```
glEnable(GL_DEPTH_TEST);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT0, GL_POSITION, lightpos);
glEnable(GL_CULL_FACE);
glDisable(GL_LIGHTING);
```

7. SNAPSHOTS

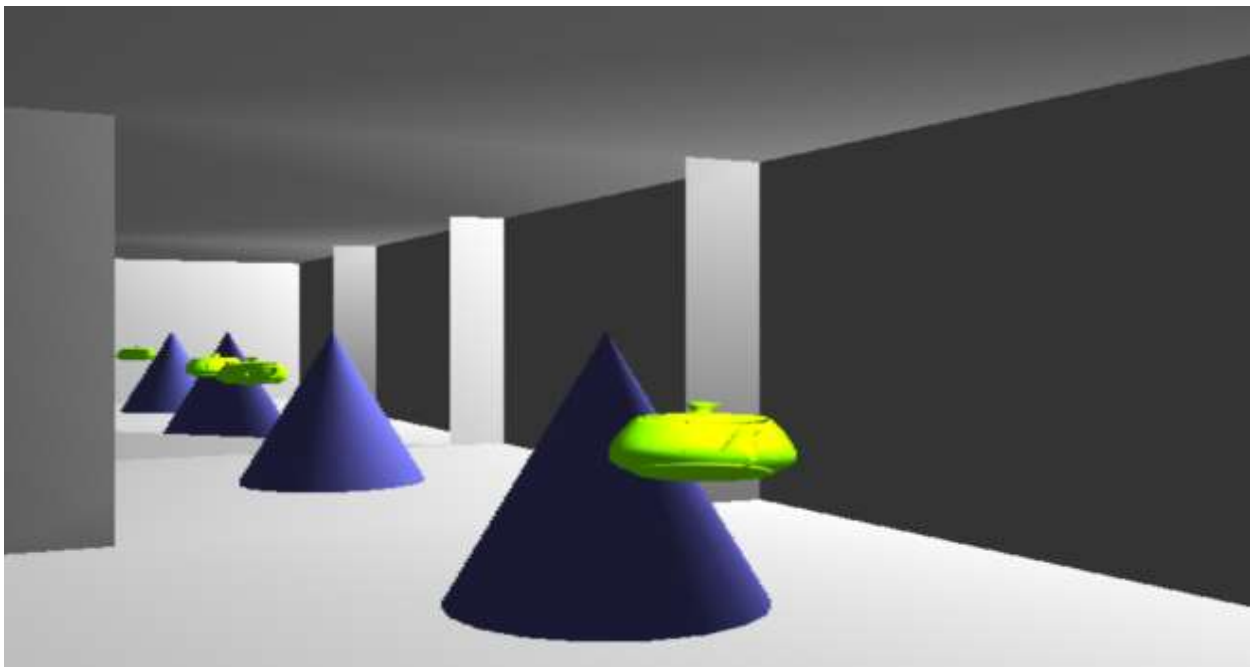
SNAPSHOT OF INPUT (Fig 7.1)



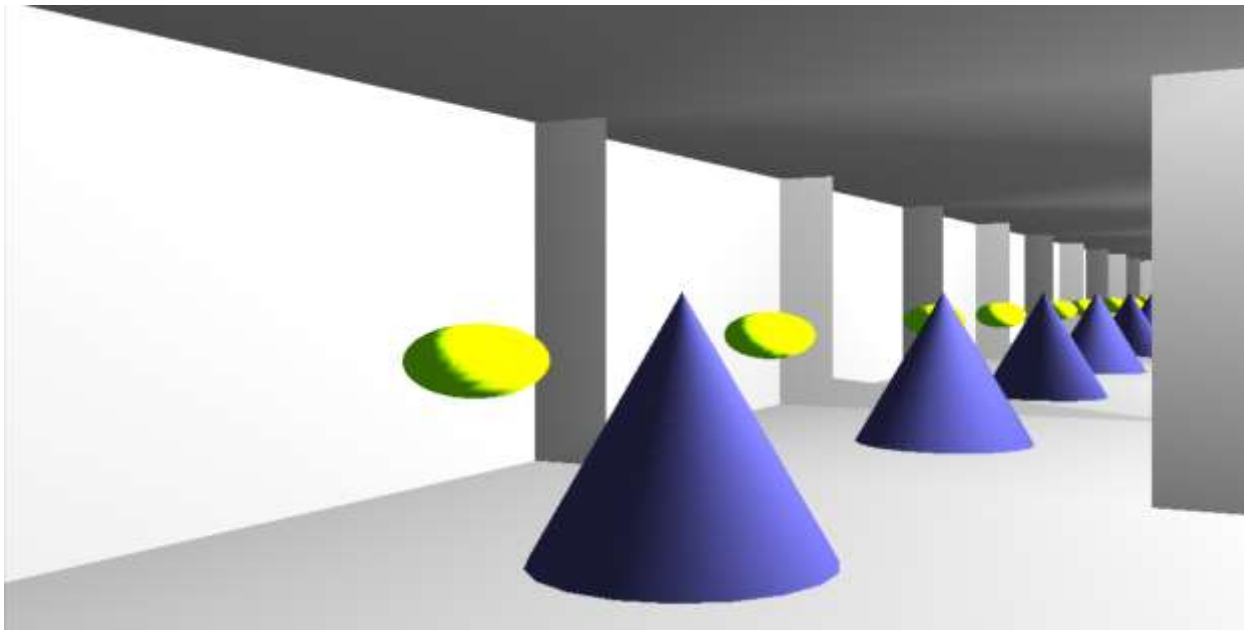
SPIKEBALL ROTATING AROUND CONE (Fig 7.2)



TEAPOT ROTATING AROUND CONE (Fig 7.2)



SPHERE ROTATING AROUND CONE (Fig 7.4)



8. CONCLUSION

Here we conclude that using OpenGL any form of images, designs and animations can be produced and also we can give realistic effects to it like in our project.

In OpenGL, we get smooth surface to draw the objects and we have provided texture and lighting effect to the objects.

We have used many built-in functions which are user friendly. We have tried to use many of the graphic concepts using OpenGL such as translation, rotation, scaling viewer motions etc.

9. REFERENCE

- Interactive Computer graphics by Edward Angel
- Opengl.org
- Stackoverflow.com
- Graphics Under C by Yashwanth Kanetkar

10. APPENDIX

a)Glut functions

1 >>> **glutDisplayFunc :-** glutDisplayFunc sets the display callback for the current window.

Syntax :- void glutDisplayFunc(void(*func)(void));

func:- The new display callback function.

2 >>> **glutReshapeFunc :-** glutReshapeFunc sets the reshape callback for the current window.

Syntax :- void glutReshapeFunc(void(*func)(int width,int height));

func:- The new reshape callback function.

3 >>> **glutMouseFunc :-** glutMouseFunc sets the mouse callback for the current window.

Syntax :- void glutMouseFunc(void(*func)(int button,int state,int x,int y));

func:- The new mouse callback function.

4 >>> **glutKeyboardFunc :-** glutKeyboardFunc sets the keyboard callback for the current window.

Syntax :- void glutKeyboardFunc(void(*func)(unsigned char key,int x,int y));

func:- The new Keyboard callback function.

5 >>> **glutCreateWindow :-** glutCreateWindow creates a top-level window.

Syntax :- int glutCreateWindow(char *name);

name:- ASCII character string for use as window name.

6 >>> **glutInit :-** glutInit is used to initalize the GLUT library.

Syntax :- void glutInit(int *argc,char **argv);

argc:- A pointer to the program's unmodified argc variable from main.Upon return,the value pointed to by argc will be updated because glutInit extracts any command line options intented for the GLUT library.

argv:- The program's unmodified argv variable from main.Like argc,the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

7 >>> **glutInitWindowPosition ,glutInitWindowSize :-** glutInitWindowPosition and glutInitWindowSize set the initial window position and size respectively.

8 >>> **glutInitDisplayMode :-** glutInitDisplayMode sets the initial display mode.

Syntax :- void glutinitDisplayMode(unsigned int mode);

mode:- Display mode,normally the bitwise OR-ing of GLUT display mode bit masks.

9 >>> **glutMainLoop :-** glutMainLoop enters the GLUT event processing loop.

Syntax :- void glutMainLoop(void);

10 >>> **glutPostRedisplay :-** glutPostRedisplay marks the current window as rendering to be redisplayed.

Syntax :- void glutPostRedisplay(void);

11 >>> **glutBitmapCharacter :-** glutBitmapCharacter renders a bitmap character using OpenGL.

Syntax :- void glutBitmapCharacter(void *font,int charcter);

12 >>> **glEnable :-**

Syntax :- void glEnable(GLenum cap);

cap:- specifies a symbolic constant indicating a GL capability.

13 >>> **glBegin and glEnd :-** The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives.

Syntax :- void glBegin(GLenum mode);

mode:- The primitives that will be created from vertices presented between glBegin and the subsequent glEnd.

14 >>> **glClear :-** The glClear function clears buffers to present values.

Syntax :- void glClear(GLbitfield mask);

mask:- Bitwise OR operators of masks that indicate the buffers to be cleared.

15 >>> **glClearColor :-** The glClearColor unction specifies clear values for the color buffers.

Syntax :- void glClearColor(red,green,blue,aplha);

red:- The red value that glClear uses to clear the color buffers.The default value is zero;

green:- The green value that glClear uses to clear the color buffers.The default value is zero;

blue:- The blue value that glClear uses to clear the color buffers.The default value is zero;

alpha:- The alpha value that glClear uses to clear the color buffers.The default value is zero;

16 >>> **glColor3i :-** Sets the color.

Syntax :- void glColor3i(GLint red,GLint green,GLint blue);

red:- The new red value for the current value .

green:- The new green value for the current value .

blue:- The new blue value for the current value .

17 >>> **glColor3fv :-** Sets the current color from an already existing array of colors.

Syntax :- void glColor3fv(const GLfloat *v);

v:- A pointer to an array that contains red,green,blue values.

18 >>> **glFlush :-** The glFlush function forces execution of OpenGL functions in finite time.

Syntax :- void glFlush(void);

This function has no parameters.

19 >>> glLoadIdentity :- The glLoadIdentity function replaces the current matrix with the identity matrix.

20 >>> glOrtho :- The glOrtho function multiplies the current matrix by an orthographic matrix.

Syntax :- void glOrtho(GLDouble left, GLDouble right, GLDouble bottom, GLDouble top, GLDouble zNear, GLDouble zFar);

left:- The coordinates of the left vertical clipping plane.

right:- The coordinates of the right vertical clipping plane.

bottom:- The coordinates of the bottom horizontal clipping plane.

top:- The coordinates of the top horizontal clipping plane.

zNear:- The distances to the nearer depth clipping plane.This distance is negative if the plane is to be behind the viewer.

zFar:- The distances to the farther depth clipping plane.This distance is negative if the plane is to be behind the viewer.

21 >>> glPointSize :- The glPointSize function specifies the diameter of rasterized points.

Syntax :- void glPointSize(GLfloat size);

Size:-The diameter of rasterized points.The default is 1.0.

22 >>> glPushMatrix and glPopMatrix :- The glPushMatrix and glPopMatrix functions push and pop the current matrix stack.

Syntax :- void WINAPI glPopMatrix(void);

23 >>> glRotatef :- The glRotatef function multiplies the current matrix by a rotation matrix.

Syntax :- void glRotatef(GLfloat angle,GLfloat x,GLfloat y,GLfloat z);

angle:-The angle of rotation.

x:-The x coordinate of vector.

y:-The y coordinate of vector.

z:-The z coordinate of vector.

24 >>> glScalef :- The glScaled and glScalef functions multiplies the current matrix by a general scaling matrix.

Syntax :- void glScalef(GLfloat x,GLfloat y,GLfloat z);

x:-Scale factors along x axis.

y:-Scale factors along y axis.

z:-Scale factors along z axis.

25 >>> glTranslatef :- The glTranslatef function multiplies the current matrix by a translation matrix.

Syntax :- void glTranslatef(GLfloat x,GLfloat y,GLfloat z);

x:- the x coordinate of a translation vector.

y:- the y coordinate of a translation vector.

z:- the z coordinate of a translation vector.

26 >>> **glVertex2d :-** Specifies a vector.

Syntax :- void glVertex2d(GLdouble x,GLdouble y);

x:- Specifies the x-coordinate of a vertex.

y:- Specifies the y-coordinate of a vertex.

b). SOURCE CODE

```
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

int object;
float a,b;
float c,d;
GLUquadricObj *cone, *base, *qsphere;

int draw_passes = 8;

int headsUp = 0;

typedef struct
{
  GLfloat verts[4][3];
  GLfloat scale[3];
  GLfloat trans[3];
} Mirror;
```

```
Mirror mirrors[] = {
  /* mirror on the left wall */
  {{{-1., -.75, -.75}, {-1., 1.0, -.75}, {-1., 1.0, .75}, {-1, -1.0, .75}},
     {-1, 1, 1}, {2, 0, 0}},

  /* mirror on right wall */
  {{{1., -1.0, .75}, {1., 1.0, .75}, {1., 1.0, -.75}, {1., -1.0, -.75}},
     {-1, 1, 1}, {-2, 0, 0}},
};
int nMirrors = 2;

void init(void)
{
  static GLfloat lightpos[] = {0.5, .75, 1.5, 1};

  glEnable(GL_DEPTH_TEST);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_POSITION, lightpos);

  glEnable(GL_CULL_FACE);

  glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
```

```
 cone = gluNewQuadric();
 qsphere = gluNewQuadric();
}

void make_viewpoint(void)
{
 if (headsUp)
{
float width = (1 + 2*(draw_passes/nMirrors)) * 1.25;
                    float height = (width / tan((30./360.) * (2.*M_PI))) + 1;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60, 1, height - 3, height + 3);
gluLookAt(0, height, 0,0, 0, 0,0, 0, 1);
          }
else
{
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60, 1, 0.5, 4 + 2*(draw_passes / nMirrors));

   gluLookAt(-2, 0,a,0, 0, 0,0, 1, 0);
 }

 glMatrixMode(GL_MODELVIEW);
```

```
  glLoadIdentity();
}

void reshape(GLsizei w, GLsizei h)
{
  glViewport(0, 0, w, h);
  make_viewpoint();
}

void draw_room(void)
{
  /* material for the walls, floor, ceiling */
  static GLfloat wall_mat[] = {1.f, 1.f, 1.f, 1.f};

  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, wall_mat);

  glBegin(GL_QUADS);

  /* floor */
  glNormal3f(0, 1, 0);
  glVertex3f(-1, -1, 1);
  glVertex3f(1, -1, 1);
  glVertex3f(1, -1, -1);
  glVertex3f(-1, -1, -1);
```

```
/* ceiling */
glNormal3f(0, -1, 0);
glVertex3f(-1, 1, -1);
glVertex3f(1, 1, -1);
glVertex3f(1, 1, 1);
glVertex3f(-1, 1, 1);

/* left wall */
glNormal3f(1, 0, 0);
glVertex3f(-1, -1, -1);
glVertex3f(-1, 1, -1);
glVertex3f(-1, 1, 1);
glVertex3f(-1, -1, 1);

/* right wall */
glNormal3f(-1, 0, 0);
glVertex3f(1, -1, 1);
glVertex3f(1, 1, 1);
glVertex3f(1, 1, -1);
glVertex3f(1, -1, -1);

/* far wall */
glNormal3f(0, 0, 1);
glVertex3f(-1, -1, -1);
glVertex3f(1, -1, -1);
```

```
  glVertex3f(1, 1, -1);
  glVertex3f(-1, 1, -1);

  /* back wall */
  glNormal3f(0, 0, -1);
  glVertex3f(-1, 1, 1);
  glVertex3f(1, 1, 1);
  glVertex3f(1, -1, 1);
  glVertex3f(-1, -1, 1);
  glEnd();
}

void draw_cone(void)
{
    static GLfloat cone_mat[] = {.5, .5f, 1.f, 1.f};

  glPushMatrix();
  glTranslatef(0, -1, 0);
  glRotatef(-90, 1, 0, 0);

  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, cone_mat);


  glutSolidCone(.3,1,19,18);
  glPopMatrix();
```

```
}

void draw_sphere(GLdouble secs)
{int key;
 static GLfloat sphere_mat[] = {1.0f, 2.f, 0.f, 0.f};
 GLfloat angle;

 /*Revolution */
  secs = secs - b*trunc(secs / b);
  angle = (secs/b) * (360.);

  glPushMatrix();
  glTranslatef(0, -.3, 0);
  glRotatef(angle, 0, 1, 0);
  glTranslatef(.6, 0, 0);

  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, sphere_mat);

   glScalef(0.18,0.18,0.18);
if(object==1)
glutSolidIcosahedron();
if(object==2)
        glutSolidTeapot(.7);
else
        glutSolidSphere(.9,25,27);
```

```
glPopMatrix();
}

GLdouble get_secs(void)
{
  return glutGet(GLUT_ELAPSED_TIME) / 1000.0;
}

void draw_mirror(Mirror *m)
{
  glBegin(GL_QUADS);
  glVertex3fv(m->verts[0]);
  glVertex3fv(m->verts[1]);
  glVertex3fv(m->verts[2]);
  glVertex3fv(m->verts[3]);
  glEnd();
}

GLenum reflect_through_mirror(Mirror *m, GLenum cullFace)
{
  GLenum newCullFace = ((cullFace == GL_FRONT) ? GL_BACK :
GL_FRONT);

  glMatrixMode(GL_PROJECTION);
```

```
  glScalef(m->scale[0], m->scale[1], m->scale[2]);
  glTranslatef(m->trans[0], m->trans[1], m->trans[2]);
  glMatrixMode(GL_MODELVIEW);

  /* must flip the cull face since reflection reverses the orientation
   * of the polygons */
  glCullFace(newCullFace);

  return newCullFace;
}

void undo_reflect_through_mirror(Mirror *m, GLenum cullFace)
{
  glMatrixMode(GL_PROJECTION);
  glTranslatef(-m->trans[0], -m->trans[1], -m->trans[2]);
  glScalef(1./m->scale[0], 1./m->scale[1], 1./m->scale[2]);
  glMatrixMode(GL_MODELVIEW);

  glCullFace(cullFace);
}

void draw_scene(GLdouble secs, int passes, GLenum cullFace,GLuint stencilVal,
GLuint mirror)
{
  GLenum newCullFace;
```

```
  int passesPerMirror, passesPerMirrorRem;
  unsigned int curMirror, drawMirrors;
  int i;

  /* one pass to draw the real scene */
  passes--;

  /* only draw in my designated locations */
  glStencilFunc(GL_EQUAL, stencilVal, 0xffffffff);

  /* draw things which may obscure the mirrors first */
  draw_sphere(secs);
  draw_cone();
  if (mirror != 0xffffffff)
{
    passesPerMirror = passes / (nMirrors - 1);
    passesPerMirrorRem = passes % (nMirrors - 1);
    if (passes > nMirrors - 1) drawMirrors = nMirrors - 1;
    else drawMirrors = passes;
  }
else
{
    /* mirror == -1 means that this is the initial scene (there was no
     * mirror) */
    passesPerMirror = passes / nMirrors;
```

```
    passesPerMirrorRem = passes % nMirrors;
    if (passes > nMirrors) drawMirrors = nMirrors;
    else drawMirrors = passes;
  }
  for (i = 0; drawMirrors > 0; i++)
{
    curMirror = i % nMirrors;
    if (curMirror == mirror) continue;
    drawMirrors--;

    /* draw mirror into stencil buffer but not color or depth buffers */
    glColorMask(0,0,0,0);
    glDepthMask(0);
    glStencilOp(GL_KEEP, GL_KEEP, GL_INCR);
    draw_mirror(&mirrors[curMirror]);
    glColorMask(1, 1, 1, 1);
    glDepthMask(1);
    glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

    /* draw reflected scene */
    newCullFace = reflect_through_mirror(&mirrors[curMirror], cullFace);
    if (passesPerMirrorRem) {
      draw_scene(secs, passesPerMirror + 1, newCullFace, stencilVal + 1,
                    curMirror);
      passesPerMirrorRem--;
```

```
    } else {
      draw_scene(secs, passesPerMirror, newCullFace, stencilVal + 1,
                 curMirror);
    }
    undo_reflect_through_mirror(&mirrors[curMirror], cullFace);

    /* back to our stencil value */
    glStencilFunc(GL_EQUAL, stencilVal, 0xffffffff);
  }

  draw_room();
}

void draw(void)
{
  GLenum err;
  GLfloat secs = get_secs();

  glDisable(GL_STENCIL_TEST);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |
GL_STENCIL_BUFFER_BIT);

  if (!headsUp) glEnable(GL_STENCIL_TEST);

  draw_scene(secs, draw_passes, GL_BACK, 0, (unsigned)-1);
```

```
  glDisable(GL_STENCIL_TEST);

  if (headsUp) {
    /* draw a red floor on the original scene */
    glDisable(GL_LIGHTING);
    glBegin(GL_QUADS);
    glColor3f(1, 0, 0);
    glVertex3f(-1, -.95, 1);
    glVertex3f(1, -.95, 1);
    glVertex3f(1, -.95, -1);
    glVertex3f(-1, -.95, -1);
    glEnd();
    glEnable(GL_LIGHTING);
  }

  err = glGetError();
  if (err != GL_NO_ERROR) printf("Error:  %s\n", gluErrorString(err));

  glutSwapBuffers();
}

/* ARGSUSED1 */
void key(unsigned char key, int x, int y)
{
  switch(key) {
```

```
    case '>':  case '+':
    draw_passes++;
    printf("Passes = %d\n", draw_passes);
    make_viewpoint();
    break;
   case '<':  case '-':
    draw_passes--;
    if (draw_passes < 1) draw_passes = 1;
    printf("Passes = %d\n", draw_passes);
    make_viewpoint();
    break;
  case 'h':  case 'H':
    /* heads up mode */
    headsUp = (headsUp == 0);
    make_viewpoint();
    break;
  case 27:
    exit(0);
  }
}

#define MIN_COLOR_BITS 4
#define MIN_DEPTH_BITS 8

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

```
{
printf("press '1' for spikeball\npress '2' for teapot\npress '3' key for sphere\n");
scanf("%d",&object);
printf("ENTER THE LOOKING POSITION  \n");
scanf("%f",&a);
printf("enter the speed \n");
scanf("%f",&b);
 d = 1.0 - 0.0;
 c = ((float) rand()/RAND_MAX)*d+0.0;
glutInit(&argc, argv);
glutInitWindowSize(500,500);
glutInitWindowPosition(0, 0);
   if (argc > 1) {
     glutInitDisplayString("samples stencil>=3 rgb depth");
   } else {
     glutInitDisplayString("samples stencil>=3 rgb double depth");
   }
   glutCreateWindow(argv[0]);
   glutDisplayFunc(draw);
   glutIdleFunc(draw);
   glutKeyboardFunc(key);
   glutReshapeFunc(reshape);
   init();

   glutMainLoop();
```

```
    return 0;
}
```