# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"JNANA SANGAMA", BELGAUM - 590018**

**2020-21**

**COMPUTER GRAPHICS AND VISUALIZATION[18CSL67]**

**Mini Project Report**

**On**

# " MOVABLE CAMERA AND THE OBJECT"

**SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT IN 6TH SEMESTER CG LABORATORY WITH MINI PROJECT WORK (18CSL67) OF BACHELOR OF ENGINEERING**

**IN**

## COMPUTER SCIENCE AND ENGINEERING

**By**


**SOWMYA D** **4VM20CS051**


**UNDER THE GUIDANCE OF**


**Mr. Madhusudhan G K**
**Assistant Professor**
**Dept. of ISE, VVIET**


ESTD : 1991
VIDYA VIKAS
EDUCATIONAL TRUST®


**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**VIDYA VIKAS INSTITUTE OF ENGINEERING & TECHNOLOGY**

#127-128, Mysore - Bannur Road, Alanahally, Mysuru, Karnataka 570028

Vidya Vikas Educational Trust ®

# VIDYA VIKAS INSTITUTE OF ENGINEERING

# & TECHNOLOGY

#127-128, Mysore - Bannur Road, Alanahally, Mysuru, Karnataka 570028

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## CERTIFICATE

Certified that the **COMPUTER GRAPHICS AND VISUALIZATION** Mini Project Work entitled "**MOVABLE CAMERA AND THE OBJECT**" carried out by SOWMYA D [4VM20CS051] ,bonafide students of VVIET in partial fulfillment for the award of degree Bachelor of Engineering in **COMPUTER SCIENCE and ENGINEERING** as prescribed by **VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM** during the academic year 2022-23. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for the said degree.

| | |
|---|---|
| **Mr. Madhusudhan G K** | **Dr. MADHU B K** |
| **Assistant Proffessior** | **Proffessior&Head** |
| **Dept. of CSE** | **Dept. of CSE** |

## EXTERNAL VIVA

| **Name of the examinars** | **Signature with date** |
|---|---|
| **1.**_____ | _____ |
| **2.**_____ | _____ |

# ACKNOWLEDGEMENT

We would like to thank and express our heartfelt gratitude to God almighty for the abundant blessings without which this project would not have been successful.

We would like to express our sincere gratitude to **Sri. Vasu,** Chairman of VVIET, **Mr. Kaveesh Gowda V**, Secretary of VVIET and all management members of VVIET, for their constant support.

We acknowledge and express our sincere thanks to our beloved Principal **Dr. Manjunath,** VVIET, Mysuru who is the source of inspiration.

We would like to express our deepest sense of gratitude towards **Dr. Madhu B K**, Head of the Department, CSE, VVIET, Mysuru for his valuable suggestions, support and encouragement.

We would like to extend our heartfelt gratitude to **Mr. Madhushudan G K,** Assistant Professor, Dept. of CSE, for the valuable guidance and advice. We would also like to thank him for his guidance and useful suggestions, which helped us in completing the project work on time.

We would also thank all other teaching and non-teaching staffs of the Computer Science Department who has directly or indirectly helped us in completion of this project.

Our thanks and appreciation also go to our family and friends who have willingly helped us out with their abilities.

**Regards,**

SOWMYA D

4VM20CS051

# ABSTRACT

In our project we are going to demonstrate the moving of a “MOVABLE CAMERA AND THE OBJECT” viewer can see a cameras moving. We have mainly used the concept of translation. We have developed this project using OpenGL. It is a software interface to graphic hardware. The OpenGL utility library (GLU) provides many of the modelling features. GLU isa standard part of every OpenGL Implementation.

# TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

## 1.1 What is OpenGL?

OpenGL provides a set of commands to render a three dimensional scene. That means youR provide the data in an OpenGL-usable form and OpenGL will show this data onthe screen. It is developed by many companies and it is free to use. You can developOpenGL-applications without licensing.

OpenGL is a hardware-independent and system-independent interface. An OpenGL-application will work on every platform as long as there is an installed implementation.

As OpenGL is system independent there are no functions to create windows etc.,but there helper functions for each platform. A very useful thing is GLUT.Most of our applications can be accessed in OpenGL using three libraries:-

- GL-(or OpenGL in Windows), has names that begin with GL andare stored in a library.
- GLU-(or OpenGL Utility Library), uses only GL functions butcontains code for creating
  common objects and viewing.
- GLUT-(or OpenGL Utility Toolkit), provides the minimumfunctionality that is expected in modern windowing system.

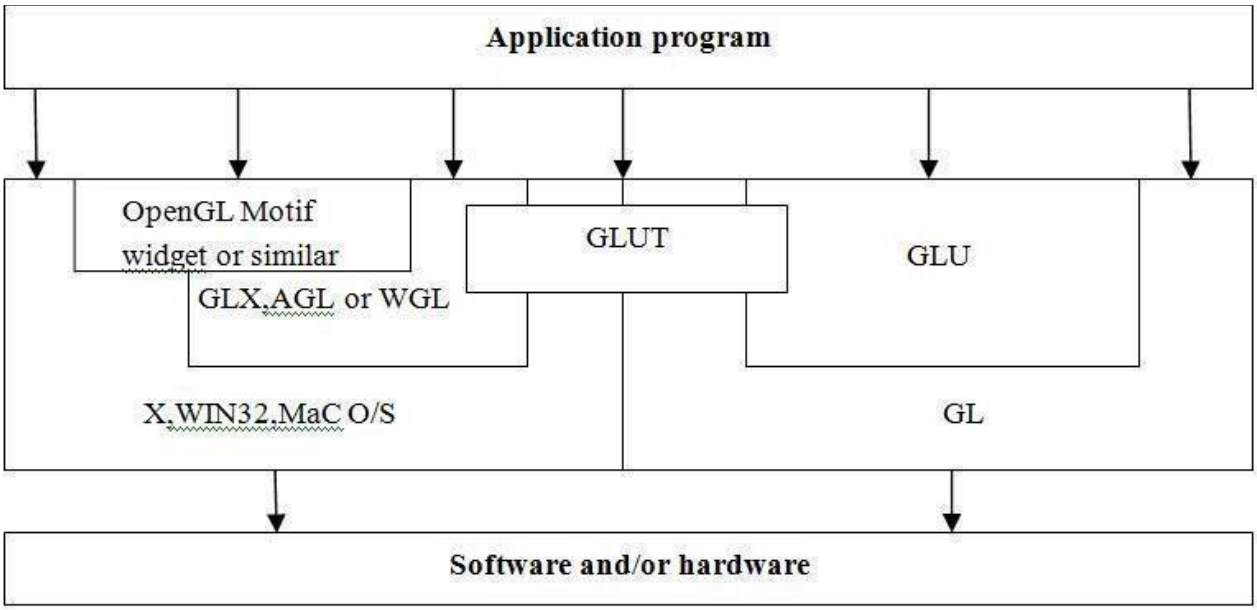



**Figure 1.1 Block diagram of OpenGL utility toolkit**

OpenGL is based on the state variables. There are many values, for example the color, that remain after being specified. That means, you can specify a color once and draw several polygons, lines or whatever with this color then. There are no classes like in DirectX. However, it is logically structured. Before we come to the commands themselves, here is another thing:

To be hardware independent OpenGL provides its own data types. They all begin with“GL”. For example, GLfloat, GLint and so on. There are also many symbolic constants,they all begin with “GL_” like GL_POINTS, GL_POLYGON. Finally the commands havethe prefix “gl” like glVertex3f(). There is a utility library called GLU, here the prefixes are“GLU_” and “glu”. GLUT commands begin with “glut”, it is the same for every library.

A very important thing is to know that there are two important matrices which effectthe transformation from the 3D-world to the 2D-screen: The projection matrix and themodel view matrix. The projection matrix contains information, how a vertex: let’s say a“point” in space – shall be mapped to the screen. This contains whether the projection shallbe isometric or from a perspective, how wide the field of view is and so on.Computer graphics is divided in to two broad classes:

(i) Non-interactive Graphics

(ii) Interactive Graphics

(i) **Non-interactive Graphics**: In non-interactive graphics or passive graphics the pictures produced on a screen . while in interactive graphics the pictures displayed on the screen are controlled by the user through an input device.

(ii) **Interactive Graphics**: This is the type of computer graphics in which the pictures produced can be controlled by the user. It involves two way communication between user and computer . The computer upon receiving the signal from the input device can modify the displayed picture appropriately.

## 1.2 What is GLUT?

The **OpenGL Utility Toolkit** (**GLU**) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wire-frame mode) are also provided,including cubes, spheres and the Utah teapot. GLUT also has some limited support for creating pop-up menus.

In your first line you always write glutInit(&argc,argv). After this, you must tell GLUT, which display mode you want -single or double buffering, color index mode or RGB and so on. This is done by calling glutInitDisplayMode(). The symbolic constants are connected by a logical OR, so you could use glutDisplayMode(GLUT_RGB|GLUT_SINGLE). After the initialization you can call glCreateWindow() with the window name as parameter.

Then you can pass some methods for certain events. The most important ones are “reshape” and “display”. In reshape you need to (re)define the field of view and specify a new area in the window, where OpenGL is allowed to draw to.

Display should clear the so called color buffer - let’s say this is the sheet of paper -and draw the objects.

You pass the methods by glut*Func(), for example glutDisplayFunc(). At the end of the main function glutMainLoop(). This function doesn’t return, but calls the several functions passed by glut*Func.

## 1.3 About the Project

The development of any project will improve the user’s knowledge. MOVABLECAMERA AND OBJECT is a interactive simulation created using Microsoft VisualBasic and OpenGL programming toolkit. This package shows simple applications of computergraphics in consecutive transformations applied to altered coordinate systems.

The Computer Graphics is one of the most powerful and interesting face of computers.The aim of this project is to learn about the concepts in Graphics and know standard library “Graphics Function” and also to explore some other functions.

This program enables the user to move camera and object along the X/Y/Z axis. Here wehave both keyboard and mouse interaction. When a key is pressed and mouse button is pressedthecorresponding actiontakes place.

# CHAPTER 2

# REQUIREMENT SPECIFICATIONS

Requirement specification is a document or set of documents that specifically says the requirements of the project. It is of two types i.e. Software requirements and Hardware requirements.

Software requirements deal with the software that we need to execute the project. Hardware requirements on the other hand deal with the operating system processor we have, its speed, model, manufacturer, etc.

## 2.1 HARDWARE REQUIREMENTS

Processor : Intel Processor 4.0GHz and above

RAM : 1GB DDR with 256 MHZ

Hard disk : 4 GB and above

## 2.2 SOFTWARE REQUIREMENTS

Operating System : Ubuntu 14.04

Language : Vi Editor

API : OpenGL Library Toolkit

# CHAPTER 3

## SYSTEM DESIGN

Design is the planning that lays the basics for the making of every object or system. The chapter involves the designing of the various aspects and different stages of the project. When the program is made to execute, the output window is displayed first. The flow of the operation from the output window is shown in the figure.

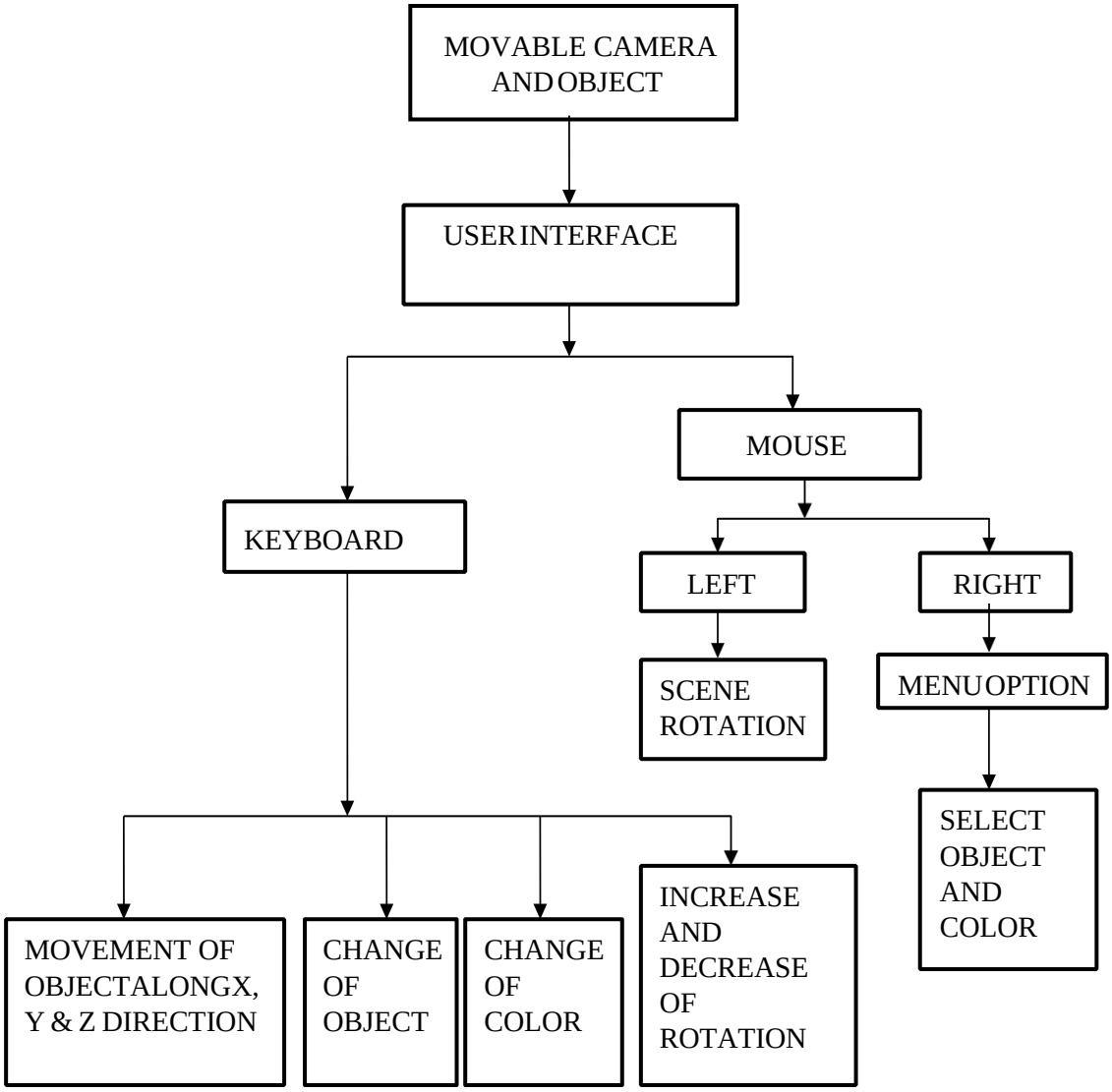



**Figure 3.1** Design of the project

# CHAPTER 4

# IMPLEMENTATION

## 4.1 FUNCTIONS

The functions that are used in the program are discussed below. This section contains description of all the headers and functions. These functions are as follows:

### 4.1.1 Headers Defined

The in-built are defined in the OpenGL library. Some of the headers that are used as follows:

- **#include <stdio.h>:** to take input from standard input and write to standard output.
- **#include <stdlib.h>**: to include standard library functions.
- **#include <GL/glut.h>**: to include glut library files.
- **#include <math.h>**: to include mathematics library files.
- **#include <unistd.h>**: to include sleep function.

### 4.1.2 Inbuilt functions

- **Void glutInit(int argc,char **argv);**

  glutInit() should be called before any other GLUT routine, because it initializes GLUT library. glutInit() will also process command line options, but the specific Options are window system dependent. For the X Window system -iconic, -geometry, and -display are examples of command line options, processed by glutInit().

- **Void glutInitDisplayMode(unsigned int mode);**

  Specifies a display mode (such as RGBA or color-index, or single- or double- buffer) for windows created when glutCreateWindow() is called. You can also specify that the window have an associated, stencil, and/or accumulation buffer. The mask argument is a bitwise ORed combination of GLUT_RGBA or GLUT_INDEX,GLUT_SINGLE or GLUT_DOUBLE and any of the buffer-enabling flags: GLUT_DEPTH, GLUT_STENCIL, or G LUT_ACCUM.

- **Void glutInitWindowPosition(int x, int y);**

  Specifies the initial position of the top-left corner of the window in pixels.

- **Void glutInitWindowSize(int width, int height);**

Specifies the initial height and width of the window in pixels.

- **Void CreateWindow(char *name);**

  Opens a window with previously set characteristics (display mode, width, height, and so on). The string name may appear in the title bar if your window system does that sort of thing. The window is not initially displayed until glutMainLoop() is entered, so do not render into the window until then.

- **Void glutKeyboardFunc(void (*func)( unsigned int key, int x, int y);**

  Specifies the function, func, that's called when a key that generates an ASCII character is pressed. The key callback parameter is the generated ASCII value. The x and y callback parameter indicate the location of the mouse (in window-relative coordinates) when the key was pressed.

- **Void glutDisplayFunc(void(*func)(void));**

  Specifies the function that's called whenever the contents of the window need to be redrawn. The contents of the window may need to be redraw when the window is initially opened, when the window is popped and window damage is exposed, and when glutRedsplay() is explicitly called.

- **Void glutPostRedisplay(void);**

  Whenever circumstances indicate that your window is in need of being redisplayed, you may call glutPostRedisplay*()* to tell OpenGLUT that you want to redraw your graphics. Multiple calls to this function may be coalesced by OpenGLUT to avoid excessive invocation of your drawing support. The ultimate effect of this function is to call your Display callback for the *current window.*

- **Void glutSwapBuffers(void);**

  Performs a buffer swap on the *layer in use* for the *current window*. Specifically, glutSwapBuffers promotes the contents of the back buffer of the *layer in use* of the *current window* to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffersis called. An implicit glFlush is done by glutSwapBuffers before it returns. Subsequent OpenGL commands can be issued immediately after calling glutSwapBuffers, but are not executed until the buffer exchange is completed. If the *layer in use* is not double buffered, glutSwapBuffers has no effect.

- **Void glutMainLoop(void);**

  Enters the GLUT processing loop, never to return. Registered callback functions will be called when the corresponding events instigate them.
- **void glutTimerFunc(unsigned int msecs, void (*func)(int value), value);**

  glutTimerFunc registers a timer callback to be triggered in a specified number of milliseconds. msecs Number of milliseconds to pass before calling the callback. func. Value is the Integer value to pass to the timer callback.
- **Int glutCreateMenu(void (*func)(int value));** func: The callback function for the menu that is called when a menu entry from the menu is selected. glutCreateMenu creates a new pop-up menu.

  Value: passed to the callback is determined by the value for the selected menu entry.
- **Void glutAddSubMenu(char *name, int menu);** name: ASCII character string to display in the menu item from which to cascade the sub- menu.

  Menu: IdentifiglutAddSubMenu adds a sub-menu trigger to the bottom of the current menu
- **Void glutAddMenuEntry(char *name, int value);** glutAddMenuEntry: adds a menu entry to the bottom of the current menu. name: ASCII character string to display in the menu entry.

  Value: Value to return to the menu's callback function if the menu entry is selected.
- **void glutAttachMenu(int button);** glutAttachMenu attaches a mouse button for the current window to the identifier of the current menu.

  button: The button to attach a menu or detach a menu.
- **Void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpa);** Specify the red, green, blue, and alpha values used when the color buffers are cleared. The default values are all zero.
- **Void glMultMatrixf(Type * m);**

  Post multiplies the current matrix by the 4x4 array obtained from the 16 element array of TYPE GLfloat or GLdouble. The **glMultMatrix** function multiplies the current matrix by the one specified in *m*. That is, if M is the current matrix and T is the matrix passed to **glMultMatrix**, then M is replaced with M • T.
- **glPushMatrix();**

  There is a stack of matrices for each of the matrix modes. In GL_MODELVIEW mode, the

stack depth is at least 32. In the other two modes, GL_PROJECTION and GL_TEXTURE, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode. The **glPushMatrix** function pushes the current matrix stack down by one, duplicating the current matrix. That is, after a **glPushMatrix** call, the matrix on the top of the stack is identical to the one below it.

- **glPopMatrix();**
  The **glPopMatrix** function pops the current matrix stack, replacing the current matrix with the one below it on the stack. Initially, each of the stacks contains one matrix, an identity matrix.
- **glLight( GLenum light, GLenum pname, const GLfloat *params);**
  The **glLightfv** function sets the value or values of individual light source parameters. The light parameter names the light and is a symbolic name of the form GL_LIGHT*i*, where $0 \leq i <$ GL_MAX_LIGHTS.
  The pname parameter specifies one of the light source parameters, again by symbolic name. The params parameter is either a single value or a pointer to an array that contains the new values. Some of the params are:

| | |
|---|---|
| GL_AMBIENT | The params parameter contains four floating-point values that specify the ambient RGBA intensity of the light. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default ambient light intensity is (0.0, 0.0, 0.0, 1.0). |
| GL_DIFFUSE | The params parameter contains four floating-point values that specify the diffuse RGBA intensity of the light. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default diffuse intensity is (0.0, 0.0, 0.0, 1.0) for all lights other than light zero. The default diffuse intensity of light zero is (1.0, 1.0, 1.0, 1.0). |

| | |
|---|---|
| GL_SPECULAR | The params parameter contains four floating-point values that specify the specular RGBA intensity of the light. Floating-point values are mapped directly. Neither integer nor floating-point values |
| | are clamped. The default specular intensity is (0.0, 0.0, 0.0, 1.0) for all lights other than light zero. The default specular intensity of light zero is (1.0, 1.0, 1.0, 1.0). |
| GL_POSITION | The params parameter contains four floating-point values that specify the position of the light in homogeneous object coordinates.<br>Both integer and floating-point values are mapped directly.<br>The position is transformed by the model view matrix<br>when glLightfv is called and it is stored in eye coordinates. Diffuse and specular lighting calculations take the light's direction, but not its actual position, into account, and attenuation is disabled. |

Lighting calculation is enabled and disabled using **glEnable** and **glDisable** with argument GL_LIGHTING. When lighting is enabled, light sources that are enabled contribute to the lighting calculation. Light source *i* is enabled and disabled using **glEnable** and **glDisable** with argument GL_LIGHT*i*.

It is always the case that GL_LIGHT*i* = GL_LIGHT0 + *i*.

- **void glEnable(GLenum cap);** glEnable enable the various capabilities. Use glIsEnabled or glGet to determine the current setting of any capability.
- **GLPerspective(rect = None, fovy = 20, near = 0.1, far = 1000)**
  Creates a GLPerspective instance with the given initial values for its projection parameters.
- **Void ViewPort(int x,int y,GLfloatsizei width,GLfloatsizei height);**
  Here *x*, *y* Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0, 0) width, height.
  Specify the width and height of the viewport. When a GL context is first attached to a window,

width and height are set to the dimensions of that window. It also specifies the affine transformation of xx and yy from normalized device coordinates to window coordinates. Let (xnd,ynd)xndynd be normalized device coordinates. Then the window coordinates (xw,yw)xwyw computed as follow xw=(xnd+1)(width÷2)+xxw; yw=(ynd+1)(height÷2)+ys;

Viewport width and height are silently clamped to a range that depends on the implementation. To query this range, call glGet with argument GL_MAX_VIEWPORT_DIMS.

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

  The first argument specifies how many degree you want to rotate around an axis. The following three argument specify the x, y, z axis to rotate, respectively. In that statement, you're rotating the y and z axis 0 degrees. That's why you don't see any change.

  Also, the last three parameters are clamped to the range of [0, 1].

- **void glTranslatef(GLfloat x,GLfloat y,GLfloat z);** Here, x-The x coordinate of a translation vector. y-The y coordinate of a translation vector. z-The z coordinate of a translation vector.

  The glTranslatef function produces the translation specified by (x, y, z).

  If the matrix mode is either GL_MODELVIEW or GL_PROJECTION, all objects drawn after glTranslatef is called are translated. Use glPushMatrix and glPopMatrix to save and restore the untranslated coordinate system.

- **void gluCylinder(GLUquadric* quad, GLdouble radius, GLint slices, GLint stacks);**

  Quad specifies the quadrics object (created with gluNewQuadric). Radius specifies the radius of the sphere. Slices Specifies the number of subdivisions around the z axis (similar to lines of longitude). Stacks specifies the number of subdivisions along the z axis (similar to lines of latitude).gluCylinder draws a cylinder of the given radius centered around the origin.

- **multLookAt()**

  Create a matrix to make an object, such as a camera, "look at" another object or location, from a specified position.

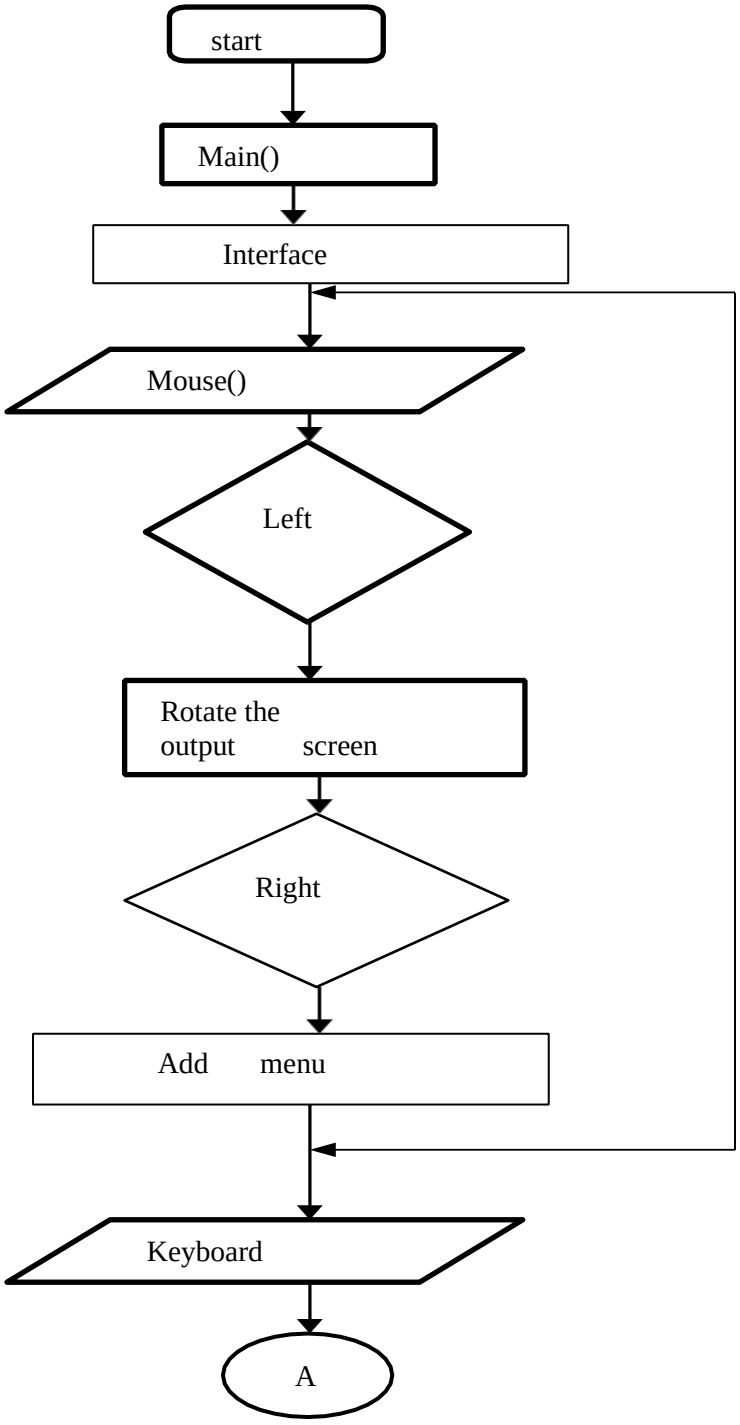
  Parameter:

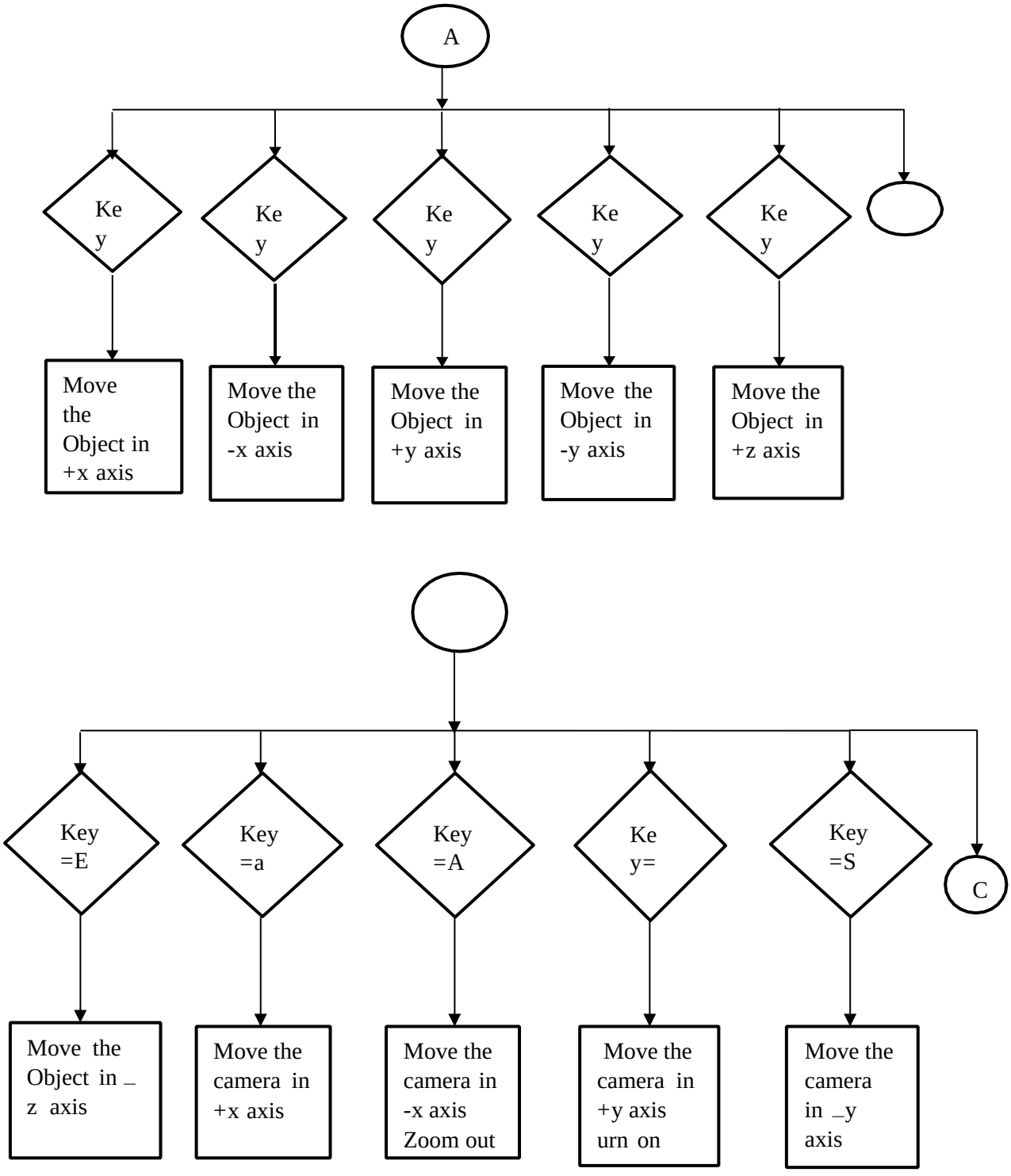
  eye[x|y|z] Desired location of the camera object

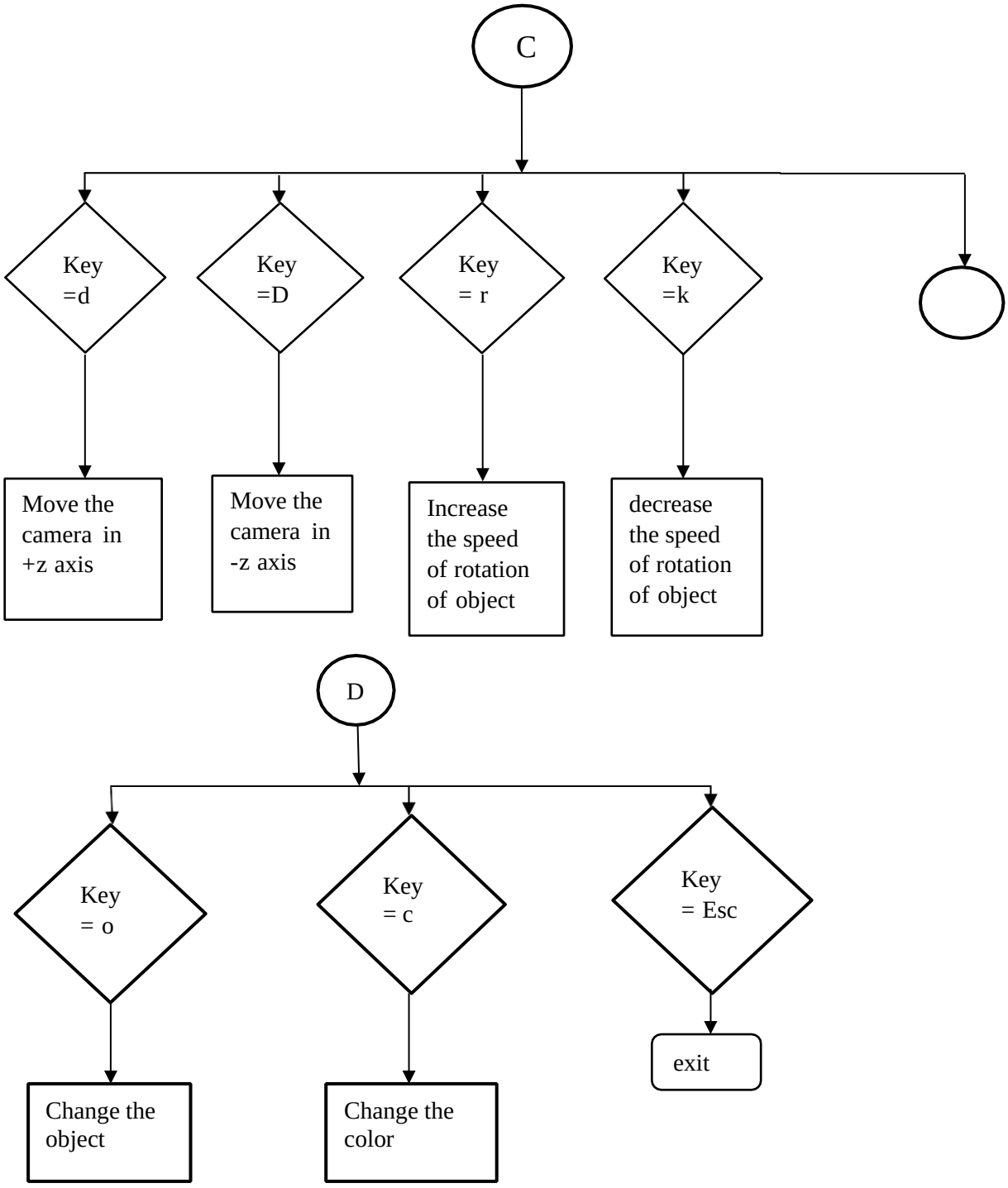
  at[x|y|z] Location for the camera to look at

  up[x|y|z] Up vector of the camera

**4.2 Flowchart**




A
Ke y
Ke y
Ke y
Ke y
Ke y
Move the Object in +x axis
Move the Object in -x axis
Move the Object in +y axis
Move the Object in -y axis
Move the Object in +z axis
Key =E
Key =a
Key =A
Ke y=
Key =S
C
Move the Object in _ z axis
Move the camera in +x axis
Move the camera in -x axis Zoom out
Move the camera in +y axis urn on
Move the camera in _y axis


C
Key =d
Key =D
Key = r
Key =k
Move the camera in +z axis
Move the camera in -z axis
Increase the speed of rotation of object
decrease the speed of rotation of object
D
Key = o
Key = c
Key = Esc
Change the object
Change the color
exit

# CHAPTER 5

# TESTING

## 5.1 Introduction to Testing

Verification and validation is a generic name given to checking processes, which ensures that the software confirms to its specification and meets the demands of users.

✦ **Validation**

Are we building the right product?

Validation involves checking that the program has implanted meets the requirement of the users.

✦ **Verification**

Are we building the product right?

Verification involves checking that the program confirms to its specification.

## 5.2 Stages in the Implementation of Testing

- **Unit testing**

  Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other components. Since this program has implemented in C++, the requirements are as per the language we used.

- **Integration Testing**

  Upon completion of unit testing , the units or modules are to be integrated which gives raise to integration testing. The purpose of integration testing is to verify the functional, performance, and reliability between the modules that are integrated.

- **System testing**

  This is concern with Sub-system testing to integrate the system requirements. This in actual includes of finding the errors and interaction with the interface problem. This software will make sure that all the errors have been debugged and no warnings. So the software can be run softly.

- **User Acceptance testing**

  The code can make sure that all the errors and omissions in system requirements definition have made since acceptance testing may reveal errors at the running time.

### 5.3 Test Cases

| TC Id | Test Case Description | Input | Expected Result | Actual Result | Remark |
|---|---|---|---|---|---|
| 1. | Screen Displaying the menu | Click the RIGHT mouse button on the display screen | Menu with • Objects • Color • Quit Should be displayed | Menu with • Object • Color • Quit Is Displayed, In Figure 6.1 | Pass |
| 2. | It Displays the objects like Sphere, Octahedron, ,Icosahedron | Click the object option in menu | Objects should be displayed | Objects displayed , In Figure 6.2 | Pass |
| 3. | It Displays the objects with one of the different colors like orange, red, blue, purple, pink | Click the colors option in menu | Object should be displayed with color | Objects with colors, In Figure 6.3 | Pass |
| 4. | Object moves in x-axis | Press the key Q or q to move the object in x-axis | Object should be moved in x-axis | Object moves in x-axis, In Figure 6.4 | Pass |

| | | | | | |
|---|---|---|---|---|---|
| 5. | Object moves in y-axis | Press the key W or w to move the object in y- axis | Object should be moved in y-axis | Object moves in y-axis, In Figure 6.5 | Pass |
| **6.** | Object moves in z-axis | Press the key E or e to move object in z- axis | Object should be moved in z-axis | Object moves in z-axis, In Figure 6.6 | Pass |
| 7. | Camera moves in x-axis | Press the key A or a to move Camera in x-axis | Camera should be moved in x-axis | Camera moves in x-axis, In Figure 6.7 | Pass |
| 8. | Camera moves in y-axis | Press the key S or s to move Camera in y-axis | Camera should be moved in y-axis | Camera moves in y-axis, In Figure 6.8 | Pass |
| 9. | Camera moves in z-axis | Press the key D or d to move Camera in z-axis | Camera should be moved in z-axis | Camera moves in z-axis, In Figure 6.9 | Pass |
| 10. | Screen will rotate | Click the RIGHT mouse button | Screen should be rotated | Screen rotate , In Figure 6.10 | Pass |

| | | | | | |
|---|---|---|---|---|---|
| 11. | Quit the screen | Click the Quit option in menu | Screen exit | Screen exit | Pass |

**Table 5.1** Test Cases

# CHAPTER 6

## SOURCE CODE

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include<stdarg.h>
#include <GL/glut.h>
#include <GL/glu.h>
GLUquadric  *quadCyl;
int windowWidth;
// Rotate the scene about the Y axis
GLfloat yrot;
// Location of the target and camera objects, respectively
float target[3], camera[3];
#define DELTA 0.25f
#define TIMER 33
void *font = GLUT_STROKE_ROMAN;
void *fonts[] = {GLUT_STROKE_ROMAN};
static float xrot;
int value,val,valu,v;
void polygon();
int theObject = -1;
char
*objectNames[10] ={"Sphere","Octahedron","Tetrahedron","Icosahedron"};
void setup()
{
glClearColor(.4, .5, .6, .0);
}
```

```
void output(GLfloat x, GLfloat y, char *format,...)
{
va_list args;
char buffer[500], *p;
if(theObject>=0)
{
va_start(args, format);
vsprintf(buffer, format, args);
va_end(args);
glPushMatrix();
glTranslatef(x, y, 0);
for (p = buffer; *p; p++)
glutStrokeCharacter(font, *p);
glPopMatrix();
}
}
void text()
{
glColor3f(1.0,1.0,0.0);
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
gluOrtho2D(0, 3000, 0, 3000);
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadIdentity();
glColor3f(.0,.0,1.0);
output(1800,2800, "%s",objectNames[theObject]);
glPopMatrix();
glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
```

```
glPopAttrib();
}
void Colors()
{
switch(val)
{
case 3:
glColor3f(1.0,.5,.0);
break;
case 4:
glColor3f(1.0,.0,0.0);
break;
case 5:
glColor3f(0.0,.0,1.0);
break;
case 6:
glColor3f(.5,.0,.5);
break;
case 7:
glColor3f(1.0,.0,0.5);
break;
default: glColor3f(.6, .3, .5);
break;
}
}
void col(int p)
{
val=p;
}
void drawScene (GLenum order)
{
GLfloat pos[4] = {-2.5, 5., 2.2, 1.};
```

```
    glLightfv (GL_LIGHT1, GL_POSITION, pos);
    glPushMatrix();
    glEnable (GL_CULL_FACE);
    glCullFace (GL_BACK);
    glFrontFace (order);
    text();
    /* Draw the polygon */
    glRotatef(xrot, 1.,1., 1.);
    Colors();
    glPushMatrix ();
    glPopMatrix ();
    polygon();
    glPopMatrix();
    }
    void sphere()
    {
    theObject=0;
    glutSolidSphere(0.5,28,12);
    }
    void oct()
    {
  theObject=1;
    glutSolidOctahedron();
    }
    void tet()
    {
  theObject = 2;
    glutSolidTetrahedron();
    }
    void ico()
    {
    theObject = 3;
```

```
    glutSolidIcosahedron();
    }
    void polygon()
    {
    if(value==14)
    {
    sphere();
    }

    if(value==15)
  oct();
  if(value==16)
  tet();
  if(value==17)
  {
  ico();
 }
   }
    void pol(int p)
    {
    value=p;
    }
    void main_menu(int valu)
    {
    value=valu;
    if (value == 0)
    exit(0);
    }
    static void key (unsigned char key, int x, int y)
    {
    int nv=value,cv=val;
    if(key=='o' && nv>=14 && nv<=17)  value++;
```

```
if(key=='o' && nv==17)  value=14;
if(key=='c' && cv>=3 && cv<=7)  val++;
if(key=='c' && cv==7)  val=3;
if(key=='r' && v<16)  v++;
if(key=='k' &&v>=1)  v--;
if(key=='q') target[0] -= DELTA;
if(key=='Q') target[0] += DELTA;
if(key=='w') target[1] -= DELTA;
if(key=='W') target[1] += DELTA;
if(key=='e') target[2] -= DELTA;
if(key=='E') target[2] += DELTA;
if(key=='a') camera[0] -= DELTA;
if(key=='A') camera[0] += DELTA;
if(key=='s') camera[1] -= DELTA;
if(key=='S') camera[1] += DELTA;
if(key=='d') camera[2] -= DELTA;
if(key=='D') camera[2] += DELTA;
glutPostRedisplay ();
}
static void cross(float dst[3],float srcA[3],float srcB[3])
{
dst[0] = srcA[1]*srcB[2] - srcA[2]*srcB[1];
dst[1] = srcA[2]*srcB[0] - srcA[0]*srcB[2];
dst[2] = srcA[0]*srcB[1] - srcA[1]*srcB[0];
}
static void normalize (float vec[3])
{
const float squaredLen = vec[0]*vec[0] + vec[1]*vec[1]
+ vec[2]*vec[2];
const float invLen = 1.f / (float) sqrt (squaredLen);
vec[0] *= invLen;
vec[1] *= invLen;
```

```
vec[2] *= invLen;
}
static void scale (float v[3], float s)
{
v[0] *= s;
v[1] *= s;
v[2] *= s;
}
/* multLookAt -- Create a matrix to make an object, such as
a camera, "look at" another object or location, from
a specified position.
glMatrixMode (GL_MODELVIEW)
// Define the usual view transformation here using
//   gluLookAt or whatever
glPushMatrix();
multLookAt (orig[0], orig[1],orig[2],
at[0],at[1],at[2],up[0], up[1], up[2]);
// Define "camera" object geometry here
glPopMatrix();
Warning: Results become undefined as (at-eye) approaches coincidence with (up).*/

static void multLookAt (float eyex, float eyey, float eyez,
float atx, float aty, float atz,
float upx, float upy, float upz)
{
float m[16];
float *xaxis = &m[0],
*up = &m[4],
*at = &m[8];
// Compute our new look at vector, which will be the new negative Z axis of our
transformed object.
 at[0] = atx-eyex;
```

```
    at[1] = aty-eyey;
    at[2] = atz-eyez;
    normalize (at);
    up[0] = upx;
    up[1] = upy;
    up[2] = upz;

    //Cross product of the new look at vector and the current up vector will produce a vector
//which is the new positive X axis of our transformed object.

    cross (xaxis, at, up);
    normalize (xaxis);
    cross (up, xaxis, at);
    scale (at, -1.f);
    m[3] = 0.f;     // xaxis is m[0..2]
    m[7] = 0.f;     // up is m[4..6]
    m[11]= 0.f;    // -at is m[8..10]
    m[12]= eyex;
    m[13]= eyey;
    m[14]= eyez;
    m[15]= 1.f;
glMultMatrixf (m);
    }
 static void display( void )
    {
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity ();

    // View transform, place 'eye' at (0,1,5), and allow the scene
  glTranslatef (0., -1., -5.);
```

```
    //to rotate around its local Y axis in front of the eye.
     glRotatef (yrot, 0., 1., 0.);
      //To Draw the target object
    //glColor3f (.7,.4,.1);
    glPushMatrix ();
    glTranslatef (target[0], target[1], target[2]);
    /* Draw the real scene */
drawScene(GL_CCW);
glPopMatrix ();
    glPushMatrix ();

    //Create transform so our next set of geometry will"look at" the target.
multLookAt (camera[0], camera[1], camera[2],
    target[0], target[1], target[2],
    0., 1., 0.);
    glRotatef (-90., 1., 0., 0.);
    glColor3f (0.2, .2, .2);
    glutSolidCube(.45);
    glRotatef (-90., 1., 0., 0.);
    glColor3f (1., .6, .4);
    gluCylinder (quadCyl, .20, 0., 0.6, 8, 2);
    glPopMatrix ();

    glutSwapBuffers();
    }
    void selectFont(int newfont)
 {
    font = fonts[newfont];
    glutPostRedisplay();
    }
    void reshape(int w, int h)
    {
```

```
    windowWidth=w;
    glViewport (0, 0, w, h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluPerspective (50., (float)w/(float)h, 1., 20.);
    }
    static void timer (int value)
    {
    xrot +=v;
    if (xrot > 360.f) xrot -= 360.f;
    glutPostRedisplay ();
    glutTimerFunc (TIMER, timer, 0);
    }
  static void motion (int x, int y)
    {
    yrot = (float)x*360.f/(float)windowWidth - 180.f;
    glutPostRedisplay ();
    }
 static void init ()
    {
    int s1,s2,s3;
    xrot = 1.;
    target[0] = target[1] = target[2] = 1.25f;
    camera[0] = camera[1] = camera[2] = 0.f;
    yrot = 0.f;
    glEnable (GL_DEPTH_TEST);
    {
    GLfloat pos[4] = {3., 5., 2., 1.};
    GLfloat white[4] = {1., 1., 1., 1.};
    GLfloat black[4] = {0., 0., 0., 0.};
    glEnable (GL_LIGHTING);
    glEnable (GL_LIGHT1);
```

```
    glLightfv (GL_LIGHT1, GL_POSITION, pos);
    glLightfv (GL_LIGHT1, GL_DIFFUSE, white);
    glLightfv (GL_LIGHT1, GL_SPECULAR, white);

    /* ambient and diffuse will track glColor */
    glEnable (GL_COLOR_MATERIAL);
    glColorMaterial (GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
    glMaterialfv (GL_FRONT, GL_SPECULAR, white);
    glMaterialf (GL_FRONT, GL_SHININESS, 20.);
    }
  //  quadSphere = gluNewQuadric ();

    quadCyl = gluNewQuadric ();
    glutDisplayFunc (display);
    glutReshapeFunc (reshape);
    glutMotionFunc (motion);
    glutKeyboardFunc (key);
    glutTimerFunc (TIMER, timer, 0);
    s1=glutCreateMenu(pol);
    glutAddMenuEntry("Sphere",14);
    glutAddMenuEntry("Octahedron", 15);
    glutAddMenuEntry("Tetrahedron",16);
    glutAddMenuEntry("Icosahedron",17);
   s2= glutCreateMenu(col);
    glutAddMenuEntry("orange", 3);
    glutAddMenuEntry("red", 4);
    glutAddMenuEntry("Blue",5);
    glutAddMenuEntry("purple",6);
    glutAddMenuEntry("pink",7);
    s3=glutCreateMenu(selectFont);
    glutAddMenuEntry("Roman",0);
    // glutAddMenuEntry("Mono Roman",1);
```

```
  glutCreateMenu(main_menu);
  // glutCreateMenu(col);
   glutAddSubMenu("Object",s1);
 glutAddSubMenu("Colors",s2);
   //         glutAddSubMenu("texts",s3);
    glutAddMenuEntry("Quit", 0);
 glutAttachMenu(GLUT_RIGHT_BUTTON);
 }
int main(int argc, char** argv)
  {
  glutInit (&argc,argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
  glutInitWindowSize (windowWidth=300,300);
  glutInitWindowPosition (0,0);
  glutCreateWindow("MOVABLE CAMERA AND OBJECT");

  setup();

  init ();

  printf ("Left mouse button rotates the scene.\n");
  printf ("Move the target object:\n");
  printf ("\tq/Q\talong the X axis;\n");
  printf ("\tw/W\talong the Y axis;\n");
  printf ("\te/E\talong the Z axis.\n");
  printf ("Move the camera object:\n");
  printf ("\ta/A\talong the X axis;\n");
  printf ("\ts/S\talong the Y axis;\n");
  printf ("\td/D\talong the Z axis.\n");
  printf ("Any other key exits.\n");
  glutMainLoop ();
  }
```

# CHAPTER 7

## SNAPSHOT

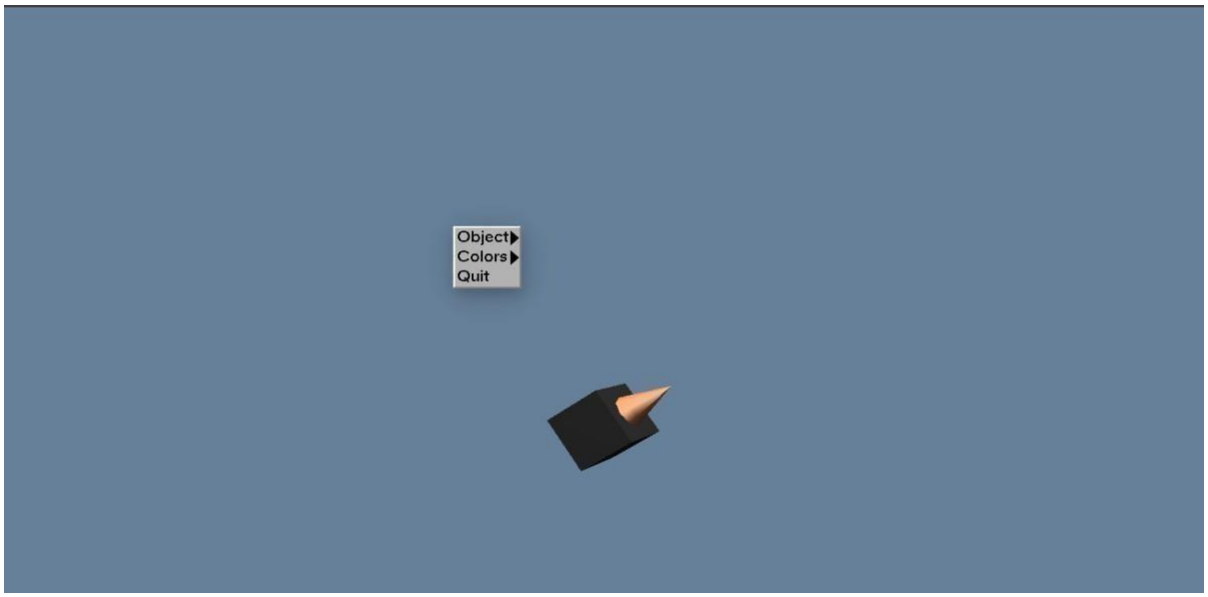



**Figure 7.1** screen with the menu By clicking on right mouse button the menu option will display on the output screen.

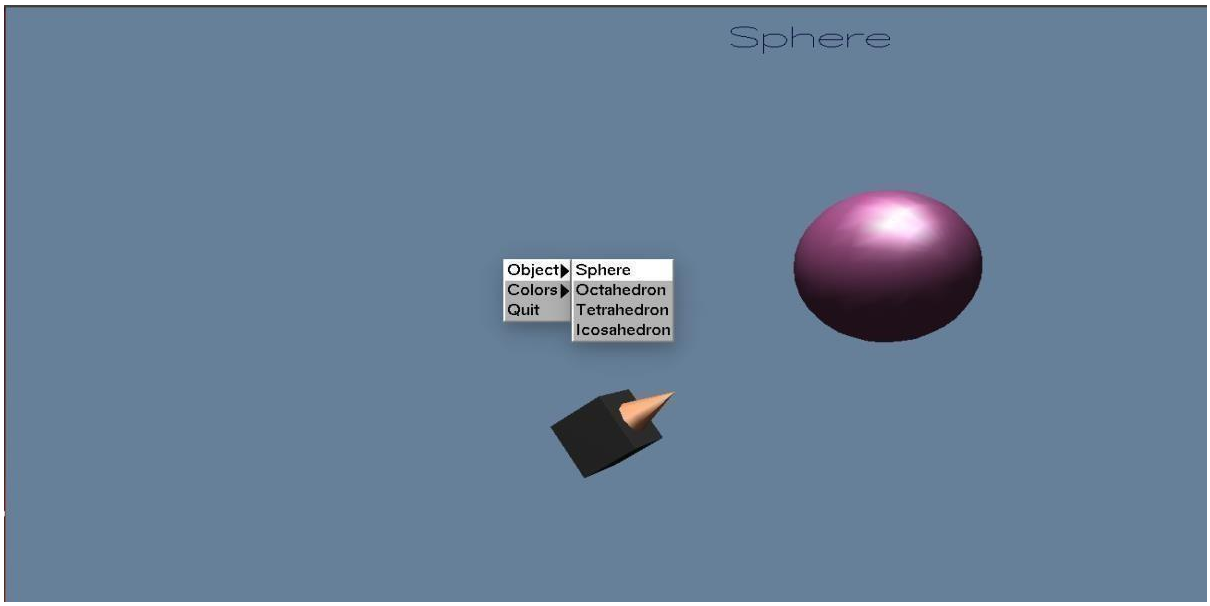



**Figure 7.2** Selecting object from menu By clicking on object in menu and selecting one object like sphere then sphere will display.

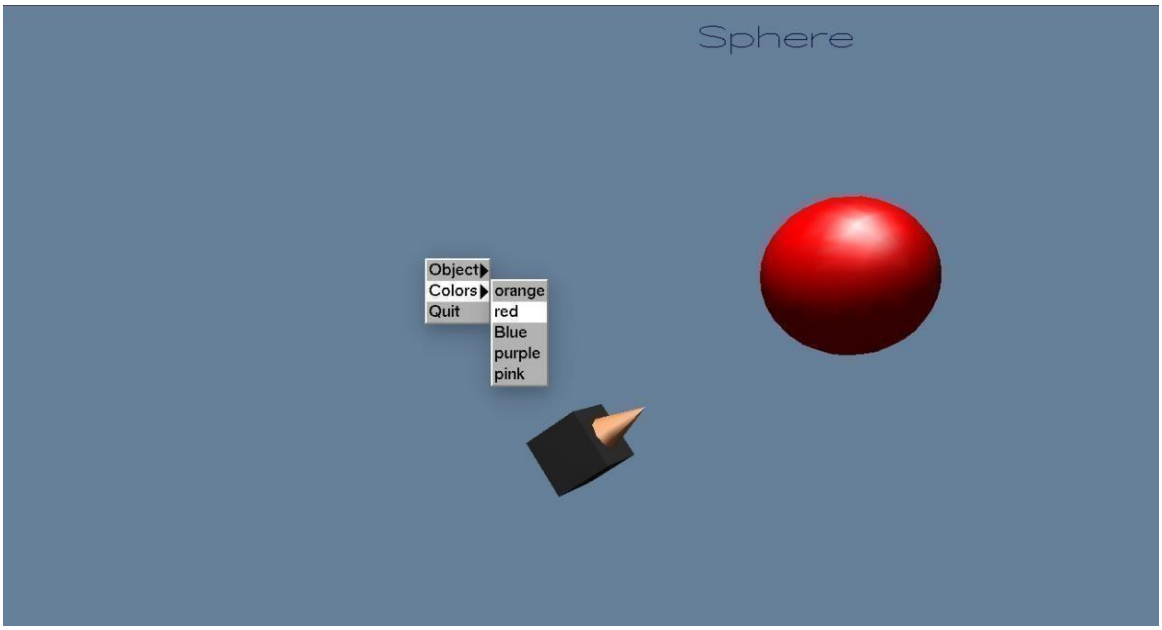



**Figure 7.3** Displays the different color options for object By selecting a color from menu like red then object color will change to red.

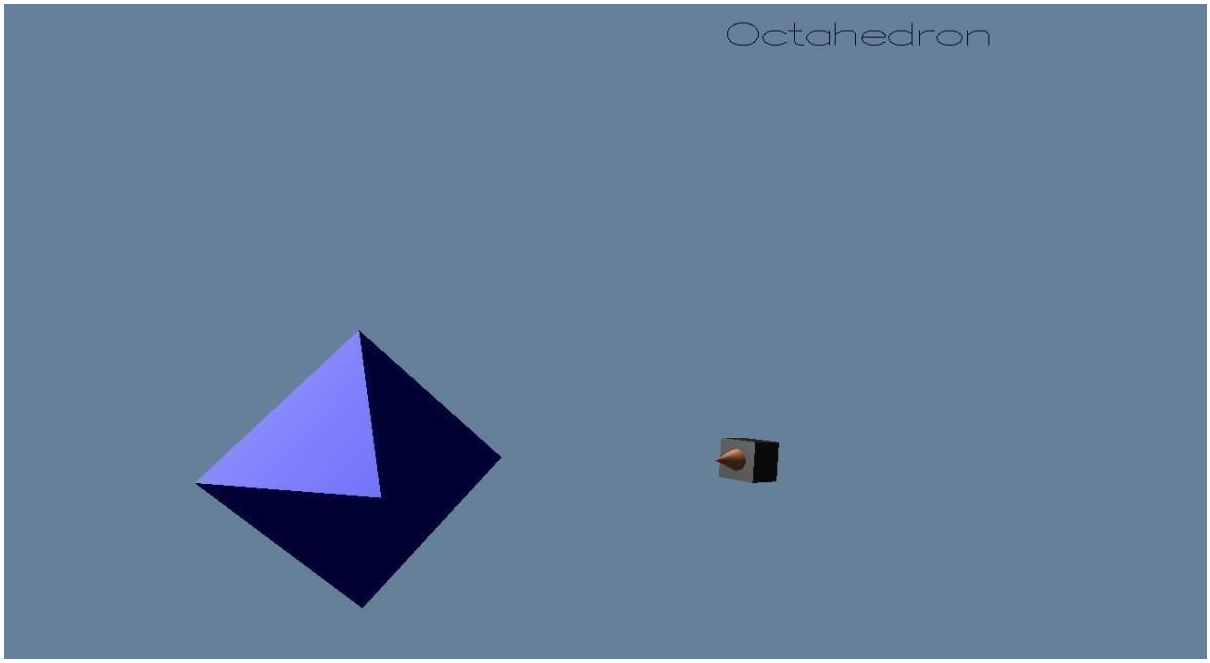



**Figure 7.4** The object moves in x-axis by pressing q/Q, the target object will move along x axis.

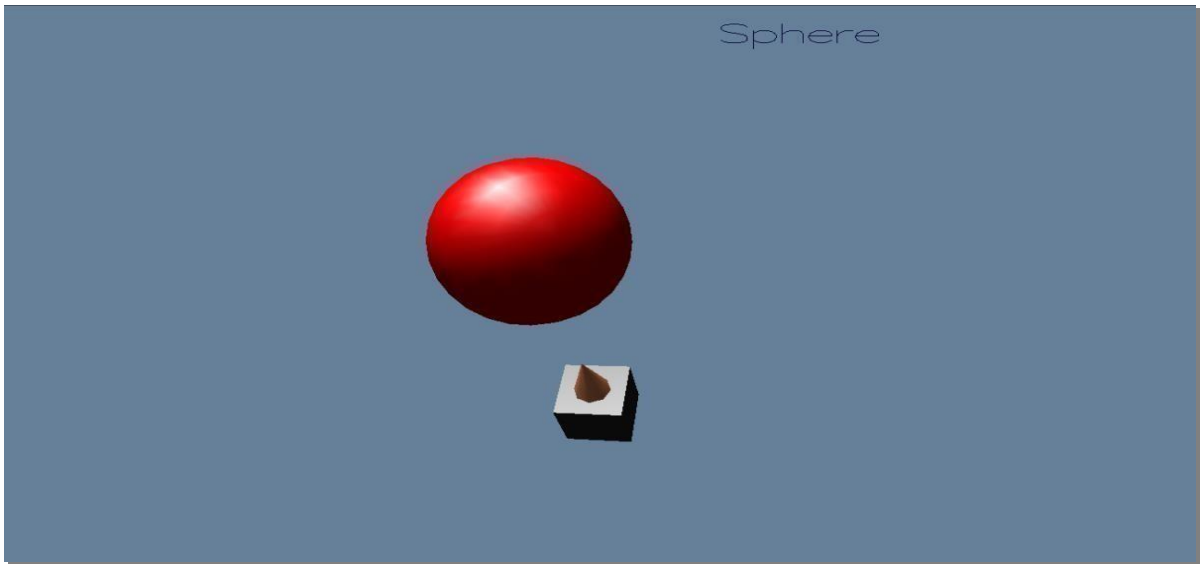



**Figure 7.5** The object moves in y-axis By pressing w/W,the target object will move along y axis.

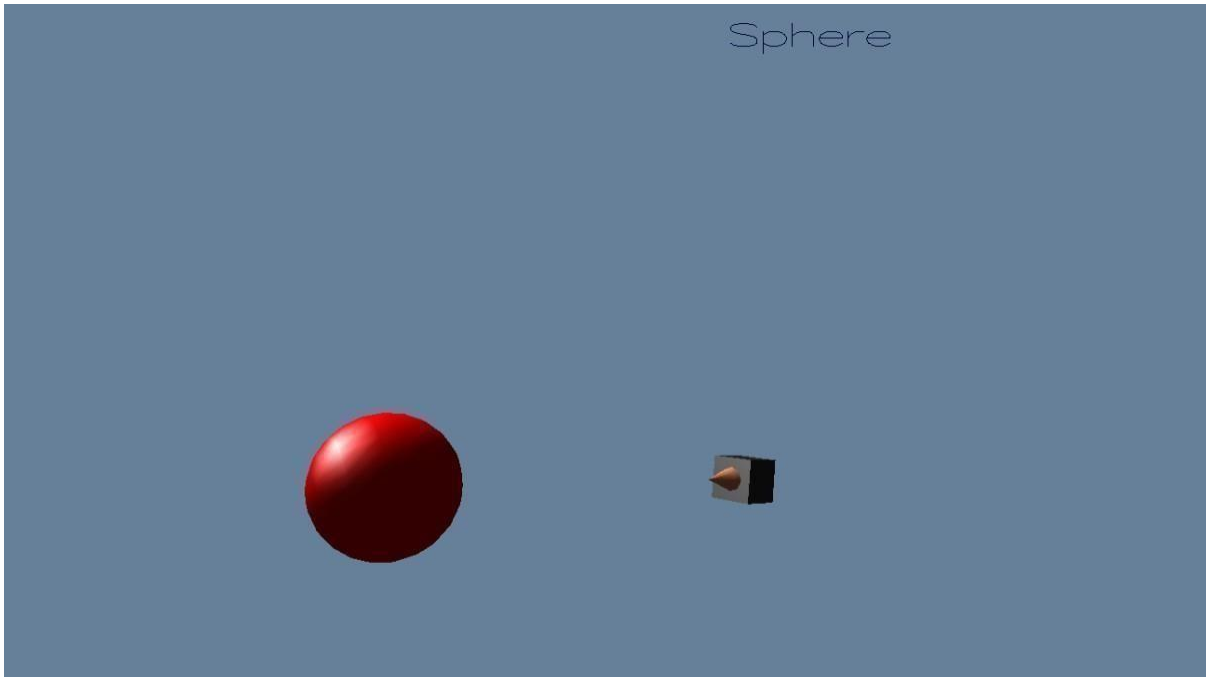



**Figure 7.6** The object moves in z-axis By pressing e/E, the target object will move along z axis.

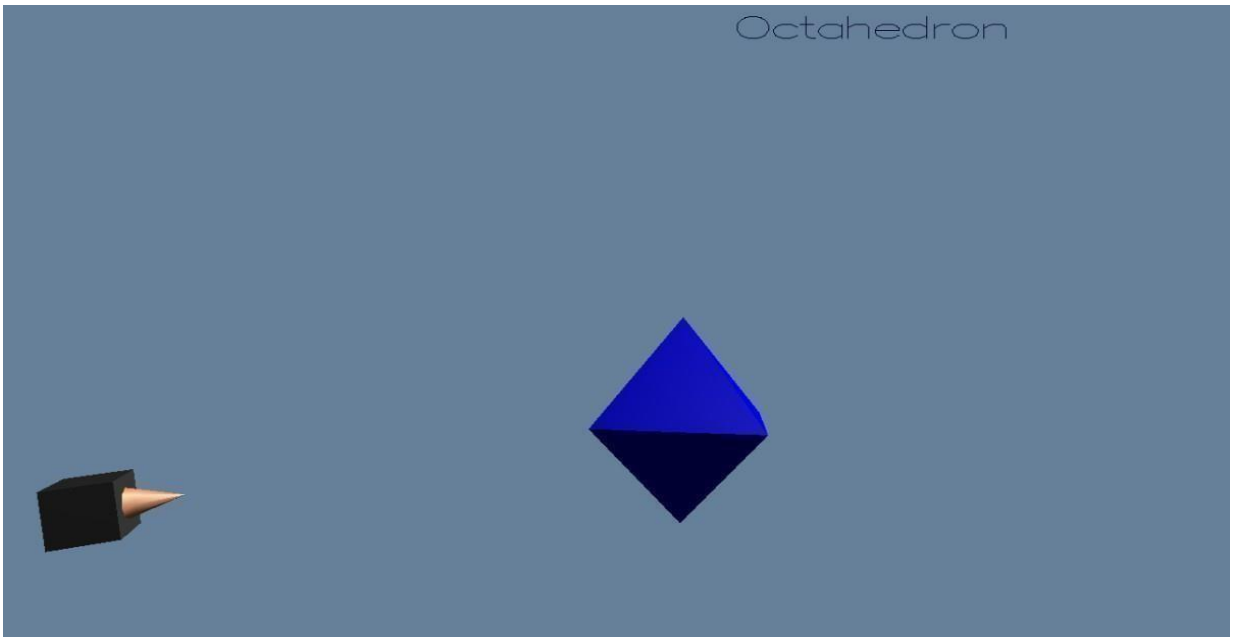



**Figure 7.7** The Camera moves in x-axis By pressing a/A, the camera will move along x axis.

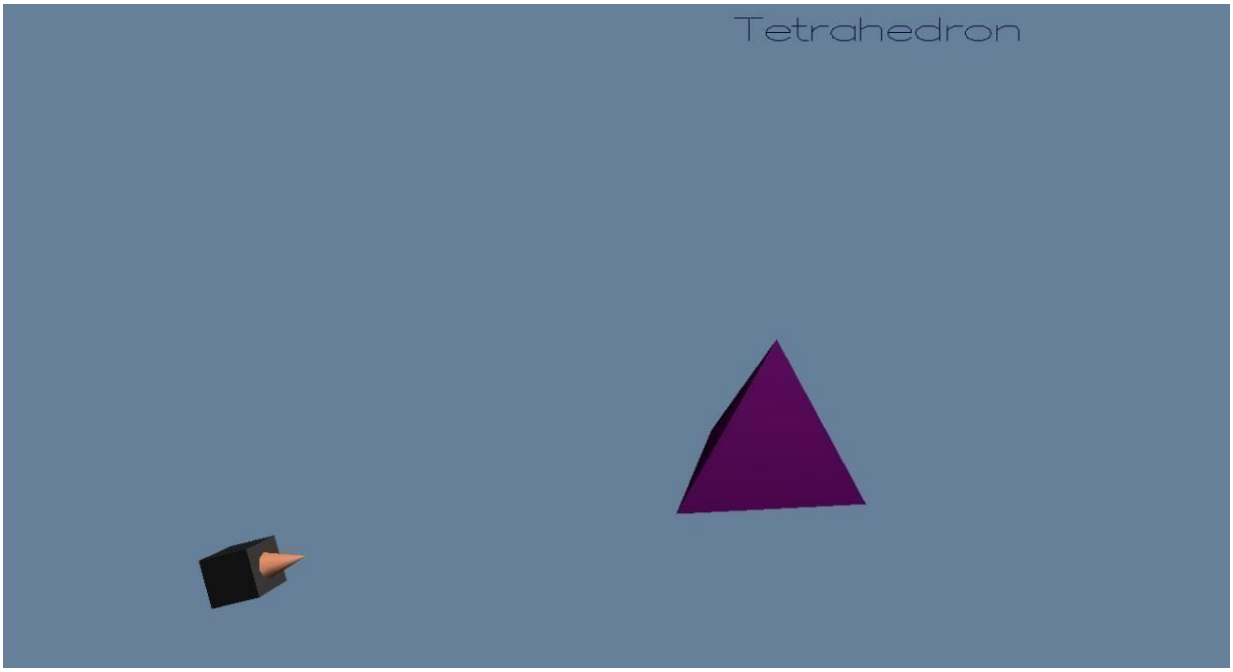



**Figure 7.8** The Camera moves in y-axis By pressing s/S, the camera will move along y axis.

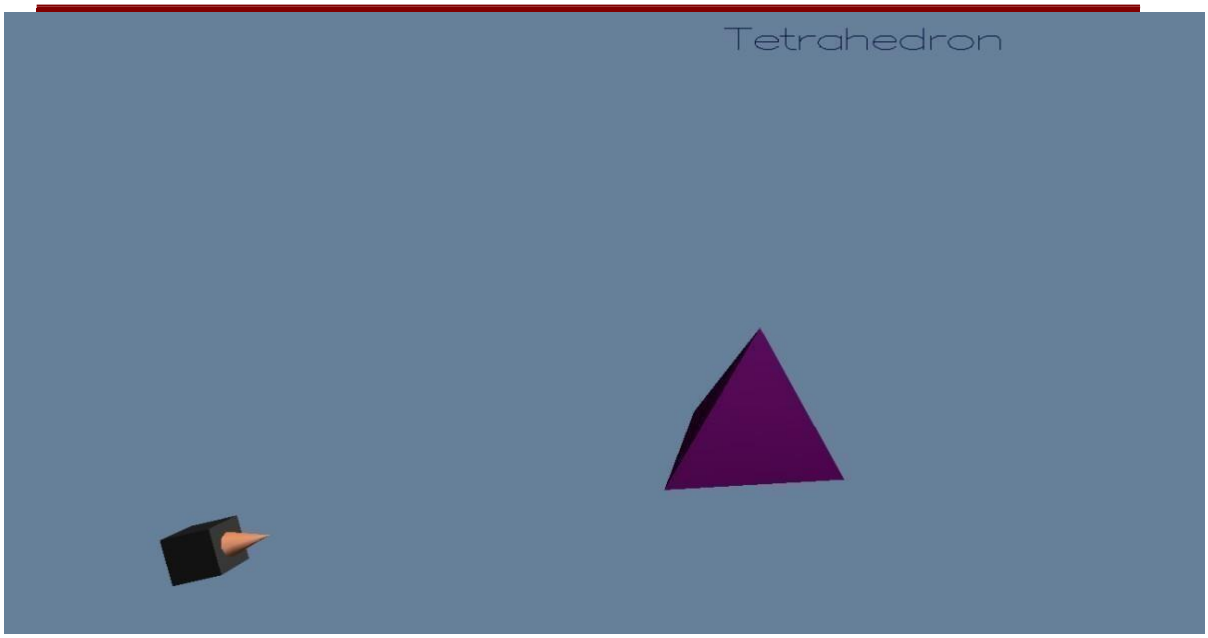



**Figure 7.9** The Camera moves in z-axis By pressing d/D, the camera will move along z axis.

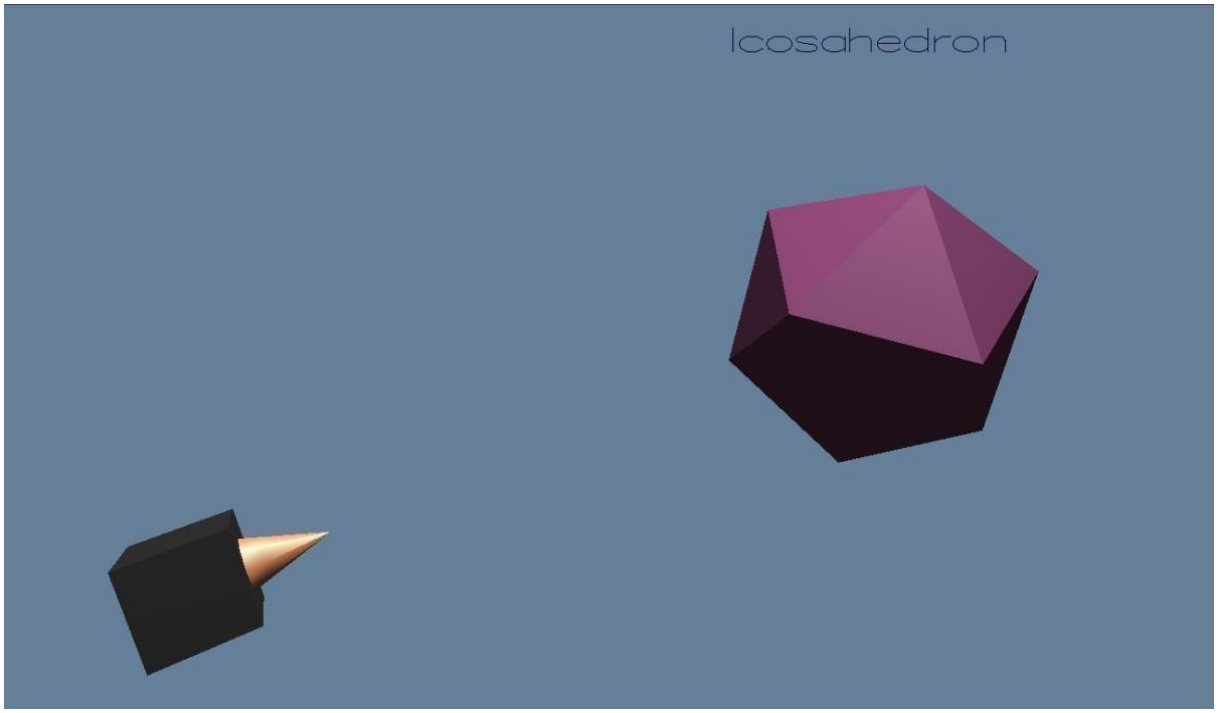



**Figure 7.10** Scene rotation By clicking and holding the left mouse button, the scene will be rotated.

# CHAPTER 8

## CONCLUSION

We have proposed in this paper a review of methods for moving objects detection with a moving camera categorized. the current image is registered to the background model in order to do the subtraction and obtain the moving objects. when several cameras are used, static and moving, it could be interesting to couple information to detect moving objects. when a moving object is detected in the static camera, generally with a large-view, the moving camera, generally a PTZ camera, will move to detect the moving object.

## REFERENCES

[1] Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version,3rd / 4th Edition, Pearson Education,2011

[2] Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL, 5th edition. Pearson Education, 2008

Google [online] available at https://www.opengl.org/resources/libraries/glut/spec3/node113.html accessed on [20-05-2019]

[3] Google [online] available at https://www.artificialintelligenceinindia.com/water-jug-problem-artificial-intelligence/amp/ accessed on [21-05-2019]

[4] Snehal Solanki, Prem Parmar, Parth Shukla, Dhruvin Patel, Mr. Nimit Modi, Dr. Sheshang Degadwala, “A Review on Two Water Jugs Problem via an Algorithmic Approach” International Journal of Scientific Research in Science, Engineering and Technology| Online ISSN : 2394-4099, Volume 4, Issue 5 (March-April, 2017), PP. 331-334

[5] Yiu-Kwong Man, “An Arthimetic Approach to the General Two Water Jugs Problem” Proceedings of the World Congress on Engineering 2013| Online ISSN : 2078-0966, Volume 1, (July 2013)