VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI

MINI PROJECT REPORT ON

3D DOLL

A report submitted in partial fulfillment of the requirements for

COMPUTER GRAPHICS AND VISUALIZATION LABORATORY(17CSL68)

in

IN COMPUTER SCIENCE & ENGINEERING

Submitted By

KAVYA	**4AL17CS041**
KAVYA R SHETTY	**4AL17CS042**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ALVA'S INSTITUTE OF ENGINEERING AND TECHNOLOGY

MOODBIDRI-574225, KARNATAKA

2020 – 2021

ALVA'S INSTITUTE OF ENGINEERING AND TECHNOLOGY MIJAR, MOODBIDRI D.K. -574225 KARNATAKA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the Mini Project entitled **"3D DOLL"** has been successfully completed by

KAVYA	**4AL17CS041**
KAVYA R SHETTY	**4AL17CS042**

the bonafide students of **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING** of the **VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI** during the year 2020–2021. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The Mini project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the Bachelor of Engineering Degree.

Ms.Shilpa

Mini Project Guide

Dr. Manjunath Kotari

HOD CSE

External Viva

Name of the Examiners **Signature with Date**

1.

2.

ALVA'S INSTITUTE OF ENGINEERING AND TECHNOLOGY MIJAR, MOODBIDRI D.K. -574225 KARNATAKA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Declaration

We,

KAVYA

KAVYA R SHETTY

hereby declare that the dissertation entitled, 3D DOLL is completed and written by us under the supervision of my guide **Ms. Shilpa , Assistant Professor Department of Computer Science and Engineering, Alva's Institute of Engineering And Technology, Moodbidri, DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING** of the **VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI** during the academic year 2020-2021.The dissertation report is original and it has not been submitted for any other degree in any university.

KAVYA	**4AL17CS041**
KAVYA R SHETTY	**4AL17CS042**

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany a successful completion of any task would be incomplete without the mention of people who made it possible, success is the epitome of hard work and perseverance, but steadfast of all is encouraging guidance.

So, with gratitude we acknowledge all those whose guidance and encouragement served as beacon of light and crowned the effort with success.

The selection of this mini project work as well as the timely completion is mainly due to the interest and persuasion of our mini project coordinator **Ms.Shilpa,** Department of Computer Science & Engineering. We will remember her contribution for ever.

We sincerely thank, **Dr. Manjunath Kotari**, Professor and Head, Department of Computer Science & Engineering who has been the constant driving force behind the completion of the project.

We thank our beloved Principal **Dr. Peter Fernandes,** for his constant help and support throughout.

We are indebted to **Management of Alva's Institute of Engineering and Technology, Mijar, Moodbidri** for providing an environment which helped us in completing our mini project.

Also, we thank all the teaching and non-teaching staff of Department of Computer Science & Engineering for the help rendered.

KAVYA 4AL17CS041

KAVYA R SHETTY 4AL17CS042

ABSTRACT

Complex operations have been eschewed in favor of simple, direct control over the fundamental of 2D and 3D graphics. Higher level graphical functions may, however, is built from Open GL's low level operators, as the operators have been designed with such layering in mind. This project demonstrates 3D dolls.. In which the dolls can be moved when the mouse is moved and we can perform zoom in, out operation on the same.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Computer Graphics

Computer graphics is concerned with all aspects of producing pictures or images using a computer. This field began almost 50 years ago, with the display of a few lines on a cathode ray tube. Now we can create images by computer that is indistinguishable from photographs of real objects.

Computer graphics is one of the most effective and most commonly used means of communication with the user. It displays the information in the form of graphical objects such as pictures, charts, graphs and diagrams instead of simple text. The pictures or graphical objects may vary from engineering drawings, business graphs and architectural structures to animated movies. All the functionalities required for the development and presentation of such as environment or interface to the user is provided by the graphics package.

There are numerous ways in which computer graphics had made user interaction fast, effective and fun. Graphics has enabled the designers to introduce the concept of windows that act as the virtual graphics terminals, each of which is capable of running an independent application. The introduction of the mouse has made the section of the objects on the interface easy by the “point and click” facility and a lot more.

With the speedily increasing enhancements in the field of computer graphics one can simulate real world objects, create motion by using the different strategies introduced in 2D,3D and 4D dynamics, one can produce independent frames, produce packages for scientific and engineering visualizations, in the field of medicine for the study of human behaviour and a lot more. So, we can see that computer graphics has become an integral part of life today and will continue and ease the usage of computers further more in the near future.

1.2 History of Computer Graphics

Computer Graphics is the creation, manipulation, and storage of models and images of picture objects by the aid of computers. This was started with the display of data on plotters and CRT. Computer Graphics is also defined as the study of techniques to improve the communication between user and machine, thus Computer Graphics is one of the most effective medium of communication between machine and user.

William fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeng. One of the first displays of computer animation was future world (1976), which included an animation of a human face and hand-produced by Carmull and Fred Parkle at the University of Utah.

There are several international conferences and journals where the most significant results in computer-graphics are published. Among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on Graphics journals.

1.3 Applications of Computer Graphics

The applications of computer graphics can be divided into four major areas:

Display of information

Design

Simulations and animation

User interfaces

Display of information

Computer graphics has enabled architects, researchers and designers to pictorially interpret the vast quantity of data. Cartographers have developed maps to display the celestial and geographical information. Medical imaging technologies like Computerized Tomography(CT), Magnetic Resonance Imaging(MRI), Ultrasound and many others make use of computer graphics.

Design

Professions such as engineering and architecture are concerned with design. They start with a set of specifications, seek cost-effective solutions that satisfy the specifications. Designing is an interactive process. Designers generate a possible design, test it and then use the results as the basis for exploring other solutions. The use of interactive graphical tools in computer Aided Design(CAD) pervades the fields including architecture, mechanical engineering, the design of very large scale integrated(VLSI)circuits and creation of characters for animation.

Simulation and animation

Once, the graphic system evolved to be capable of generating sophisticated images in real time, engineers and researches began to use them as simulations. Graphical flight simulators have proved to increase the safety and to reduce the training expenses. The field of virtual reality(VR) has opened many new horizons. A human viewer can be equipped with a display headset that allow him/her to see the images with left eye and right eye which gives the effect of stereoscopic vision. This has further led to motion pictures and interactive video games.

User interfaces

Computer graphics has to led to the creation of graphical user interface(GUI) using which even naïve users are able to interact with a computer. Interaction with the computer has been dominated by a visual paradigm that includes windows, icons, menus and a pointing device such as mouse. Millions of people are internet users; they access the internet through the graphical network browsers such as Microsoft internet explorer and Mozilla Firefox.

CHAPTER 2

OpenGL

2.1 Introduction to OpenGL

OpenGL (Open Graphics Library) is a hardware-independent, operating system independent, vendor neutral graphics API specifications. It is a standard specification defining a cross language, cross platform API for writing applications that produce 2D and 3D computer graphics. Many vendors provide implementation of this specification for a variety of hardware platforms. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scene from simple primitives. OpenGL has been designed using a client/server paradigm, allowing the client application and the graphics server controlling the display hardware to exist on the same or separate machines. OpenGL was developed by Silicon Graphics Inc.(SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, and flight simulation. It is used in video games, where it competes with Direct3D on Microsoft platforms. OpenGL is managed by a non-profit technology consortium, the Khronous group.

GLUT is designed to fill the need for a window system independent programming interface for OpenGL programs. Removing window system operations from OpenGL is a sound decision because it allows the OpenGL graphics system to be retargeted to various systems including powerful but expensive graphics workstations as well as mass-production graphics systems like video games, set-top boxes for interactive television, and PCs. GLUT simplifies the implementation of programs using OpenGL rendering. The GLUT routines also take relatively few parameters.

Some features of OpenGL include the following:

Geometric and raster primitives

RGBA or color index mode

Display list or immediate mode

Viewing and modelling transformations

Lighting and shading

Hidden surface removal (Depth Buffer)

Texture Mapping

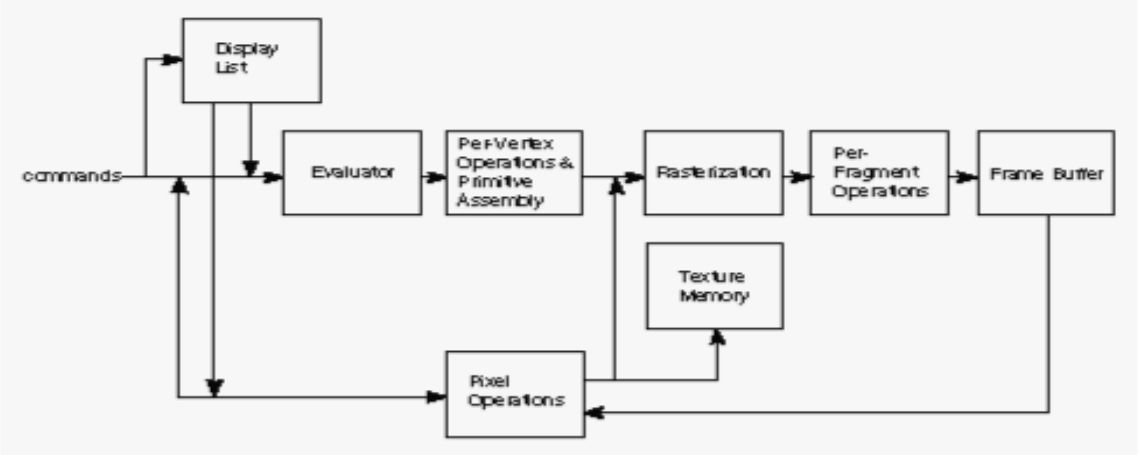


Fig 1.1: OpenGL Block Diagram

The figure shown above gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

We can choose to accumulate some of commands in a *display list* for processing at a later time. The *evaluator* stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. *Rasterization* produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each *fragment* so produced is fed into the last stage, *per-fragment operations*, which perform the final operations on the data before it's stored as pixels in the *frame buffer*.

2.2 Limitation

OpenGL is case-sensitive.

Line Color, Filled Faces and Fill Color not supported.

Bump mapping is not supported.

Navigation render is not supported.

3D measurement is not supported.

Streaming of individual 3D objectives is not supported.

2.3 Advantages of OpenGL

Stable

OpenGL implementation has been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled and proposed updates are announced in time for developers to adopt changes. A backward compatibility requirement ensures that existing applications do not become obsolete.

Reliable and portable

All OpenGL applications produce consistent visual display result on OpenGL API-complaint hardware, regardless of operating system or windowing system.

Easy to use

OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in application with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drives encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.

Well-documented

Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

Industry standard

An independent consortium, the OpenGL architecture review board, guides the OpenGL specification. With board industry support, OpenGL is the truly open, vendor-neutral, multiplatform graphics standard.

CHAPTER 3

REQUIREMENT SPECIFICATION

3.1 Functional Requirements

Here, we have made an attempt to design a Public Park algorithm using OpenGL. The software has been written using C language and data structures like simple structure type are used.

The project requires the access of OpenGL utility toolkit through the use of the header file “glut.h”. This header file, in addition to the usual header files is needed for the working of the project. For running the program, any basic PC running compatible version of Microsoft Visual Studio is sufficient.

3.2 Non-Functional Requirements

The software should produce the informative error messages, if any errors are found in the input program. It should use memory as less as possible, dynamic memory allocation is preferable to accomplish this task.

3.3 Hardware Requirements

The minimum/recommended hardware configuration required for developing the proposed software is given below:

PENTIUM-2 and above compatible systems.

512 MB RAM.

Approximately 170 MB free space in the hard disk.

Hard disk access time must be less than 19 milliseconds

3.4 Software Requirements

Open GL

WINDOWS XP/ME/9X

Ubuntu

3.4.1 Why C Language for this project?

C is a minimalist programming language. Among its design goals were that it could be compiled in a straight forward manner using a relatively simple compiler, provided low-level access to memory, generated only a few machine language instructions for each of its core language elements and did not require extensive run-time support. As a result, C code is suitable for many system programming applications that had traditionally been implemented in assembly language.

Despite its low-level capabilities, the language was designed to encourage machine independent programming. A standard compliant and portably written C program can be compiled for a very wide variety of computer platforms and operating systems with minimal change to its source code.

3.4.2 Graphics in C

C itself doesn't support any graphics library. In order for C to be completely portable from platform to platform it has focused on providing platform independent functions, such as file access, text string manipulation, mathematical functions, etc. So in order to get graphics in C, one needs to do one of four things:

Write a 16 bit DOS program and use assembly language for graphics

Make calls to the Windows API

Make calls to the OpenGL

subsystem Use a graphics library

3.4.3 OpenGL subsystem

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to simplify the programming tasks, including the following:

The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and

rendering surfaces. The library is provided as part of every OpenGL implementation.

The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit. GLUT routines use the prefix glut.

CHAPTER 4

SYSTEM DESIGN

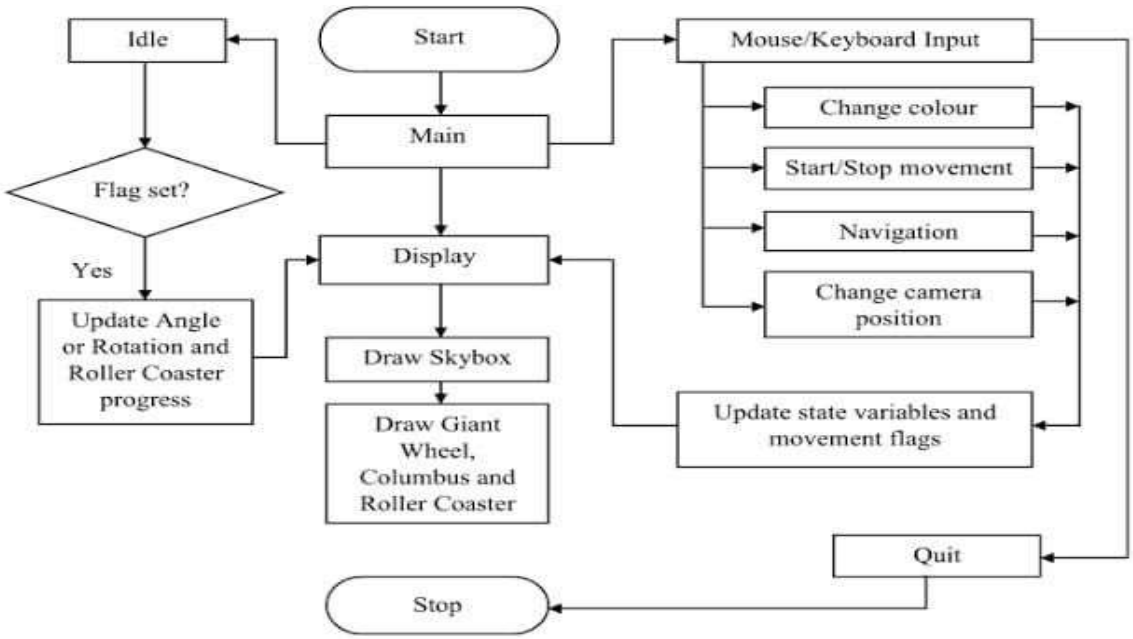
4.1 Initialization

- Initialize to interact with the windows.
- Initialize the display mode that is double buffer and RGB color system.
- Initialize Main window position and size and Sub window position and size.
- Initialize and create the Main window and Sub window to display the output.

4.2 Display

- Displaying player bow and arrows
- The Operations performed are
    - Arrow have to shoot the moving balloon using keyboard keys.

4.3 Flow Diagram



CHAPTER 5

IMPLEMENTATION

5.1 Header files Used

#include<stdio.h>

The standard input/output library requires the use of a header file called stdio.h. This include command is a directive that tells the compiler to use the information in the header file called stdio.h. The initial stdio stands for standard input output, and stdio.h file contains all the instructions the compiler needs to work with disk files and send information to the output device.

#include<GL/glut.h>

GL is the fundamental OpenGL library.it provides functions that are permanent part of OpenGL. The function starts with characters ‘gl’.

GLUT, the GL utility tool kit supports developing and managing menus and managing events. The functions start with characters ‘glut’.

GLU, the GL utility Library provides high-level routines to handle certain matrix operations, drawing of quadric surfaces such as sphere and cylinders. The functions start with characters ‘glu’.

#include<stdlib.h>

The ISO C standard this header file as a place to declare certain standard library fu8nctions.These include the memory management functions (malloc, free) communication with the environment(aqblort,exit)and others, not yet all the standard functions of this header file are supported, If a declaration is present in the supplied header file, then UCR supports it and will continue to support it. If a function is not there, it will be added in time.

5.2 Functions for the project

The Algorithm is constructed in a set of functions. We have used several user defined functions and built-in functions.

The following are the functions used in this project and their purpose **glutInitDisplayMode (GLUT_DOUBLE|GLUT_RGB)** specifies whether to use an *RGB* or color-index color model. You can also specify whether you want a single- or double-buffered window. (If you're working in color-index mode, you'll want to load certain colors into the color map; use **glutSetColor** () to do this.) Finally, you can use this routine to indicate that you want the window to have an associated depth, stencil, and/or accumulation buffer. For example, if you want a window with double buffering, the RGBA color model, and a depth buffer, you might call **glutInitDisplayMode (*GLUT_DOUBLE | GLUT _RGB | GLUT_DEPTH*).**

5.3 Display callbacks

The **glutDisplayFunc** (void (*func)(void)) is the first and most important event callback in the code. Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by **glutDisplayFunc**() is executed. Therefore, it registers display callback function.

If the program changes the contents of the window,sometimes we will have to call **glutPostReDisplay**(void),which gives **glutMainLoop()** a nudge to call the registered display callback at its next opportunity.

5.4 Running the program

The very last thing to be done is call glutMainLoop (void). All windows that have been created are now shown, and rendering to those windows is now effective. Event processing begins, and the registered display callback is triggered. Once this loop is entered, it is never exited!

The various other functions used are described in more detail below:

1. **glutBitmapCharacter (font, string[size])** it renders the character in the named bitmap font and advances the current raster position.
2. **glutInit (&argc, char **argv):** this command initializes GLUT. The argument from main are passed in and be used by the application.
3. **glutCreateWindow (“name”)** creates a window on the display. The string title can be used to label the window.

4. **glutInitWindowSize()** specifies the initial height and width of the window in pixels.
5. **glutMainLoop ()** causes the program to enter an event processing loop.
6. **glutPostRedisplay()** marks the current window as needing to be redisplayed.
7. **glutSwapBuffers**() swaps the buffers of the current window if double buffered.
8. **glutTimerFunc()** registers the timer callback function to be triggered in atleast milliseconds
9. **glutDisplayFunc()** Registers the display function that is executed when the window needs to be redrawn.
10. **glRasterPosition()** specifies a raster position.
11. **glMatrixMode()** specifies which matrix will be affected by subsequent transformations, Mode can be GL_MODEL_VIEW.
12. **glViewport()** specifies the affine transformation of x and y from normalized device coordinates to window coordinates.
13. **glVertex2f()** specifies x and y co-ordinates of the vertex.
14. **glColor3f()** sets the current color.
15. **glFlush()** forces any buffered OpenGL commands to execute

5.5 User Implementation Code

VARIABLES

```
#include<windows.h>
#include <stdio.h>
#include <GL/glut.h
#include <time.h>
#define INIT_VIEW_X 0.0
#define INIT_VIEW_Y 0.0
#define INIT_VIEW_Z -4.5
#define VIEW_LEFT -2.0
#define VIEW_RIGHT 2.0
#define VIEW_BOTTOM -2.0
#define VIEW_TOP 2.0
#define VIEW_NEAR 1.0
#define VIEW_FAR 200.0
GLfloat AmbientLight[]={0.3,0.3,0.3,1.0};
GLfloat DiffuseLight[] ={0.8,0.8,0.8,1.0};
GLfloat SpecularLight[] ={1.0,1.0,1.0,1.0};
```

```
    GLfloat SpecRef[] = {0.7,0.7,0.7,1.0};
    GLfloat LightPos[] = {-50.0,50.0,100.0,1.0};
    GLint Shine =128;
   GLint walkX=0,walkY=0,lookX=0,lookY=0;
   GLint world=1,oldX=-1,oldY=-1;
   GLint doll=-1

  void eyeright()
  {
     glPushMatrix();
     glTranslatef(.17,1.1,.75);
     glRotatef(-45,0,0,1);
     glScalef(.9,.7,.7);
     glColor3f(1.0,1.0,1.0);
     gluSphere(gluNewQuadric(),.3,100,100);
     glPopMatrix();
  }

    void eyeleft()
  {
      glPushMatrix();
      glTranslatef(-.17,1.1,.75);
      glRotatef(45,0,0,1);
      glScalef(.9,.7,.7);
      glColor3f(1.0,1.0,1.0);
     gluSphere(gluNewQuadric(),.3,100,100);
     glPopMatrix();
  }

void legleft()
{
   glPushMatrix();
   glColor3f(1,1,1);
   glTranslatef(.3,-.5,0);
   glRotatef(-90.0,1,0,0);
   glScalef(.8,.8,.8);
   gluCylinder(gluNewQuadric(),.5,.5,.5,30,6);
   glPopMatrix();
}
```

```
void legright()
{
    glPushMatrix();
    glColor3f(1,1,1);
    glTranslatef(-.3,-.5,0);
    glRotatef(-90.0,1,0,0);
    glScalef(.8,.8,.8);
    gluCylinder(gluNewQuadric(),.5,.5,.5,30,6);
    glPopMatrix();
}

void armleft()
{
    glPushMatrix();
    glColor3f(0,0,0);
    glTranslatef(-.82,0,.1);
    glRotatef(90,0,1,0);
    glRotatef(-50,1,0,0);
    gluCylinder(gluNewQuadric(),.15,.15,.48,30,6);
    glPopMatrix();
}
void armright()
{
    glPushMatrix();
    glColor3f(0,0,0);
    glTranslatef(.82,0,.1);
    glRotatef(90,0,1,0);
    glRotatef(-130,1,0,0);
    gluCylinder(gluNewQuadric(),.15,.15,.48,30,6);
    glPopMatrix();
}
void handleft()
{
    glPushMatrix();
    glColor3f(1,1,1);
    glTranslatef(.82,0,.1);
    glScalef(.4,.3,.3);
    gluSphere(gluNewQuadric(),.4,100,100);
    glPopMatrix();
}
```

```
void handright()
{
   glPushMatrix();
   glColor3f(1,1,1);
   glTranslatef(-.82,0,.1);
   glScalef(.4,.3,.3);
   gluSphere(gluNewQuadric(),.4,100,100);
   glPopMatrix();
}

void mouth()
{
   glPushMatrix();
   glTranslatef(0,.78,.74);
   glScalef(.4,.4,.1);
   glColor3f(0.0,0.0,0.0);
   gluSphere(gluNewQuadric(),.4,100,100);
   glPopMatrix();
}

void teeth()
{
   glPushMatrix();
   glColor3f(1.0,1.0,1.0);
   glTranslatef(-.08,.72,.76);
   glTranslatef(.055,0,.005 );
   glutSolidCube(.035);
   glTranslatef(.055,0,0 );
   glutSolidCube(.035);
   glPopMatrix();
}
void eyebrowleft()
{
   glPushMatrix();
   glTranslatef(-.3,1.5,.97);;
   glRotatef(90,0,1,0);
   glRotatef(45,1,0,0);
   glColor3f(0.0,0.0,0.0);
   gluCylinder(gluNewQuadric(),.05,.01,.3,4,6);
   glPopMatrix();
}
```

```
void eyebrowright()
{
   glPushMatrix();
   glTranslatef(.3,1.5,.97);
   glRotatef(270,0,1,0);
   glRotatef(45,1,0,0);
   gluCylinder(gluNewQuadric(),.05,.01,.3,4,6);
   glPopMatrix();
}
   void neckring()
{
   glPushMatrix();
   glColor3f(1,1,1);
   glTranslatef(0,.5,0);
   glScalef(.59,.59,.59);
   glRotatef(90.0,1,0,0);
   glutSolidTorus(.1,1.0,20,20);
   glPopMatrix();
}
```

FUNCTION FOR HEAD MOMENT

```
void head()
{
   glPushMatrix();
   glTranslatef(0,1.2,0);
   glScalef(.9 ,.9,.9 );
   glColor3f(1.0,0.8,0.6);
   gluSphere(gluNewQuadric(),1,100,100);
   glPopMatrix();
}
void maintopball()
{
   glPushMatrix();
   glTranslatef(0,2.2,0);
   glScalef(.9,.9,.9);
   gluSphere(gluNewQuadric(),.18,100,100);
   glPopMatrix() ;
}
void hatring()
{
   glPushMatrix();
```

```
    glTranslatef(0,1.4,0);
    glScalef(.84,.84,.84);
    glRotatef(90.0,1,0,0);
    glutSolidTorus(.1,1.0,20,20);
    glPopMatrix();
}

void footleft()
{
    glPushMatrix();
    glTranslatef(-.3,-.5,0);
    glScalef(1.5,.3,1.5);
    glColor3f(0.0,0.0,0.0);
    gluSphere(gluNewQuadric(),.3,100,100);
    glPopMatrix();
}

void footright()
{
    glPushMatrix();
glTranslatef(.3,-.5,0);
    glScalef(1.5,.3,1.5);
    glColor3f(0.0,0.0,0.0);
    gluSphere(gluNewQuadric(),.3,100,100);
    glPopMatrix();
}

void bellyCoatbottom()
{
    glPushMatrix();
   // glColor3f(1,1,1);
    glTranslatef(0,-.2,0);
    glScalef(1,.7,1);
    glRotatef(90.0,1,0,0);
    gluDisk(gluNewQuadric(),0,.8,30,30);
    glPopMatrix();
}
void BellyCoat()
{
    glPushMatrix();
    glColor3f(0,0,0);
    glTranslatef(0,.5,0);
    glScalef(1,.7,1);
    glRotatef(90.0,1,0,0);
```

```
    gluCylinder(gluNewQuadric(),.6,.8,1,100,100);
    glPopMatrix();
}

void pupilleft()
{
    glPushMatrix();
    glColor3f(0,0,0);
    glTranslatef(-.17,1.1,.88);
    glScalef(.9,.9,.9);
    gluSphere(gluNewQuadric(),.1,100,100);
    glPopMatrix();
}

void pupilright()
{
    glPushMatrix();
    glTranslatef(.17,1.1,.88);
    glScalef(.9,.9,.9);
    gluSphere(gluNewQuadric(),.1,100,100);
    glPopMatrix();
}

void topbutton()
{
    glPushMatrix();
    glColor3f(1,1,1);
    glTranslatef(-.1,.4,.85);
    glScalef(1.9,1.9,1.9);
    gluSphere(gluNewQuadric(),.04,100,100);
    glPopMatrix();
}
void middlebutton()
{
    glPushMatrix();
    glColor3f(1,1,1);
    glTranslatef(-.1,.15,.98);
    glScalef(1.9,1.9,1.9);
    gluSphere(gluNewQuadric(),.04,100,100);
    glPopMatrix();
}
void bottombutton()
{
    glPushMatrix();
```

```
    glColor3f(1,1,1);
   glTranslatef(-.1,-.1,.92);
   glScalef(1.9,1.9,1.9);
  glColor3f(0.0,0.0,0.0);
   gluSphere(gluNewQuadric(),.04,100,100);
   glPopMatrix();
}
```

DISPLAY FUNCTION

```
void Display()
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glColor3ub(50, 50, 150);//Change the draw color to slate blue
   glPushMatrix():
   if(world==1)
           {
              glTranslatef(walkX,-1,walkY);
             glRotatef(lookY,0,1,0);
             glRotatef(lookX,1,0,0);
           }
   glPushMatrix();
   glTranslatef(-1,0,-6);
   if(doll==1)
           {
             glTranslatef(walkX,-1,walkY);
             glRotatef(lookY,0,1,0);
             glRotatef(lookX,1,0,0);
           }
   eyeright();
  eyeleft();
  eyebrowleft()
  eyebrowright();
   glColor3f(0.0,1.0,0.0);
   neckring();
   glColor3ub(50,40,60);
   legright();
   legleft();
   glColor3ub(255,90,0);
   armleft();
   armright();
   BellyCoat();
   bellyCoatbottom();
   glColor3ub(0,185,0);
   handleft();
```

```
    handright();
    mouth();
    teeth();
    glColor3ub(255,222,173);
    head();
    glColor3f(0.0,0.0,0.0);
    footleft();
    footright();
    topbutton();
    middlebutton();
    bottombutton();
    pupilleft();
    pupilright();
    glPopMatrix();
    glPopMatrix();
    glutSwapBuffers();

}
void SetupRend()
{
    glClearColor(0.7,0.7,1.0,1.0);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_LIGHTING);
  glLightfv(GL_LIGHT0,GL_AMBIENT,AmbientLight);
  glLightfv(GL_LIGHT0,GL_DIFFUSE,DiffuseLight);
  glLightfv(GL_LIGHT0,GL_SPECULAR,SpecularLight);
  glEnable(GL_LIGHT0);
  glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);
    glMaterialfv(GL_FRONT,GL_SPECULAR,SpecRef);
    glMateriali(GL_FRONT,GL_SHININESS,Shine);
}
```

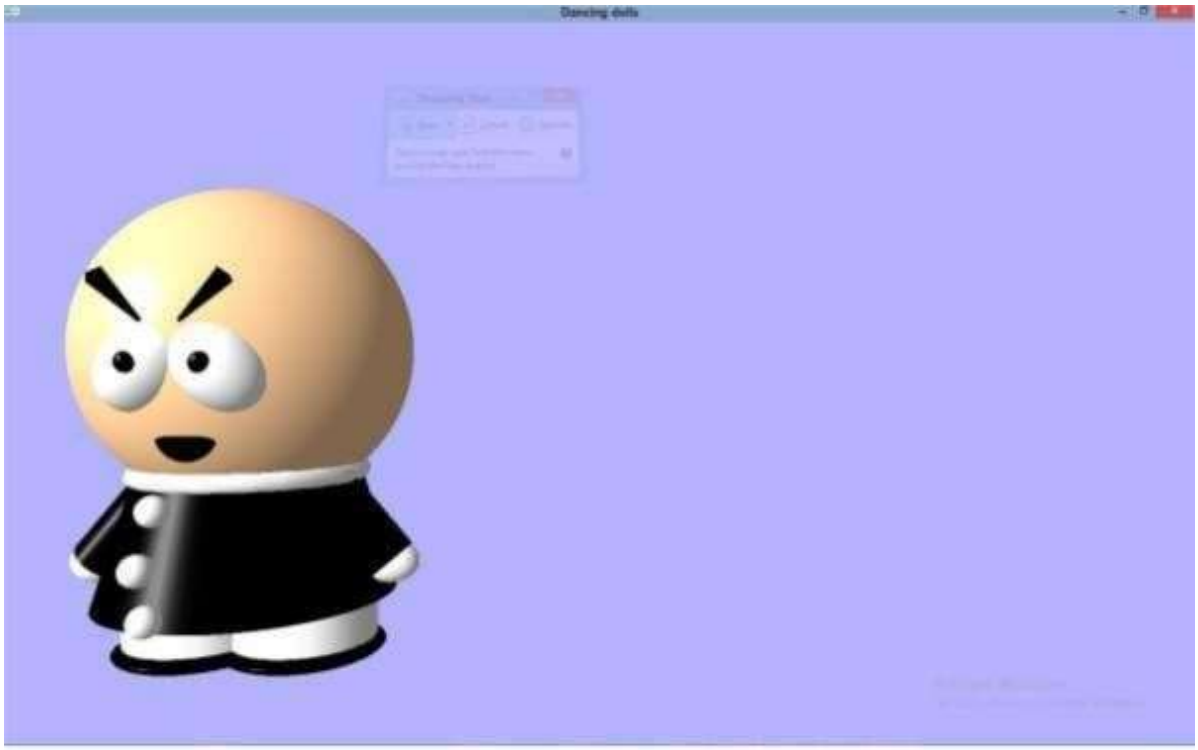
FUNCTION FOR CHANGE POSITION

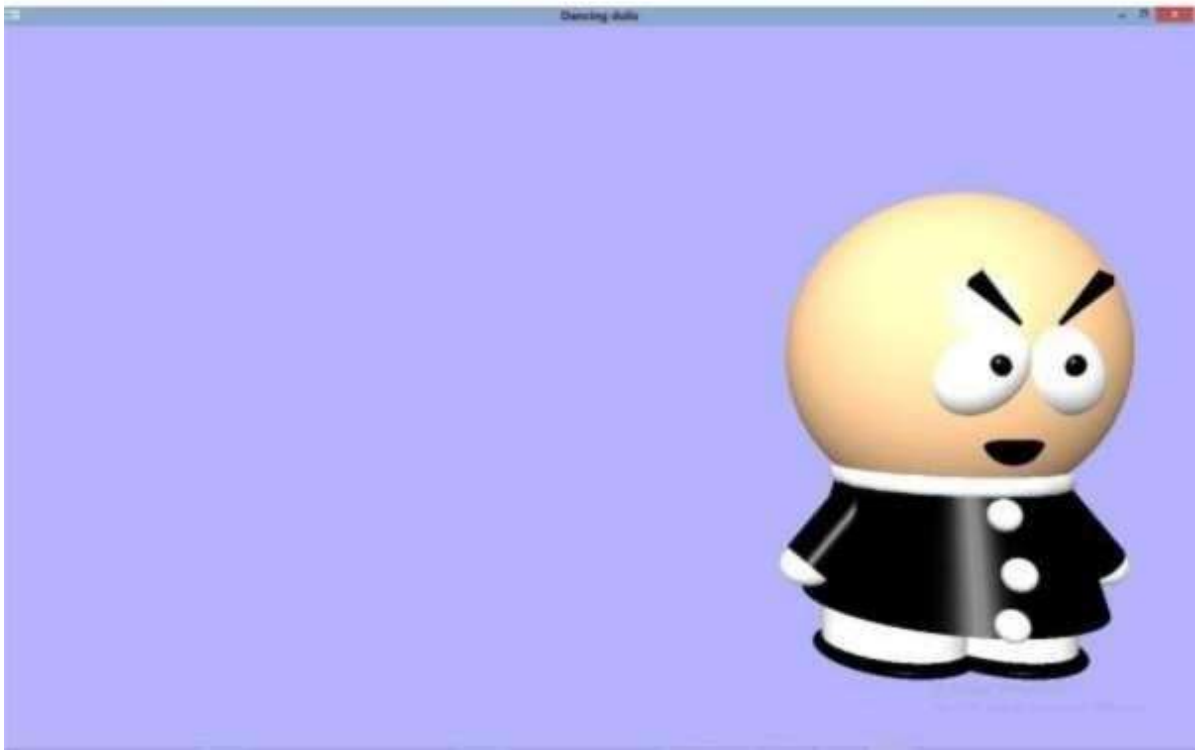
```
void walk(int key,int x,int y)
{
    if(key==GLUT_KEY_UP) walkY+=1;
    if(key==GLUT_KEY_DOWN) walkY-=1;
    if(key==GLUT_KEY_RIGHT) walkX+=1;
    if(key==GLUT_KEY_LEFT) walkX-=1;
    if(key==GLUT_KEY_F10) world=-world;
    if(key==GLUT_KEY_F9) doll=-doll;
```

```
}

void gaze(int x,int y)
{
   if((oldX<0) || (oldY<0))
         {
             oldX=x;
             oldY=y;
          }
   lookX+=y-oldY;lookY+=x-oldX;oldX=x;oldY=y;
}

void myReshape(int w, int h)
{
   GLfloat Ratio;
   glViewport(0,0,w,h);
   Ratio=1.0*w/h;
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(50.0,Ratio,VIEW_NEAR,VIEW_FAR);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslatef(INIT_VIEW_X,INIT_VIEW_Y,INIT_VIEW_Z);
   glLightfv(GL_LIGHT0, GL_POSITION, LightPos);
}

int main(int argc, char ** argv)
{
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
   glutCreateWindow("Dancing dolls");
   glutInitWindowSize(700,800);
  glutReshapeFunc(myReshape);
  glutDisplayFunc(Display);
  glutIdleFunc(Display);
  glutSpecialFunc(walk);
 // glutPassiveMotionFunc(gaze);
   SetupRend();
   glEnable(GL_NORMALIZE);
   glutMainLoop();
}
```

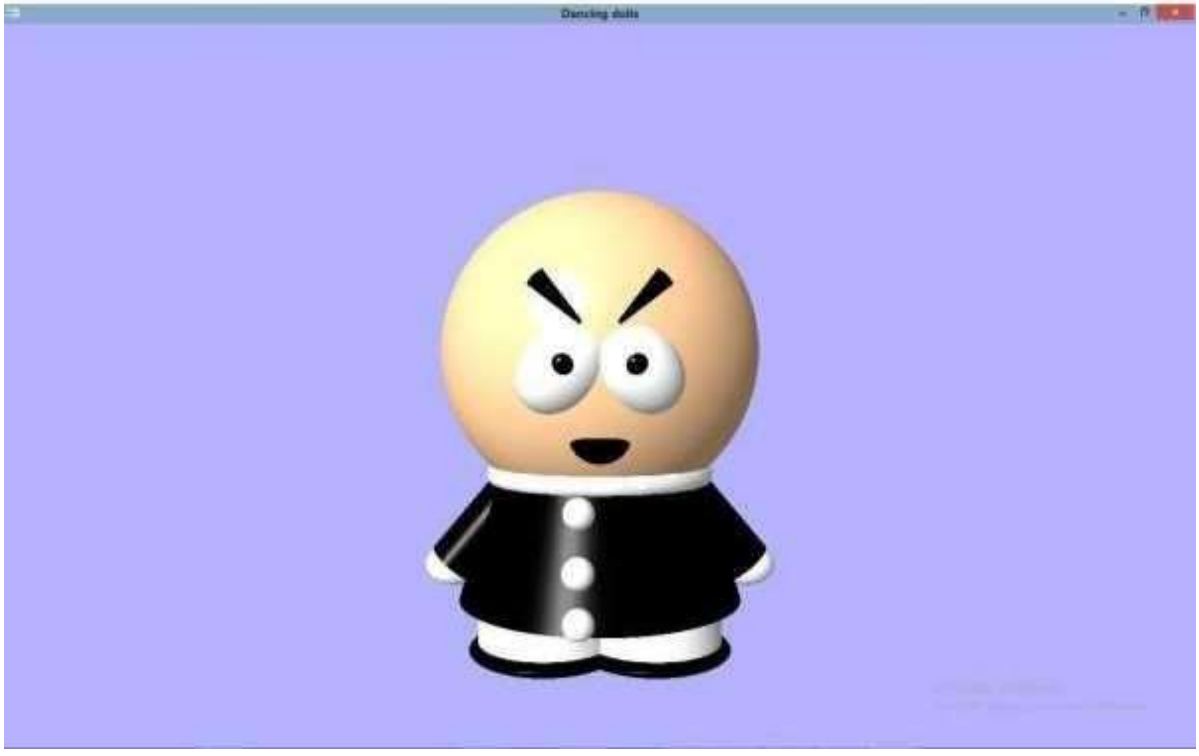
CHAPTER 6

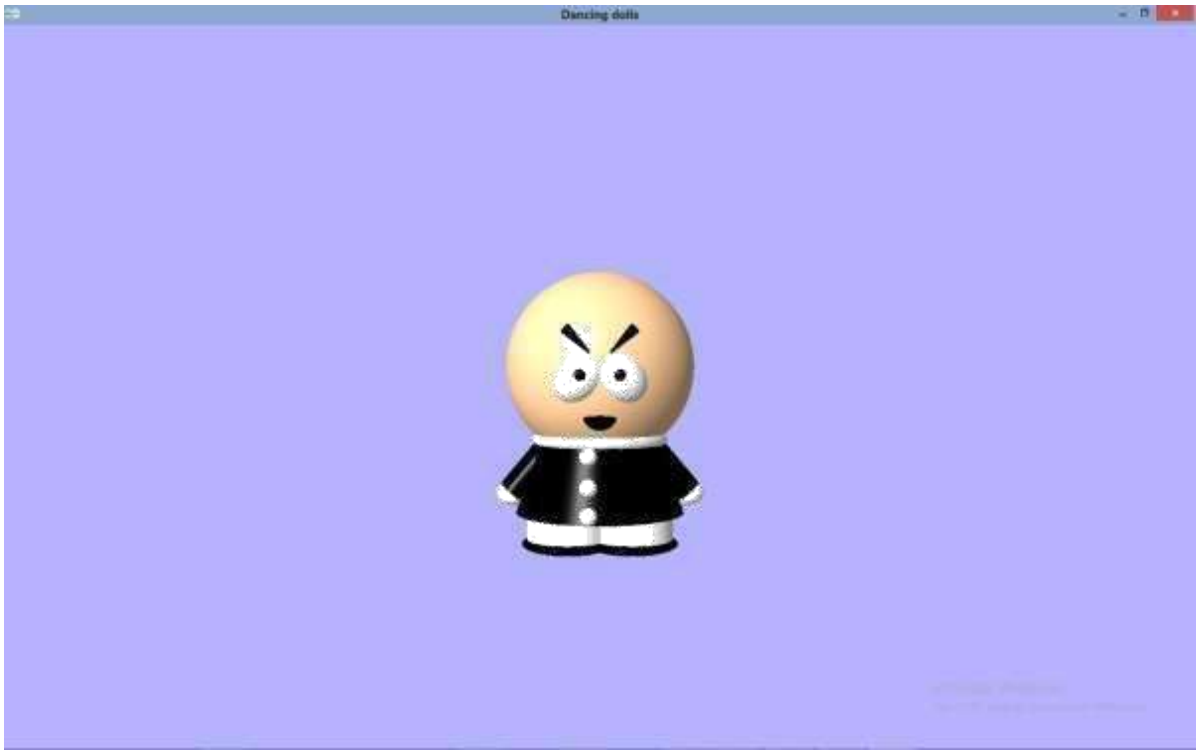
SNAPSHOTS






Dancing dolls


Dancing dolls


Dancing dolls

CHAPTER 7

CONCLUSION

This project was done to have an idea about Computer graphics using OpenGL. This project helped us to understand the concepts behind OpenGL and its programming. It also helped us to implement concepts in our project such as translation, drawing objects on the window screen using points, lines, polygon on the objects. This has given us a brief insight as to how programs, involving graphics, are written using OpenGL.

This project was a demonstration to show that OpenGL can be used to implement graphics which can be applied in various fields such advertising industries to endorse company's products animation studios to make animated films, cartoons; gaming industries by displaying use of high-end graphics in designing games; in engineering, architecture, medical fields by creating real-time models for better understanding, clarity and bringing out fresh, new ideas to enhance them.

Many improvements can be thought of to this project, such as including animation ,rotation to the doll. Care was taken to avoid bugs. Bugs may be reported to creator as the need may be. So, we conclude on note that we are looking forward to develop more such projects with an appetite in learning more in computer graphics.