TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION TO OPENGL

Computer graphics started with the display of data on hardcopy plotters and Cathode Ray Tube (CRT) screens soon after the introduction of computers themselves. It has grown to include the creation, storage and manipulation ofmodels and images of objects. These models come from a diverse and expandingset of fields and include physical, mathematical, engineering, architectural and even conceptual structures, natural phenomenon and so on.

1.1 Introduction to computer graphics

Computer graphics today is largely interactive: the user controls the contents, structure, and appearance of objects and of their displayed images by using input devices, such as a keyboard, mouse, or touch-sensitive panel on the screen. Because of the close relationship between the input devices and the display, the handling of such devices is included in the study of computer graphics.

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images).

OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer. The interface consists of over 250 different function calls, which can be used to draw complex three-dimensional scenes from simple primitives.

1.2 Uses of computer graphics

- **User interface:** It is now a well-established fact that graphical interfaces provide an alternative and easy interaction between users and computers. The built in graphics provided with user interfaces use the control items. In industry, business government and education organization’s computer graphics is most commonly used to create 2D and 3D graphs of mathematical, physical and economic functions in the form of histograms, bars and pie charts which are very useful in decision making Computer Aided Drafting and Design-In CAD, interactive graphics is used to design components and systems of mechanical, electrical and electronic devices including structures such as buildings, automobile bodies, aero planes, ship hulls etc.

- **Simulation and animation for scientific visualization and environment:**

Use of graphics in simulation makes mathematical models and mechanical systems more realistic and easy to study. The interactive graphics supported by animation software proved their use in production of animated movies and cartoon films.

1.3 OpenGL

OpenGL (Open Graphic Library) is a standard specification defining a crosslanguage cross platform API for writing application that produces 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex 3D scenes from simple primitives.OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization and flight simulation. It is also used in video games, where it competes with direct 3D on Microsoft Windows Platforms, OpenGL ismanaged by the nonprofit technology consortium, and the Khronos group Inc.

OpenGL Interface is nothing but interface between window system andOpenGL functions. Applications will be designed to access OpenGL directly through functions in three libraries. The OpenGL API is typically used to interact with a Graphics Processing Unit (GPU), to achieve the hardware accelerated rendering.

The OpenGL library organization consists of three main libraries given below [

- **GL Library:** Functions in the main GL library have names that beginwith the letters gl and are stored in a library usually referred to as GL.
- **GLU Library:** Stands for OpenGL Utility Library (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmer prefers not to write the code repeatedly. The functions mainly focus on primitive rendering and mapping between screen and world-coordinates, drawing of quadric surfaces, tessellation of polygonal primitives, interpretation of OpenGL error codes, an extended range of transformation routines for setting up viewing volumes and simple positioning of the camera, generally in more human-friendly terms than the routines presented by OpenGL.

GLU functions can be easily recognized by looking at them because they all have glu as a prefix.

- **GLUT Library**: Stands for OpenGL Utility Toolkit (GLUT). The main objective is to connect the graphics package with corresponding operating system. Functions performed include window definition, window control, monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives are also provided, including cubes, spheres. GLUT also has some limited support for creating pop-upmenus.

The two aims of GLUT are to allow the creation of rather portable code between operating system and to make learning OpenGL easier. Getting started with OpenGL programming while using GLUT often takes only a few lines of code and does not require knowledge of operating system. All GLUT functions start with glut prefix. The Figure 1.1 depicts the block diagram of OpenGL interface.

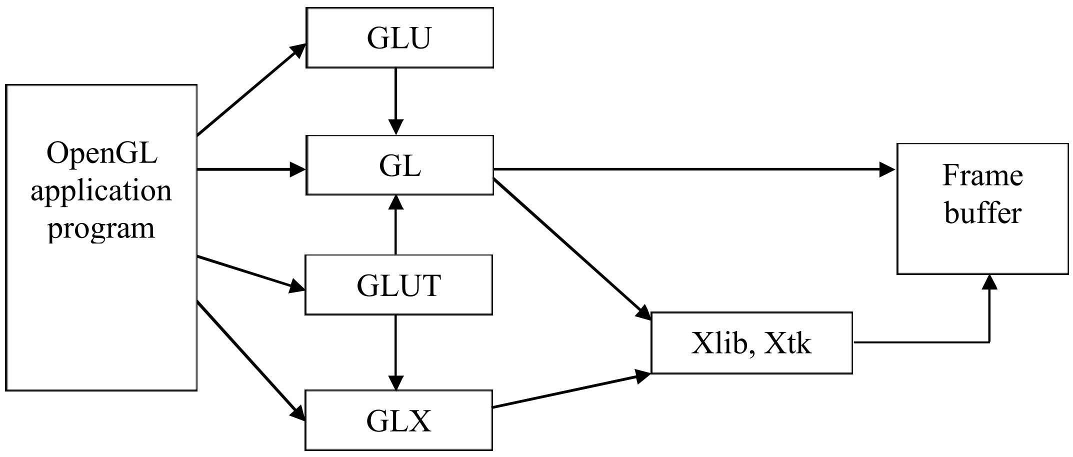

Figure 1.3 Block Diagram of OpenGL Interface.

OpenGL serves two main purposes :

- To hide the complexities of interfacing with different 3D accelerators, by presenting programmer with a single, uniform API
- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set.

OpenGL has historically been influential on the development of 3D accelerator, promoting abase level of functionality that is now common in consumer level hardware:

- Rasterized points, lines and polygons are basic primitives.
- A transform and lighting pipeline .
- Z buffering .
- Texture Mapping.
- Alpha Blending.

CHAPTER 2

MERGE SORT

MERGE SORT is an OpenGL program that demonstrates a simple illustration of how the merge sort algorithm works . OpenGL is a low-level graphics library specification. It makes available to the programmer a small set of geometric primitives-points, lines, polygons, images and bitmaps. In this program we have made use of such primitives for illustrating the lucky merge sort program.

2.1 Description

Merge Sort is an efficient, general-purpose, and comparison-based sorting algorithm. Most implementations produce a stable sort, which means that the order of equal elements is the same in the input and output. Merge sort is a divide-and-conquer algorithm that was invented by John von Neumann in 1945.

In this project we graphical represent the working of merge sort. At the beginning arrays or data blocks are divided into two halves until further division is not possible later they are sorted this whole process follows in recursive manner.

This project display the entire process occurring throughout the sorting event.

2.2 Working

- ➢ When the process starts all the blocks are randomly arranged.
- ➢ Blocks starts arranging according to merge sort algorithm.
- ➢ At the end of process the blocks are arranged to height.
- ➢ The entire process can slow down by pressing the key “-“.
- ➢ The entire process can gets faster by pressing the key “+“ .
- ➢ The process can zoomed in by pressing the key “A”.
- ➢ The process can zoomed out by pressing the key “Z”

CHAPTER 3

HARDWARE AND SOFTWARE REQUIREMENTS

In the development of any software application we require some particular system configuration of software and hardware components. This configuration helps in achieving the proper execution.

The various requirements that are essential for this project are specified over here. These requirements have to be fulfilled for successful of the project. The purpose, scope along with hardware and software requirements is given below

3.1 Hardware requirements:

The project works with any IBM PC compatibles, with Intel or AMD processors. A minimum of 1GB RAM is indispensable for smooth running of the package.

- Pentium or higher processor.
- 1GB with 2GHZ RAM.
- A standard keyboard, compatible mouse and a VGA monitor.
- Processor Speed is 800 MH
- Cache memory is 256 KB
- Diskette drive A : 1.44 MB, 3.5 inches

3.2 Software requirements:

For the purpose of this project, we decided to use C++ to code OpenGL. This graphics package has been designed for Windows Platform and uses CodeBlocks software.

➢ OS	: Windows7, Windows8, or other versions of Windows.
➢ Development Tool	: CodeBlocks
➢ Language	: C++

CHAPTER 4

DESIGN PHASE

Design is the creation of a plan or convention for the construction of an object or a system (as in architectural blueprints, engineering drawings, business processes, circuit diagrams and sewing patterns). Design has different connotations in different fields. In some cases the direct construction of an object (as in pottery, engineering, management, cowboy coding and graphic design) is also considered to be design. A specification of an object, manifested by anagent, intended to accomplish goals, in a particular environment, using a set of primitive components, satisfying a set of requirements, subject to constraints. Another definition for design is a roadmap or a strategic approach for someone to achieve a unique expectation. It defines the specifications, plans, parameters, costs, activities, processes and how and what to do within legal, political, social, environmental, safety and economic constraints in achieving that objective. Designing often necessitates considering the aesthetic, functional, economic and sociopolitical dimensions of both the design object and design process. It may involve considerable research, thought, modeling, interactive adjustment, and re-design. Meanwhile, diverse kinds of objects may be designed, including clothing, graphical user interfaces, skyscrapers, corporate identities, business processes and even methods of designing.

4.1 Flowchart

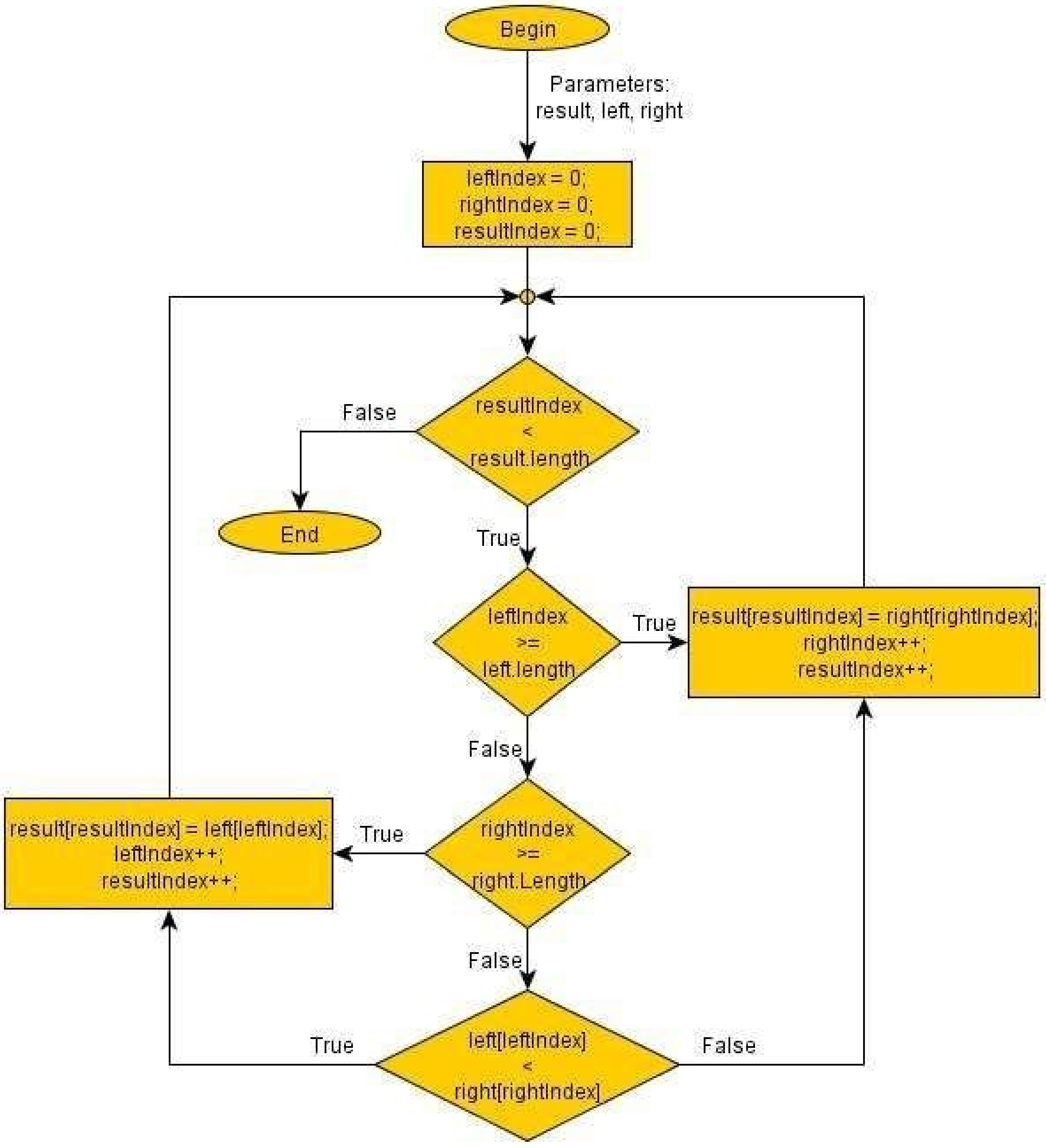
Begin
Parameters: result, left, right
leftIndex = 0; rightIndex = 0; resultIndex = 0;
resultIndex < result.length
False
End
True
leftIndex >= left.length
True
result[resultIndex] = right[rightIndex]; rightIndex++; resultIndex++;
False
rightIndex >= right.Length
True
result[resultIndex] = left[leftIndex]; leftIndex++; resultIndex++;
False
left[leftIndex] < right[rightIndex]
True
False

Fig 4.1: Flow of the program

4.2 Algorithm

```
Step 1: start
Step 2: declare array and left, right, mid variable
Step 3: perform merge function.
            mergesort(array,left,right)
            mergesort (array, left, right)
            if left > right
            return
            mid= (left+right)/2
           MergeSort(array, left, mid)
           MergeSort(array, mid+1, right)
           MergeSort(array, left, mid, right)
step 4: Stop
```

4.3 Pseudocode :

Declare left variable to 0 and right variable to n-1

- Find mid by medium formula. mid = (left+right)/2
- Call merge sort on (left,mid)
- Call merge sort on (mid+1,rear)
- Continue till left is less than right
- Then call merge function to perform merge sort.

CHAPTER 5

FUNCTIONS

A function is a group of statements that together perform a task. The code can be divided into separate functions. How the code is divided among different functions is based on the programmer's requirement, but logically the division usually is so each function performs a specific task.

5.1 Description

GL primitives can have either flat or smooth shading. Smooth shading, the default, causes the computed colors of vertices to be interpolated as the primitive is rasterized typically assigning different colors to each resulting pixel fragment. Flat shading selects the computed color of just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive[3]. A function is something that associates each element of a set with an element of another set (which may or may not be the same as the first set). The concept of function appears quite often even in nontechnical contexts.

C graphics using glut.h functions can be used to draw different shapes, display text in different fonts, change colors and many more. Using functions of glut.h in turbo c compiler you can make graphics programs, animations, projects and games. You can draw circles, lines, rectangles, bars and many other geometrical figures. You can change their colors using the available functions and fill them. Following is a list of functions of glut.h header file. Every function is discussed with the arguments it needs, its description, possible errors while using that function and a sample c graphics program with its output.

5.2 Key Functions

Important functions used in our Colors Of Nature program are listed below:

5.2.1 Inbuilt functions

Open GL functions used in the code are as follows.

- **glBegin () :**
  Specifies the primitive or primitives that will be created form vertices presented between glBegin and the subsequent

- glEnd**glutAddMenuEntry() :**
  Adds an entry with the string name displayed to the current menu. Value is returned to the menu callback when the entry is selected.

- **glutAttachMenu() :**
  Attaches the current menu to the specified mouse button.

- **glutCreateMenu( )** :
  Returns an identifier for a top-level menu and registers the callback function that returns an integer value corresponding to the menu entry selected.

- **glutPostRedispla** :
  This function marks the *current window* as needing to be redisplayed.

- **glPointSize(float size) :**
  It specifies the rasterized diameterof both aliased and antialiased points.

- **glLineWidth**( ) :
  Specifies the rasterized width of both aliased and antialiased lines. Using a line width other than 1 has different effects, depending on whether line antialiasing is enabled.

- **glutSwapBuffers**( ) :

  Swaps the buffers of the *current window* if doublebuffered.

- **glutDisplayFunc( ) :**

  It sets the display callback forthe current window.

- **glTranslate() ;**

  Alters the current matrix by a displacement of (x,y,z).

- **glLight[if]v() :**

  This sets scalar vector parameters for light source light.

CHAPTER 6

IMPLEMENTATION

Implementation is nothing but the process of putting a decision or plan into effect; execution. Implementation includes code of the happy child, in which the code for happy child is described, and code is written for a child who wants to play basketball, sliding, track run but only when his father says yes to his child.The code shows the movement of child from home to playground. At first the background shows interaction between father and the child near their home and a road which leads to playground.

6.1 Platform

Windows7 is a personal computer operating system developed by Microsoft,a version of Windows NT. Development of 7 occurred as early as 2006 underthe codename "**Blackcomb**". Windows 7 is available in six different editions, of which the Home Premium, Professional, and Ultimate editions are available for retail sale to consumers.

The advantages of windows 7 are:

- Improved performance.
- Enhanced searching capabilities.
- Location aware printing.
- Virtual hard disk support.
- Expanded security.
- Get better security built-in.
- Hassle-free backups.
- Have all your files instantly at hand.

6.2 Language

C++ is a general purpose programming language that is free-form and compiled. It is regarded as an intermediate-level language, as it comprises both high- level and low-level language features. It provides imperative, generic programming features. It is implemented on a wide variety of hardware and operating system platforms. .

6.3 Code

```
#include <GL/glut.h>
#include <iostream>
#define WIDTH  600
#define HEIGHT  600using namespace std;
double rotationX = 20.0; // Variables that control application rotation
double rotationY = 20.0;
int last_press_x = 0;
int last_press_y = 0;
double pos_z[15]; // Vectors responsible for saving the positions of the bars on the 3 axes
double pos_x[15];
double pos_y[15];
int rotY[15];
float color1[15]; // Vectors responsible for the coloring of the bars - RGB
float color2[15];
float color3[15];
bool stop = false; // Variable responsible for controlling the stop in Animation
int velocity = 50;
int step = 1; // Variable that controls the steps within the animation
int eyeCamera = 40;
void ParametersLighting()
{
GLfloat ambientlight[4]={0.8, 0.8, 0.8, 1.0};
GLfloat Diffusedlight[4]={1.0, 0.0, 0.4, 1.0};
GLfloat lightSpecific[4]={1, 1, 0.6, 1.0};
GLfloat positionLight[4]={-1.0,1.0, 1.0, 1.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, ambientlight);
glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpecific );
glLightfv(GL_LIGHT0, GL_POSITION, positionLight );
glLightfv(GL_LIGHT0, GL_POSITION, Diffusedlight );
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientlight);
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, positionLight);
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lightSpecific);
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, Diffusedlight);
}
void Draw_Bar(void){
glBegin(GL_QUADS);
glVertex3f( 0.2f, 0.2f,-0.2f);    // Top face
glVertex3f(-0.2f, 0.2f,-0.2f);
glVertex3f(-0.2f, 0.2f, 0.2f);
glVertex3f( 0.2f, 0.2f, 0.2f);
```

```
glVertex3f( 0.2f,-0.2f, 0.2f);    // Bottom face
glVertex3f(-0.2f,-0.2f, 0.2f);
glVertex3f(-0.2f,-0.2f,-0.2f);
glVertex3f( 0.2f,-0.2f,-0.2f);
glVertex3f( 0.2, 0.2, 0.2);    // Front face
glVertex3f(-0.2, 0.2, 0.2);
glVertex3f(-0.2,-0.2, 0.2);
glVertex3f( 0.2,-0.2, 0.2);
glVertex3f( 0.2f,-0.2f,-0.2f);    // Back face
glVertex3f(-0.2f,-0.2f,-0.2f);
glVertex3f(-0.2f, 0.2f,-0.2f);
glVertex3f( 0.2f, 0.2f,-0.2f);
glVertex3f(-0.2f, 0.2f, 0.2f);    // Left face
glVertex3f(-0.2f, 0.2f,-0.2f);
glVertex3f(-0.2f,-0.2f,-0.2f);
glVertex3f(-0.2f,-0.2f, 0.2f);
glVertex3f( 0.2f, 0.2f,-0.2f);    // Right face
glVertex3f( 0.2f, 0.2f, 0.2f);
glVertex3f( 0.2f,-0.2f, 0.2f);
glVertex3f( 0.2f,-0.2f,-0.2f);
glEnd();
}
void Draw_Plane(void){ // Drawing the plane below the bars and their colored lines that
define their phases
glBegin(GL_QUADS);
glColor3f(0.0f,0.0f,0.2f);
glVertex3f(30.0f, 0.0f, 30.0f);
glVertex3f(-30.0f, 0.0f, 30.0f);
glVertex3f(-30.0f,0.0f,-30.0f);
glVertex3f( 30.0f,0.0f,-30.0f);
glEnd();
glColor3f(0.0, 1.0, 0.0);
glBegin(GL_LINES);
glVertex3i(0, 0, 0);
glVertex3i(WIDTH/2, 0, 0);
glEnd();
glColor3f(0.0, 1.0, 0.0);
glBegin(GL_LINES);
glVertex3i(0, 0, 0);
glVertex3i(-WIDTH/2, 0, 0);
glEnd();
glColor3f(1, 0.0, 0.0);
```

```
glBegin(GL_LINES);
glVertex3i(0, 0, 2);
glVertex3i(WIDTH/2, 0, 2);
glEnd();
glColor3f(1, 0, 0);
glBegin(GL_LINES);
glVertex3i(0, 0, 2);
glVertex3i(-WIDTH/2, 0, 2);
glEnd();
glColor3f(1, 1.0, 0.0);
glBegin(GL_LINES);
glVertex3i(0, 0, 4);
glVertex3i(WIDTH/2, 0, 4);
glEnd();
glColor3f(1, 1, 0);
glBegin(GL_LINES);
glVertex3i(0, 0, 4);
glVertex3i(-WIDTH/2, 0, 4);
glEnd();
glColor3f(0, 1.0, 1.0);
glBegin(GL_LINES);
glVertex3i(0, 0, 6);
glVertex3i(WIDTH/2, 0, 6);
glEnd();
glColor3f(0, 1, 1);
glBegin(GL_LINES);
glVertex3i(0, 0, 6);
glVertex3i(-WIDTH/2, 0, 6);
glEnd();
}
void Draw(void) // Function that draws the components of the application: bars, planes, lines
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0.0, 0.0, eyeCamera,0.0, 0.0, -30.0,0.0, 1.0, 0.0);
ParametersLighting();
glRotatef(rotationY, 1.0, 0.0, 0.0);
glRotatef(rotationX, 0.0, 1.0, 0.0);
Draw_Plane();
glPushMatrix();
glScalef(1.0, 6.0, 1.0);
```

```
glTranslatef(pos_x[0], 0.2, pos_z[0]);
glRotatef(rotY[0],0, 10,0); // bar rotation angle in the change animation
glColor3d(color1[0], color2[0], color3[0]); // bar color
Draw_Bar(); // Bar 0
glPopMatrix();
glPushMatrix();
glScalef(1.0, 4.0, 1.0);
glTranslatef(pos_x[1], 0.2, pos_z[1]);
glRotatef(rotY[1],0, 10,0);
glColor3d(color1[1], color2[1], color3[1]);
Draw_Bar(); // Bar 1
glPopMatrix();
glPushMatrix();
glScalef(1.0, 1.0, 1.0);
glTranslatef(pos_x[2], 0.2, pos_z[2]);
glRotatef(rotY[2],0, 10,0); glColor3d(color1[2], color2[2], color3[2]);
Draw_Bar(); // Bar 2
glPopMatrix();
glPushMatrix();
glScalef(1.0, 7.0, 1.0);
glTranslatef( pos_x[3], 0.2, pos_z[3]);
glRotatef(rotY[3],0, 10,0);
glColor3d(color1[3], color2[3], color3[3]);
Draw_Bar(); // Bar 3
glPopMatrix();
glPushMatrix();
glScalef(1.0, 3.0, 1.0);
glTranslatef(pos_x[4], 0.2, pos_z[4]);
glRotatef(rotY[4],0, 10,0);
glColor3d(color1[4], color2[4], color3[4]);
Draw_Bar(); // Bar 4
glPopMatrix();
glPushMatrix();
glScalef(1.0, 10.0, 1.0);
glTranslatef(pos_x[5], 0.2, pos_z[5]);
glRotatef(rotY[5],0, 10,0);
glColor3d(color1[5], color2[5], color3[5]);
Draw_Bar(); // Bar 5
glPopMatrix();
glPushMatrix();
glScalef(1.0, 5.0, 1.0);
glTranslatef(pos_x[6], 0.2, pos_z[6]);
```

```
glRotatef(rotY[6],0, 10,0);
glColor3d(color1[6], color2[6], color3[6]);
Draw_Bar(); // Bar 6
glPopMatrix();
glPushMatrix();
glScalef(1.0, 10.0, 1.0);
glTranslatef(pos_x[7], 0.2, pos_z[7]);
glRotatef(rotY[7],0, 10,0);
glColor3d(color1[7], color2[7], color3[7]);
Draw_Bar(); // Bar 7
glPopMatrix();
glPushMatrix();
glScaled(1, 4.0, 1.0);
glTranslatef(pos_x[8], 0.2, pos_z[8]);
glRotatef(rotY[8],0, 10,0);
glColor3d(color1[8], color2[8], color3[8]);
Draw_Bar(); // Bar 8
glPopMatrix();
glPushMatrix();
glScalef(1.0, 10.0, 1.0);
glTranslatef(pos_x[9],
0.2,pos_z[9]);
glRotatef(rotY[9],0, 10,0);
glColor3d(color1[9],
color2[9], color3[9]);
Draw_Bar(); // Bar 9
glPopMatrix();
glPushMatrix();
glScalef(1.0, 5.0, 1.0);
glTranslatef(pos_x[10], 0.2, pos_z[10]);
glRotatef(rotY[10],0, 10,0);
glColor3d(color1[10], color2[10], color3[10]);
Draw_Bar(); // Bar 10
glPopMatrix();
glPushMatrix();
glScalef(1.0, 8.0, 1.0);
glTranslatef(pos_x[11], 0.2, pos_z[11]);
glRotatef(rotY[11],0, 10,0);
glColor3d(color1[11], color2[11], color3[11]);
Draw_Bar(); // Bar 11
glPopMatrix();
glPushMatrix();
```

```
glScalef(1.0, 7.0, 1.0);
glTranslatef(pos_x[12], 0.2, pos_z[12]);
glRotatef(rotY[12],0, 10,0);
glColor3d(color1[12], color2[12], color3[12]);
Draw_Bar(); // Bar 12
glPopMatrix();
glPushMatrix();
glScalef(1.0, 9.0, 1.0);
glTranslatef(pos_x[13], 0.2, pos_z[13]);
glRotatef(rotY[13],0, 10,0);
glColor3d(color1[13], color2[13], color3[13]);
Draw_Bar(); // Bar 13
glPopMatrix();
glPushMatrix();
glScalef(1.0, 12.0, 1.0);
glTranslatef(pos_x[14], 0.2, pos_z[14]);
glRotatef(rotY[14],0, 10,0);
glColor3d(color1[14], color2[14], color3[14]);
Draw_Bar(); // Bar 14
glPopMatrix();
glPushMatrix();
glScalef(1.0, 13.0, 1.0);
glTranslatef(pos_x[15], 0.2, pos_z[15]);
glRotatef(rotY[15],0, 10,0);
glColor3d(color1[15], color2[15], color3[15]);
Draw_Bar(); // Bar 15
glPopMatrix();
glFlush();
}
void Anim(int value) //In this function we have the variable "step" defining each stage of the
animation
{
if( pos_z[0]<0 && step == 1) // Bringing all bars from position z-4 to position z0
{
for(int i = 0; i<16; i++)
{

pos_z[i] += 0.05;
}
if(pos_z[15] >= 0)
{
step = 2;
}
}
if(pos_z[0]<2 && step == 2)
{
for(int i = 0; i<8; i++)
{
pos_z[i] +=0.05;
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
if(pos_z[0] >= 2)
{
step = 3;
}
}
if(pos_z[0]<4 && step == 3)
{
for(int i = 0; i<4; i++)
{
pos_z[i] +=0.05;
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[0] >= 4)
{
step = 4;
}
}
if(pos_z[0]<6 && step == 4)
{
for(int i = 0; i<2; i++)
{
pos_z[i] +=0.05;
color1[i] = 0;
color2[i] = 1;
color3[i] = 1;
}
if(pos_z[0] >= 6)
{
step = 5;
}
}
```

```
if(pos_x[0]<-7.0 && step == 5)
{
pos_x[0] += 0.02;
pos_x[1] -= 0.02;

if(rotY[0]<10000)
{
rotY[0] += 20;
rotY[1] += 20;
}
for(int i = 0; i<2; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
if(pos_x[0]>=-7)
{
step = 6;
rotY[0] = 0;
rotY[1] = 0;
for(int i = 0; i<2; i++)
{
color1[i] = 0;
color2[i] = 1;
color3[i] = 1;
}
}
}
if(pos_z[0]>4 && step == 6)
{
pos_z[0] -= 0.05;
pos_z[1] -= 0.05;
for(int i = 0; i<2; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[0]<=4)
{
step = 7;
}
}
```

```
if(pos_z[2]<6 && step == 7)
{
for(int i = 2; i<4; i++)
{
pos_z[i] +=0.05;
color1[i] = 0;
color2[i] = 1;
color3[i] = 1;
}
if(pos_z[2] >= 6)
{
step = 8;
}
}
if(pos_z[2]>4 && step == 8)
{
pos_z[2] -= 0.05;
pos_z[3] -= 0.05;
for(int i = 2; i<4; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[2]<=4)
{
step = 9;
}
}
if(pos_x[1]<-6.0  && step == 9)
{
pos_x[1] += 0.02;
pos_x[2] -= 0.02;
rotY[1] += 15;
rotY[2] += 15;
for(int i = 1; i<3; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
if(pos_x[1]>=-6.0)
{
```

```
step = 10;
rotY[1] = 0;
rotY[2] = 0;
for(int i = 1; i<3; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
}
}
if(pos_x[0]<-6.0  && step == 10)
{
pos_x[0] += 0.02;
pos_x[1] -= 0.02;
rotY[0] += 15;
rotY[1] += 15;

for(int i = 0; i<2; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
if(pos_x[0]>=-6.0)
{
step = 11;
rotY[0] = 0;
rotY[1] = 0;
for(int i = 0; i<2; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
}
}
if(pos_z[0]>2.0 && step == 11)
{
for(int i = 0; i<4; i++)
{
pos_z[i] -=0.05;
color1[i] = 1;
```

```
color2[i] = 0;
color3[i] = 0;
}
if(pos_z[0]<= 2)
{
step = 12;
}
}
if(pos_z[4]<4.0 && step == 12)
{
for(int i = 4; i<8; i++)
{
pos_z[i] +=0.05;
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[4]>= 4)
{
step = 13;
}
}

if(pos_z[4]<6.0 && step == 13)
{
pos_z[4] +=0.05;
pos_z[5] +=0.05;
for(int i = 4; i<6; i++)
{
color1[i] = 0;
color2[i] = 1;
color3[i] = 1;
}
if(pos_z[4]>= 6)
{
step = 14;
}
}
if(pos_z[4]>4.0 && step == 14)
{
pos_z[4] -=0.05;
pos_z[5] -=0.05;
for(int i = 4; i<6; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
```

```
if(pos_z[4]<=4)
{
step = 15;
}
}
if(pos_z[6]<6.0 && step == 15)
{
pos_z[6] +=0.05;
pos_z[7] +=0.05;
for(int i = 6; i<8; i++)
{
color1[i] = 0;
color2[i] = 1;
color3[i] = 1;
}
if(pos_z[6]>=6)
{
step = 16;
}
}
if(pos_z[6]>4.0 && step == 16)
{
pos_z[6] -=0.05;
pos_z[7] -=0.05;
for(int i = 6; i<8; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[6]<=4)
{
step = 17;
}
}
if(pos_x[5]<-2.0  && step == 17)
{
pos_x[5] += 0.02;
pos_x[6] -= 0.02;
rotY[5] += 15;
rotY[6] += 15;
for(int i = 5; i<7; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
if(pos_x[5]>=-2.0)
```

```
{
step = 18;
rotY[5] = 0;
rotY[6] = 0;
for(int i = 5; i<7; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
}
}
if(pos_z[4]>2.0 && step == 18)
{
for(int i = 4; i<8; i++)
{
pos_z[i] -=0.05;
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
if(pos_z[4]<= 2)
{
step = 19;
}
}
if(pos_x[1]<-4.0  && step == 19)
{
pos_x[1] += 0.02;
pos_x[4] -= 0.02;
rotY[1] += 15;
rotY[4] += 15;

for(int i = 1; i<5; i++)
{
if(i==1 || i==4)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
}
if(pos_x[1]>=-4.0)
{
step = 20;
rotY[1] = 0;
rotY[4] = 0;
for(int i = 1; i<5; i++)
```

```
{
if(i==1 || i==4)
{
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
}
}
}
if(pos_x[0]<-4.0  && step == 20)
{
pos_x[0] += 0.02;
pos_x[1] -= 0.02;
rotY[0] += 15;
rotY[1] += 15;

for(int i = 0; i<2; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
if(pos_x[0]>=-4.0)
{
step = 21;
rotY[0] = 0;
rotY[1] = 0;
for(int i = 0; i<2; i++)
{
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
}
}
if(pos_x[3]<-3.0  && step == 21)
{
pos_x[3] += 0.02;
pos_x[6] -= 0.02;
rotY[3] += 15;
rotY[6] += 15;

for(int i = 3; i<7; i++)
{
if(i==3 || i==6)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
}
if(pos_x[3]>=-3.0)
{
step = 22;
rotY[3] = 0;
rotY[6] = 0;
for(int i = 3; i<7; i++)
{
if(i==3 || i==6)
{
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
}
}
}
if(pos_z[0]>0  && step == 22)
{
for(int i = 0; i<8; i++)
{
pos_z[i] -=0.05;
color1[i] = 0;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[0]<=0)
{
step = 23;
}
}
if(pos_z[8]<2  && step == 23)
{
for(int i = 8; i<16; i++)
{
pos_z[i] +=0.05;
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
if(pos_z[8]>=2)
{
step = 24;
}
}
```

```
if(pos_z[8]<4  && step == 24)
{
for(int i = 8; i<12; i++)
{
pos_z[i] +=0.05;
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[8]>=4)
{
step = 25;
}
}
if(pos_z[8]<6  && step == 25)
{
for(int i = 8; i<10; i++)
{
pos_z[i] +=0.05;
color1[i] = 0;
color2[i] = 1;
color3[i] = 1;
}
if(pos_z[8]>=6)
{
step = 26;
}
}
if(pos_z[8]>4  && step == 26)
{
for(int i = 8; i<10; i++)
{
pos_z[i] -=0.05;
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[8]<=4)
{
step = 27;
}
}
if(pos_z[10]<6  && step == 27)
{
for(int i = 10; i<12; i++)
{
pos_z[i] +=0.05;
color1[i] = 0;
color2[i] = 1;
color3[i] = 1;
}
if(pos_z[10]>=6)
{
step = 28;
}
}
if(pos_z[10]>4  && step == 28)
{
for(int i = 10; i<12; i++)
{
pos_z[i] -=0.05;
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[10]<=4)
{
step = 29;
}
}
if(pos_x[9]<2  && step == 29)
{
pos_x[9] += 0.02;
pos_x[10] -= 0.02;
rotY[9] += 15;
rotY[10] += 15;
for(int i = 9; i<11; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
if(pos_x[9]>=2)
{
step = 30;
rotY[9] = 0;
rotY[10] = 0;
for(int i = 9; i<11; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
}
}
if(pos_x[9]<3  && step == 30)
```

```
{
pos_x[9] += 0.02;
pos_x[11] -= 0.02;
rotY[9] += 15;
rotY[11] += 15;
for(int i = 9; i<12; i++)
{
if(i==9 || i==11)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
}
if(pos_x[9]>=3)
{
step = 31;
rotY[9] = 0;
rotY[10] = 0;
for(int i = 9; i<12; i++)
{
if(i==9 || i==11)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
}
}
}
if(pos_z[8]>2  && step == 31)
{
for(int i = 8; i<12; i++)
{
pos_z[i] -=0.05;
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
if(pos_z[8]<=2)
{
step = 32;
}
}
if(pos_z[12]<4  && step == 32)
{
for(int i = 12; i<16; i++)
{
```

```
pos_z[i] +=0.05;
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[12]>=4)
{
step = 33;
}
}
if(pos_z[12]<6  && step == 33)
{
for(int i = 12; i<14; i++)
{
pos_z[i] +=0.05;
color1[i] = 0;
color2[i] = 1;
color3[i] = 1;
}
if(pos_z[12]>=6)
{
step = 34;
}
}
if(pos_z[12]>4  && step == 34)
{
for(int i = 12; i<14; i++)
{
pos_z[i] -=0.05;
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[12]<=4)
{
step = 35;
}
}
if(pos_z[14]<6  && step == 35)
{
for(int i = 14; i<16; i++)
{
pos_z[i] +=0.05;
color1[i] = 0;
color2[i] = 1;
color3[i] = 1;
}
if(pos_z[14]>=6)
```

```
{
step = 36;
}
}
if(pos_z[14]>4  && step == 36)
{
for(int i = 14; i<16; i++)
{
pos_z[i] -=0.05;
color1[i] = 1;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[14]<=4)
{
step = 37;
}
}
if(pos_z[12]>2  && step == 37)
{
for(int i = 12; i<16; i++)
{
pos_z[i] -=0.05;
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
if(pos_z[12]<=2)
{
step = 38;
}
}
if(pos_x[11]<4  && step == 38)
{
pos_x[11] += 0.02;
pos_x[12] -= 0.02;
rotY[11] += 15;
rotY[12] += 15;
for(int i = 11; i<13; i++)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
if(pos_x[11]>=4)
{
step = 39;
rotY[11] = 0;
```

```
rotY[12] = 0;
for(int i = 11; i<13; i++)
{
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
}
}
if(pos_x[9]<4  && step == 39)
{
pos_x[9] += 0.02;
pos_x[11] -= 0.02;
rotY[9] += 15;
rotY[11] += 15;
for(int i = 9; i<12; i++)
{
if(i==9 || i==11)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
}
if(pos_x[9]>=4)
{
step = 40;
rotY[9] = 0;
rotY[11] = 0;
for(int i = 9; i<12; i++)
{
if(i==9 || i==11)
{
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
}
}
}
if(pos_x[9]<5  && step == 40)
{
pos_x[9] += 0.02;
pos_x[13] -= 0.02;
rotY[9] += 15;
rotY[13] += 15;
for(int i = 9; i<14; i++)
{
```

```
if(i==9 || i==13)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
}
if(pos_x[9]>=5)
{
step = 41;
rotY[9] = 0;
rotY[13] = 0;
for(int i = 9; i<14; i++)
{
if(i==9 || i==13)
{
color1[i] = 1;
color2[i] = 0;
color3[i] = 0;
}
}
}
}
if(pos_z[8]>0  && step == 41)
{
for(int i = 8; i<16; i++)
{
pos_z[i] -=0.05;
color1[i] = 0;
color2[i] = 1;
color3[i] = 0;
}
if(pos_z[8]<=0)
{
step = 42;
}
}
if(pos_x[6]<0  && step == 42)
{
pos_x[6] += 0.02;
pos_x[8] -= 0.02;
rotY[6] += 15;
rotY[8] += 15;
for(int i = 6; i<9; i++)
{
if(i==6 || i==8)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
}
if(pos_x[6]>=0)
{
step = 43;
rotY[6] = 0;
rotY[8] = 0;
for(int i = 6; i<9; i++)
{
if(i==6 || i==8)
{
color1[i] = 0;
color2[i] = 1;
color3[i] = 0;
}
}
}
}
if(pos_x[0]<0  && step == 43)
{
pos_x[0] += 0.02;
pos_x[6] -= 0.02;
rotY[0] += 15;
rotY[6] += 15;
for(int i = 0; i<7; i++)
{
if(i==0 || i==6)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
}
if(pos_x[0]>=0)
{
step = 44;
rotY[0] = 0;
rotY[6] = 0;
for(int i = 0; i<7; i++)
{
if(i==0 || i==6)
{
color1[i] = 0;
color2[i] = 1;
color3[i] = 0;
}
}
}
```

```
}
if(pos_x[3]<1  && step == 44)
{
pos_x[3] += 0.02;
pos_x[10] -= 0.02;
rotY[3] += 15;
rotY[10] += 15;
for(int i = 3; i<11; i++)
{
if(i==3 || i==10)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
}
if(pos_x[3]>=1)
{
step = 45;
rotY[3] = 0;
rotY[10] = 0;
for(int i = 3; i<11; i++)
{
if(i==3 || i==10)
{
color1[i] = 0;
color2[i] = 1;
color3[i] = 0;
}
}
}
}
if(pos_x[5]<0  && step == 45)
{
pos_x[5] += 0.02;
pos_x[0] -= 0.02;
rotY[5] += 15;
rotY[0] += 15;
for(int i = 0; i<6; i++)
{
if(i==0 || i==5)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
}
if(pos_x[5]>=0)
{
step = 46;
rotY[5] = 0;
rotY[0] = 0;
for(int i = 0; i<6; i++)
{
if(i==0 || i==5)
{
color1[i] = 0;
color2[i] = 1;
color3[i] = 0;
}
}
}
}
if(pos_x[7]<1  && step == 46)
{
pos_x[7] += 0.02;
pos_x[3] -= 0.02;
rotY[7] += 15;
rotY[3] += 15;
for(int i = 3; i<8; i++)
{
if(i==3 || i==7)
{
color1[i] = 1;
color2[i] = 1;
color3[i] = 1;
}
}
if(pos_x[7]>=1)
{
step = 47;
rotY[7] = 0;
rotY[3] = 0;
for(int i = 3; i<8; i++)
{
if(i==3 || i==7)
{
color1[i] = 0;
color2[i] = 1;
color3[i] = 0;
}
}
}
}

}
static void key(unsigned char key, int x, int y) // Function responsible for the interaction of
the keyboard with the application
{
switch (key)
{
```

```
case 'e':
exit(0);
break;
case '+': // increasing velocity
if(velocity>10){
velocity -= 10;
}else{
velocity--;
}
break;
case '-': // decreasing velocity
if(velocity>10){
velocity += 10;
}else{
velocity++;
}
break;
case 'a': // increasing eye distance
eyeCamera++;
break;
case 'd': // decreasing eye distance
eyeCamera--;
break;
glutPostRedisplay();
}
}
void Mouse_Motion(int x, int y)
{
rotationX += (double)(x - last_press_x);
rotationY += (double)(y - last_press_y);
last_press_x = x;
last_press_y = y;
glutPostRedisplay();
}

void Mouse_Press(int button, int state, int x, int y)
{
if ( button == GLUT_LEFT_BUTTON && state == GLUT_DOWN )
{
last_press_x = x;
last_press_y = y;
}
}
void Window(int option)
{
switch(option){
case 0:
stop = false;
glutTimerFunc(velocity, Anim, 1); /* Resume */
break;
```

```
case 1:
stop = true;                  /* Pause */
break;
case 2:
velocity = 0;                       /* Advance to the end */
break;
case 3:                       /* Close Application */
exit(0);
break;
}
glutPostRedisplay();
}

void Menu()
{
glutCreateMenu(Window);
glutAddMenuEntry("Resume", 0);
glutAddMenuEntry("Pause", 1);
glutAddMenuEntry("Finish", 2);
glutAddMenuEntry("Exit", 3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
}

void Init(void)
{
        glClearColor(0.0f, 0.0f, 0.1f, 1.0f);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(40.0f, ((GLfloat)WIDTH/(GLfloat)HEIGHT), 1, 50.0f);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_DEPTH_TEST);
        glShadeModel(GL_SMOOTH);
        glColorMaterial ( GL_FRONT_AND_BACK,GL_AMBIENT) ;
        glEnable(GL_COLOR_MATERIAL);

}
int main(int argc, char **argv)
{
int j = -8;
for(int i = 0; i<16; i++){ // Initializing vectors and variables
pos_x[i] = j;
pos_z[i] = -4;
color1[i] = 0;
color2[i] = 1;
color3[i] = 0;
rotY[i] = 0;
j++;
}
glutInit(&argc, argv);
```

```
glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize (WIDTH, HEIGHT);
glutInitWindowPosition (100, 100);
glutCreateWindow("MergeSort");
glutDisplayFunc(Draw);
glutMouseFunc(Mouse_Press);
glutMotionFunc(Mouse_Motion);
glutKeyboardFunc(key);
Init();
Menu();
glutTimerFunc(1, Anim, 1000);
glutMainLoop();
}
```

CHAPTER 7

SNAPSHOTS

Snapshots are nothing but informal photograph taken quickly, typically with a small handheld camera. In other words defined as an informal photograph that is taken quickly a quick view or a small amount of information that tells you a little about what someone or something is like the complete working of the Happy Childhave been picturized using openGL program. And the outcomes are depicted using snapshots. These snapshots sequentially describe all the phases included in the code. This chapter involves snapshots of the project which provides an idea of project. These snapshots depict different situations of the output.

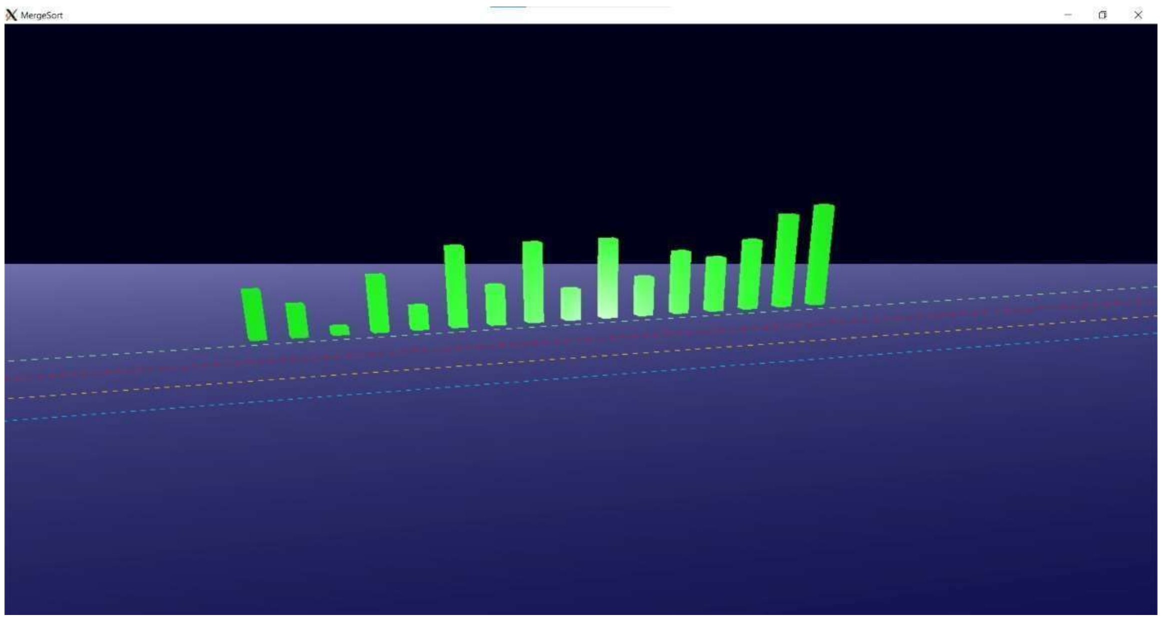

Figure 7.1: Before Merge Sort

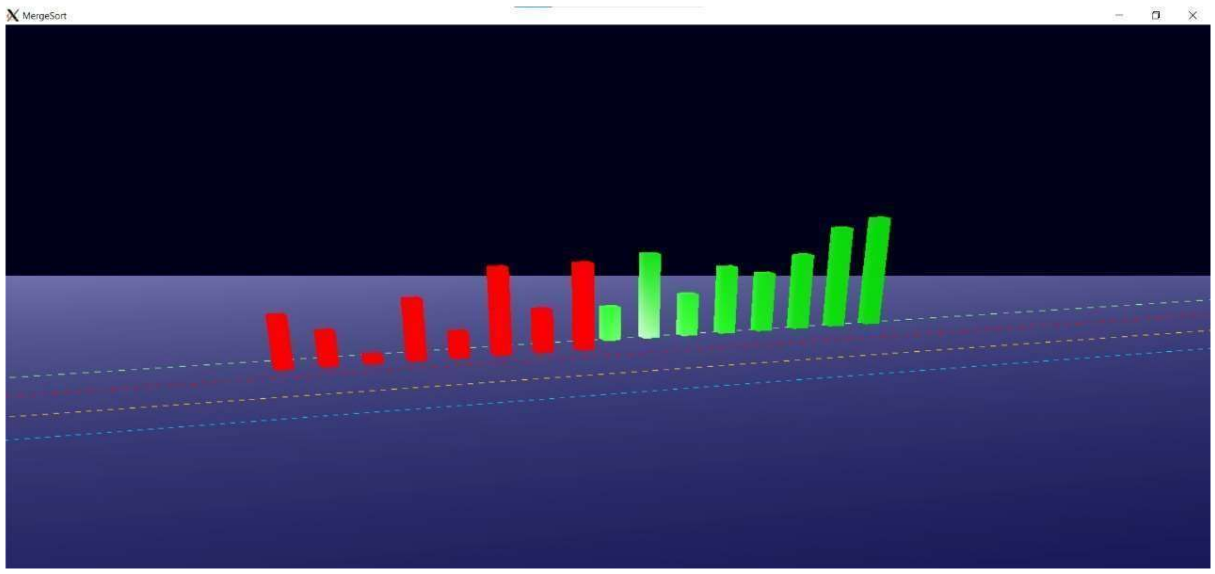

Figure 7.2: Sorting 1

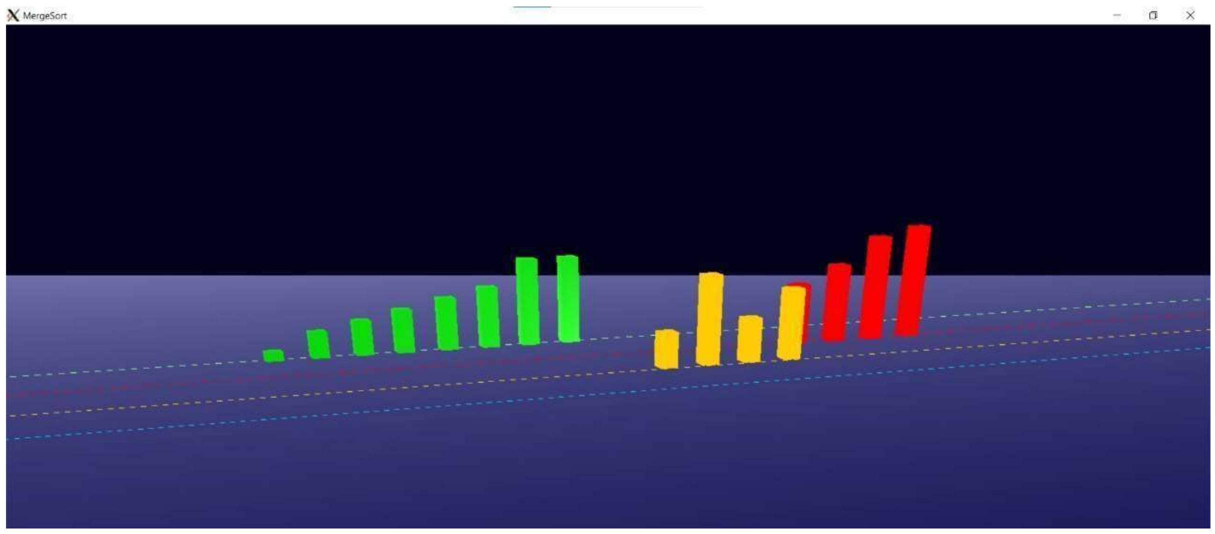

Figure 7.3: Sorting 2

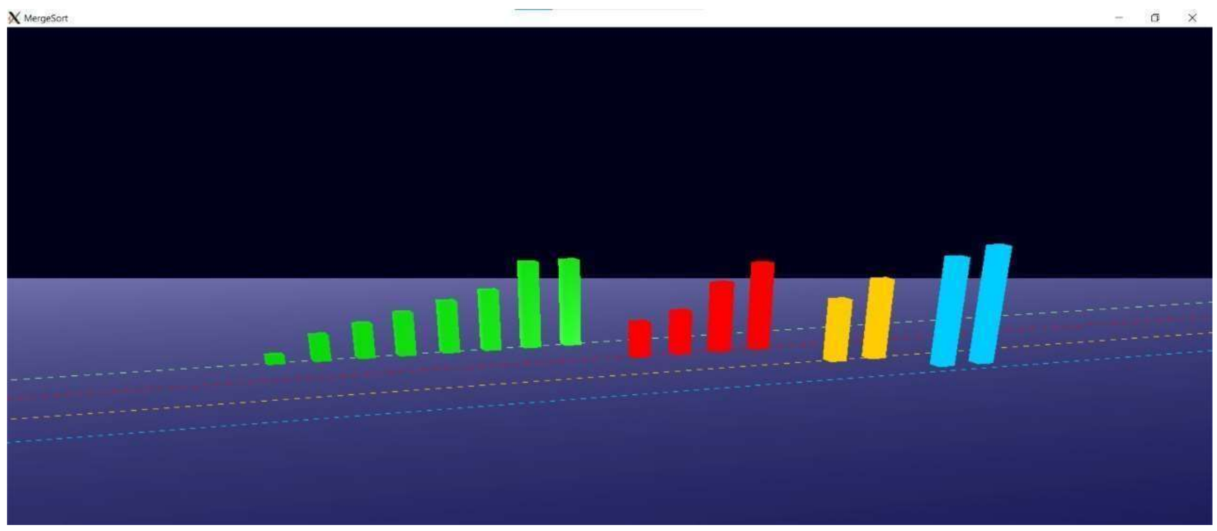

Figure 7.4: Sorting 3

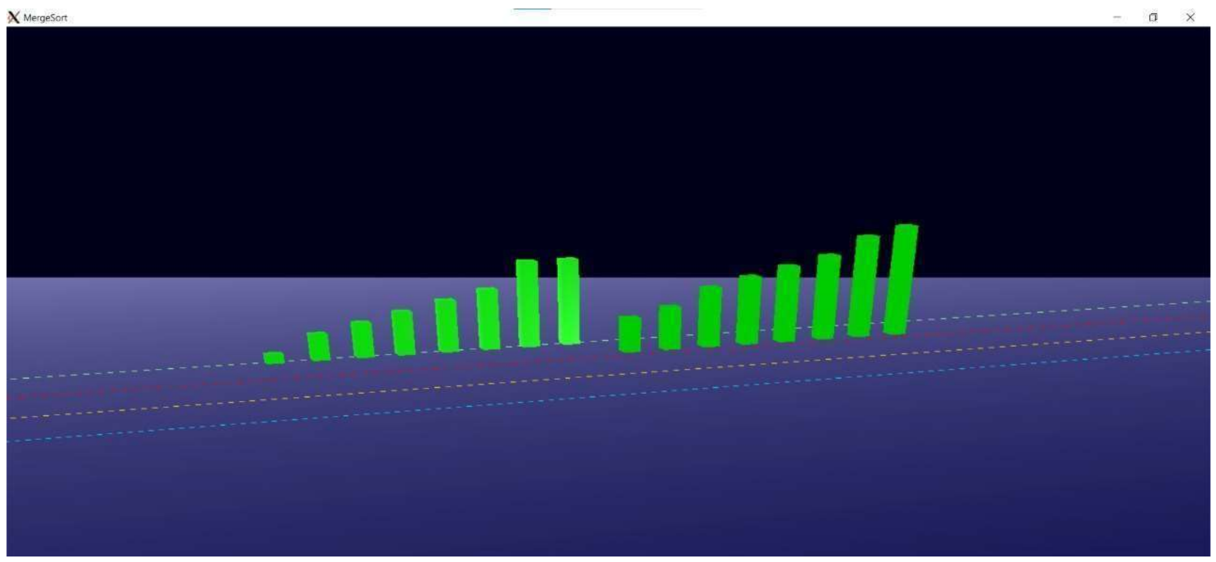

Figure 7.5: Sorting 4

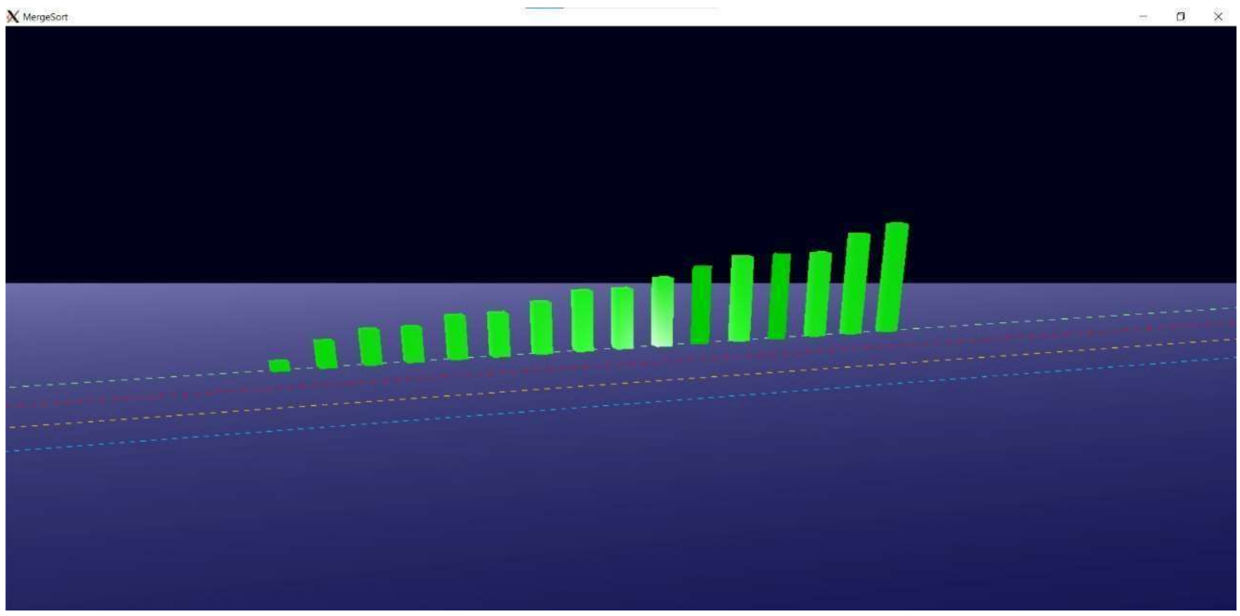

Figure 7.6s: After Merge Sort

CHAPTER 8

CONCLUSION

We have tried level best to build the project efficiency and correctly and have succeeded in building a better project, but may not be a best project. We have implemented the required functions which we had stated earlier. After all testing process, the "MERGE SORT" is now ready. This project is developed using the open GL software and so far proved to be having well Performance. We hope the user may simply just be able to easily understand the project. Over all, the project was a good experience and provides a rich fun to enhance our knowledge of Working on open GL.

Algorithms and programming examples are given in the pseudo code that resembles the programming language, which shares similar syntax and basic constructs with other widely used languages such as C++. The relative simplicity of the c style code presents an easy representation to focus attention on the technical substance of the code. Hence we conclude our project based on Computer Graphics and our report based on our project"MERGE SORT".

REFERENCES

1. Edward Angel, Interactive Computer Graphics A Top –Down Approach with OpenGL, 5th edition, Addison-Wesley, 2008.
2. James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes, Computer Graphics, Addison-Wesley, 1997.
3. Donald Hearn and Pauline Baker: Computer Graphics-OpenGL version, 2nd edition, Pearson Education, 2003 .

Appendix A

Abbreviations

1.	OpenGL	- Open Graphics Library
2.	ARQ	- Automatic Repeat Request
3.	Glut	- OpenGL Utility ToolKit
4.	NACK	- Negative acknowledgement
5.	ACK	- Acknowledgement.