VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi – 590 018

A
Mini Project Report
on

"SIMULATION OF TOWER OF HANOI"

Submitted in partial fulfillment of Computer Graphics Laboratory with Mini project 17CSL68 in Computer Science and Engineering for the Academic Year 2019-2020

Submitted by

SANDEEP VY
1GA17CS134

Under the Guidance of

Mrs. Sushmitha S
Assistant Professor

Mr. Krishna Prasad R
Assistant Professor

GLOBAL ACADEMY OF TECHNOLOGY

Department of Computer Science and Engineering
Rajarajeshwarinagar, Bengaluru - 560 098

2019 – 2020

GLOBAL ACADEMY OF TECHNOLOGY

Department of Computer Science and Engineering

CERTIFICATE

Certified that the V1 Semester Mini Project in Computer Graphics Laboratory with Mini project Entitled **"SIMULATION OF TOWER OF HANOI"** carried out by **Mr.SANDEEP V Y**, bearing **USN 1GA17CS134** is submitted in partial fulfillment for the award of the **Bachelor of Engineering** in Computer Science and Engineering from **Visvesvaraya Technological University, Belagavi** during the year 2019-2020.The Computer Graphics with Mini project report has been approved as it satisfies the academic requirements in respect of the mini project work prescribed for the said degree.

Ms. Sushmitha S Assistant Professor, Dept of CSE, GAT, Bengaluru.	**Mr. Krishna Prasad R** Assistant Professor, Dept of CSE, GAT, Bengaluru.	**Dr. Guru Prasad N** Professor & Head, Dept of CSE, GAT, Bengaluru.

Name of the Examiners Signature with date

1. _____ _____

2. _____ _____

Acknowledgement

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant encouragement and guidance crowned our efforts with success.

I consider myself proud, to be part of **Global Academy of Technology** family, the institution which stood by our way in endeavors.

I express my deep and sincere thanks to our principal **Dr. N. Rana Pratap Reddy** for his support.

I would be grateful to **Dr. Guru Prasad N**, Professor and HOD, Dept Of CSE who is source of inspiration and of invaluable help in channelizing my efforts in right direction.

I wish to thank my internal guides **Ms. Sushmitha S**, Assistant Professor, Dept of CSE and **Mr. Krishna Prasad R**, Assistant Professor, Dept of CSE for guiding and correcting various documents of mine with attention and care. They have taken lot of pain to go through the document and make necessary corrections as and when needed.

I would like to thank the faculty members and supporting staff of the Department of CSE, GAT for providing all the support for completing the Project work.

Finally, I am grateful to my parents and friends for their unconditional support and help during the course of my Project work.

SANDEEP VY

1GA17CS134

ABSTRACT

This is a mini project on Simulation Of Tower Of Hanoi based on OPENGL API for Computer graphics and visualization laboratory. The project simulates the optimal solution of the Tower of Hanoi problem given a number of disk. The project implements various geometric transformations like Translation, Rotation, and Scaling. The basic model consists of three poles, source, auxiliary and destination. The disks initially resting on the source pole reaches the destination. The poles are drawn using the GLUT library function glutSolidCone(), and the disks are drawn using glutSolidTorus().Initially the user is presented with an introduction scene which has a menu to choose the number of disks. On pressing the Enter key the actual simulation scene is loaded. The user can use the mouse wheel to advance through the simulation. An optional animation has been implemented, which shows the movement of the disks from pole to pole. The viewing model used is an Orthogonal Projection. The moves to be performed as per the optimal solution are displayed on the top left of the screen is a raster text using the function glutBitmapCharacter(). Lighting has been implemented by the inbuilt OPENGL lighting functions. Menus have been provided to modify various features such as Lighting, Movecamera, animation, change background color, to automatically simulate the complete solution, restart the simulation and to exit the program. The movement of the camera is only along the Y axis, and is implemented using the gluLookAt() function.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO COMPUTER GRAPHICS

Computer Graphics is concerned with all aspects of producing pictures or images using a computer. Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television.

Applications of Computer Graphics

1. Display of information
2. Design
3. Simulation and animation
4. User interfaces

The Graphics Architecture

Graphics Architecture can be made up of seven components:

1. Display processors
2. Pipeline architectures
3. The graphics pipeline
4. Vertex processing
5. Clipping and primitive assembly
6. Rasterization
7. Fragment processing

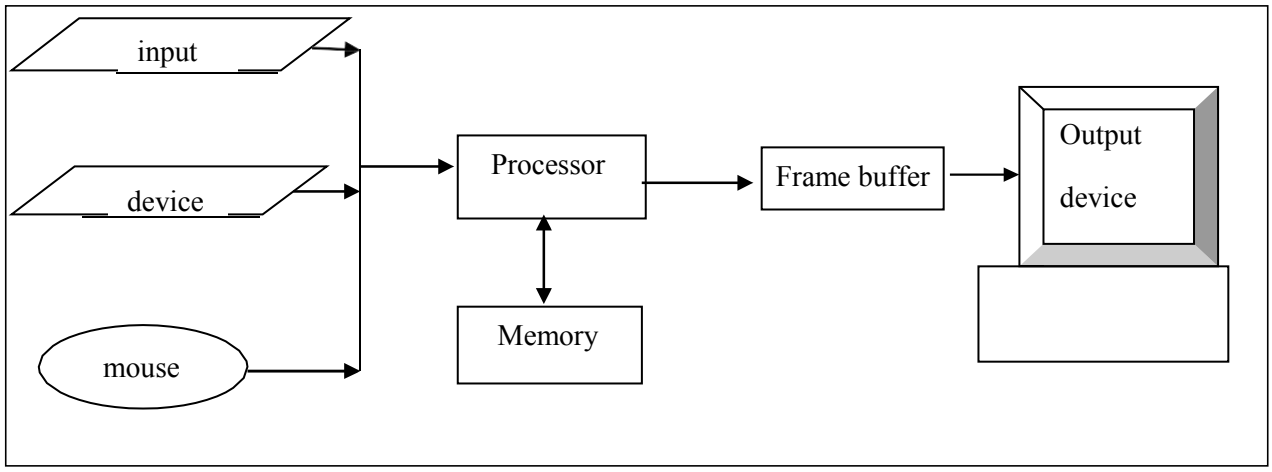

Figure 1.1: Components of Graphics Architecture and their working

1.2 INTRODUCTION TO OPENGL

OpenGL is software used to implement computer graphics. The structure of OpenGL is similar to that of most modern APIs including Java 3D and DirectX. OpenGL is easy to learn, compared with other.

APIs are nevertheless powerful. It supports the simple 2D and 3D programs. It also supports the advanced rendering techniques. OpenGL API explains following 3 components

1. Graphics functions
2. Graphics pipeline and state machines
3. The OpenGL interfaces

There are so many polygon types in OpenGL like triangles, quadrilaterals, strips and fans. There are 2 control functions, which will explain OpenGL through,

1. Interaction with window system
2. Aspect ratio and view ports

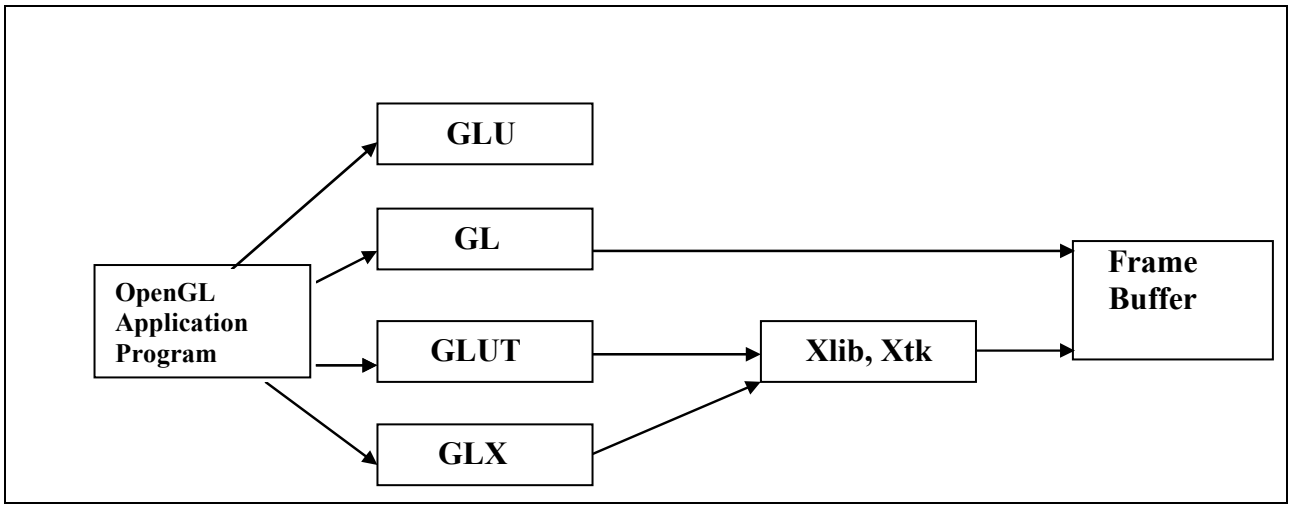

Figure 1.2: OpenGL Library organization

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. This ordering, as shown in Figure 1.2, is not a strict rule of how OpenGL is implemented but provides a reliable guide for predicting what OpenGL will do. The following diagram shows the assembly line approach, which OpenGL takes to process data. Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo the same final steps before the final pixel data is written into the frame buffer.

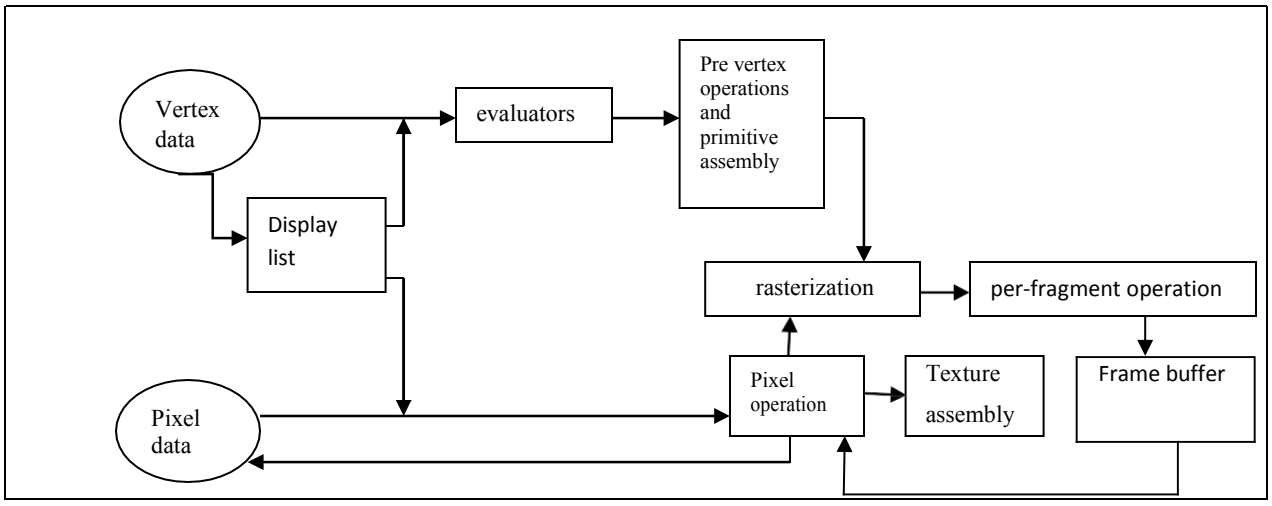

Figure 1.3: OpenGL Order of Operations

CHAPTER 2

REQUIREMENTS SPECIFICATION

2.1 SOFTWARE REQUIREMENTS

- Operating system – Windows 10/8/7
- Code::Blocks 17.12
- OPENGL library files – GL, GLU, GLUT
- Language used is C/C++

2.2 HARDWARE REQUIREMENTS

- Processor – Intel i5 7th Gen
- Memory – 8GB RAM
- 1TB Hard Disk Drive
- Mouse or other pointing device
- Keyboard
- Display device

CHAPTER 3

SYSTEM DEFINITION

3.1 PROJECT DESCRIPTION

The **Tower of Hanoi** is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is $2^n - 1$,where n is the number of disks.

Recursive Solution:

A key to solving this puzzle is to recognize that it can be solved by breaking the problem down into a collection of smaller problems and further breaking those problems down into even smaller problems until a solution is reached. For example:

- label the pegs A, B, C
- let *n* be the total number of discs
- number the discs from 1 (smallest, topmost) to *n* (largest, bottommost)

To move *n* discs from peg A to peg C:

1. move *n*−1 discs from A to B. This leaves disc *n* alone on peg A
2. move disc *n* from A to C
3. move *n*−1 discs from B to C so they sit on disc *n*

The above is a recursive algorithm, to carry out steps 1 and 3, apply the same algorithm again for *n*−1. The entire procedure is a finite number of steps, since at some point the algorithm will be required for *n* = 1. This step, moving a single disc from peg A to peg C, is trivial. This approach can be given a rigorous mathematical formalism with the theory of dynamic programming and is often used as an example of recursion when teaching programming.

CHAPTER 4

IMPLEMENTATION

4.1 SOURCE CODE

```
#include<GL/glut.h>
#pragma GCC diagnostic ignored "-Wwrite-strings"
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#define LIGHT_ON 0
#define LIGHT_OFF 1
int pos[16] = {10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85};
int peg[3] = {50,150,250};
int moves[10000][3];
int max_moves;
int POLES[3][10];
int top[3]={-1,-1,-1};
int NUM_DISKS=3;
int cnt,counter,speed=5;
int line1=90,line2=85;
float ycoordinate;
int lightflag=1,animationFlag=1,randomColorFlag=0;

void push(int p,int disk)
{
    POLES[p][++top[p]] = disk;
}

void pop(int p)
```

```
{
    top[p]--;
}

void tower(int n,int src,int temp,int dst)
{
    if(n>0)
    {
            tower(n-1,src,dst,temp);
            moves[cnt][0] = n;
            moves[cnt][1] = src;
            moves[cnt][2] = dst;
            cnt++;
            tower(n-1,temp,src,dst);
    }
}

void drawPegs()
{
    int i;
    glColor3f(0.5,0.0,0.1);
    for(i=0;i<3;i++)
    {
            glPushMatrix();
            glTranslatef(peg[i],5,0);
            glRotatef(-90,1,0,0);
            glutSolidCone(2,70,20,20);
            glutSolidTorus(2,45, 20, 20);
            glPopMatrix();
    }

}
```

```
void printString(char *text)
{
    int len=strlen(text),i;
    for(i=0;i<len;i++)
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,text[i]);
}

void drawText()
{
    glColor3f(1,1,1);
    glRasterPos3f(-70,line1,0);
    printString("Move :");
    char str[5];
    sprintf(str, "%d", counter);
    glRasterPos3f(-40,line1,0);
    printString(str);
    glRasterPos3f(-70,line2,0);
    printString("Disk");
    char str1[10];
    sprintf(str1, "%d", moves[counter][0]);
    glRasterPos3f(-50,line2,0);
    printString(str1);
    glRasterPos3f(-40,line2,0);
    printString("from");
    char src[2];
    if(moves[counter][1]==0)strcpy(src,"A");
    else if(moves[counter][1]==1)strcpy(src,"B");
    else strcpy(src,"C");
    glRasterPos3f(-20,line2,0);
    printString(src);
    glRasterPos3f(-10,line2,0);
```

```
    printString("to");
    char dst[2];
    if(moves[counter][2]==0)strcpy(dst,"A");
    else if(moves[counter][2]==1)strcpy(dst,"B");
    else strcpy(dst,"C");
    glRasterPos3f(0,line2,0);
    printString(dst);
    glColor3f(0.6,0.3,0.5);
    glBegin(GL_POLYGON);
            glVertex3f(-75,93,-5);
            glVertex3f(-75,83,-5);
            glVertex3f(10,83,-5);
            glVertex3f(10,93,-5);
    glEnd();
    glColor3f(1,0,0);
    glRasterPos3f(peg[0],70,0);
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'A');
    glRasterPos3f(peg[1],70,0);
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'B');
    glRasterPos3f(peg[2],70,0);
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'C');
}

void drawSolved()
{
    glColor3f(1,1,0);
    glRasterPos3f(-60,87,0);
    printString("Solved !!");
    glColor3f(0.6,0.3,0.5);
    glBegin(GL_POLYGON);
            glVertex3f(-75,93,-5);
            glVertex3f(-75,83,-5);
```

```
            glVertex3f(10,83,-5);
            glVertex3f(10,93,-5);
    glEnd();
    glColor3f(1,0,0);
    glRasterPos3f(peg[0],70,0);
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'A');
    glRasterPos3f(peg[1],70,0);
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'B');
    glRasterPos3f(peg[2],70,0);
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'C');
}

void display()
{
    int i,j,k;
    if(randomColorFlag)
            glClearColor((rand()%100)/100.0,(rand()%100)/100.0,(rand()%100)/100.0,0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    if(lightflag)glEnable(GL_LIGHTING);
    glPushMatrix();
    gluLookAt(0,ycoordinate,0,0,0,-1,0,1,0);
    drawPegs();
    for(i=0;i<3;i++)
    {
            k=0;
            for(j=0;j<=top[i];j++)
            {
                    glPushMatrix();
                    glTranslatef(peg[i],pos[k++],0);
                    glRotatef(90,1,0,0);
                    glColor3f(0.1*POLES[i][j],0.2*POLES[i][j],0);
                    glutSolidTorus(2.0, 4*POLES[i][j], 20, 20);
```

```
                glPopMatrix();
            }
        }
        glPopMatrix();
        glDisable(GL_LIGHTING);
        if(counter==max_moves)
            drawSolved();
        else
            drawText();
        if(lightflag)glEnable(GL_LIGHTING);
        glutSwapBuffers();
    }


    void lighting()
    {
        GLfloat shininess[] = {50};
        GLfloat white[] = {0.6,0.6,0.6,1};
        glEnable(GL_COLOR_MATERIAL);
        glColorMaterial(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE);
        GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
        GLfloat light_position[] = {100,60, 10, 0.0 };
        glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, white);
        glMaterialfv(GL_FRONT, GL_SPECULAR, white);
        glMaterialfv(GL_FRONT, GL_SHININESS, shininess);
        glEnable(GL_LIGHT0);
    }

    void init()
    {
```

```
    glClearColor(0.0,0.0,0.0,0);
    glColor3f(1,0,0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-80,350,-10,100,-100,100);
    glMatrixMode(GL_MODELVIEW);
    glEnable(GL_DEPTH_TEST);
    lighting();
}

void animate(int n,int src,int dest)
{
    int i;
    if(speed<=0)speed=1;
    for(i=pos[top[src]+1];i<90;i+=speed)
    {
            glPushMatrix();
            glTranslatef(peg[src],i,0);
            glRotatef(85,1,0,0);
            glColor3f(0.1*n,0.2*n,0);
            glutSolidTorus(2.0, 4*n, 20, 20);
            glPopMatrix();
            glutSwapBuffers();
            display();
    }
    if(peg[src]<peg[dest])
            for(i=peg[src];i<=peg[dest];i+=speed)
            {
                    glPushMatrix();
                    glTranslatef(i,90,0);
                    glRotatef(85,1,0,0);
                    glColor3f(0.1*n,0.2*n,0);
```

```
                glutSolidTorus(2.0, 4*n, 20, 20);
                glPopMatrix();
                glutSwapBuffers();
                display();
        }
    else
        for(i=peg[src];i>=peg[dest];i-=speed)
        {
                glPushMatrix();
                glTranslatef(i,90,0);
                glRotatef(85,1,0,0);
                glColor3f(0.1*n,0.2*n,0);
                glutSolidTorus(2.0, 4*n, 20, 20);
                glPopMatrix();
                glutSwapBuffers();
                display();
        }
    for(i=70;i>pos[top[dest]+1];i-=speed)
    {
        glPushMatrix();
        glTranslatef(peg[dest],i,0);
        glRotatef(85,1,0,0);
        glColor3f(0.1*n,0.2*n,0);
        glutSolidTorus(2.0, 4*n, 20, 20);
        glPopMatrix();
        glutSwapBuffers();
        display();
    }
}

void mouse(int btn,int mode,int x,int y)
{
```

```
    if(btn == 4 && mode == GLUT_DOWN)
    {
        if(counter<max_moves)
        {
            pop(moves[counter][1]);
            if(animationFlag)
                animate(moves[counter][0],moves[counter][1],moves[counter][2]);
            push(moves[counter][2],moves[counter][0]);
            counter++;
        }
    }
    if(btn == 3 && mode == GLUT_DOWN)
    {
        if(counter>0)
        {
            counter--;
            pop(moves[counter][2]);
            if(animationFlag)
                animate(moves[counter][0],moves[counter][2],moves[counter][1]);
            push(moves[counter][1],moves[counter][0]);
        }
    }
    glutPostRedisplay();
}

void restart()
{
    int i;
    memset(POLES,0,sizeof(POLES));
    memset(moves,0,sizeof(POLES));
    memset(top,-1,sizeof(top));
    cnt=0,counter=0;
```

```
    ycoordinate=0.1;
    max_moves = pow(2,NUM_DISKS)-1;
    for(i=NUM_DISKS;i>0;i--)
    {
            push(0,i);
    }
    tower(NUM_DISKS,0,1,2);
}

void processMenuLighting(int option)
{
    switch(option)
    {
            case LIGHT_OFF:
                    glDisable(GL_LIGHTING);
                    lightflag=0;
            break;
            case LIGHT_ON:
                    glEnable(GL_LIGHTING);
                    lightflag=1;
            break;
    }
    glutPostRedisplay();
}
void processMenuMain2(int option)
{

}
void processMenuCamera(int option)
{
    switch(option)
    {
```

```
            case 0:ycoordinate+=0.1;break;
            case 1:ycoordinate-=0.1;break;
    }
    glutPostRedisplay();
}

void processMenuRestart(int option)
{
    if(option==0)
    {
            restart();
            glutPostRedisplay();
    }
}

void processMenuExit(int option)
{
    if(option==0)exit(0);
}

void processMenuAnimate(int option)
{
    switch(option)
    {
            case 0:
                    animationFlag=1;
            break;
            case 1:
                    animationFlag=0;
    }
}
```

```
void processMenuSolveCompletely(int option)
{
    int temp=animationFlag;
    animationFlag=0;
    int i,j;
    while(counter<max_moves)
    {
            mouse(4,GLUT_DOWN,0,0);
            display();
            for(i=0;i<100000;i++)
                    for(j=0;j<100;j++);
    }
    animationFlag=temp;
}

void processMenuBgColor(int option)
{
    switch(option)
    {
            case 0:glClearColor(0,0,0,0);randomColorFlag=0;break;
            case 1:glClearColor(1,1,1,0);randomColorFlag=0;break;
            case 2:glClearColor(1,0,0,0);randomColorFlag=0;break;
            case 3:glClearColor(0,1,0,0);randomColorFlag=0;break;
            case 4:glClearColor(0,0,1,0);randomColorFlag=0;break;
            case 5:randomColorFlag=1;break;
    }
    glutPostRedisplay();
}

void createGLUTMenus2()
{
    int menu = glutCreateMenu(processMenuLighting);
```

```
glutAddMenuEntry("On",LIGHT_ON);
glutAddMenuEntry("Off",LIGHT_OFF);
int menuExit = glutCreateMenu(processMenuExit);
glutAddMenuEntry("Yes",0);
glutAddMenuEntry("No",1);
int menuCamera = glutCreateMenu(processMenuCamera);
glutAddMenuEntry("+0.1",0);
glutAddMenuEntry("-0.1",1);
int menuRestart = glutCreateMenu(processMenuRestart);
glutAddMenuEntry("Yes",0);
glutAddMenuEntry("No",1);
int menuAnimate = glutCreateMenu(processMenuAnimate);
glutAddMenuEntry("On",0);
glutAddMenuEntry("Off",1);
int menuBgColor = glutCreateMenu(processMenuBgColor);
glutAddMenuEntry("Black",0);
glutAddMenuEntry("White",1);
glutAddMenuEntry("Red",2);
glutAddMenuEntry("Green",3);
glutAddMenuEntry("Blue",4);
glutAddMenuEntry("Random",5);
int menuSolveCompletely = glutCreateMenu(processMenuSolveCompletely);
glutAddMenuEntry("Start",0);
glutCreateMenu(processMenuMain2);
glutAddSubMenu("Lighting",menu);
glutAddSubMenu("Move Camera",menuCamera);
glutAddSubMenu("Animation",menuAnimate);
glutAddSubMenu("Background Color",menuBgColor);
glutAddSubMenu("Solve Completely",menuSolveCompletely);
glutAddSubMenu("Restart",menuRestart);
glutAddSubMenu("Exit",menuExit);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

```
}

void processMenuMain1(int option)
{
}

void processMenuNumDisks(int option)
{
    NUM_DISKS=option;
    restart();
    glutPostRedisplay();
}

void createGLUTMenus1()
{
    int menu = glutCreateMenu(processMenuNumDisks);
    glutAddMenuEntry("3",3);
    glutAddMenuEntry("4",4);
    glutAddMenuEntry("5",5);
    glutAddMenuEntry("6",6);
    glutAddMenuEntry("7",7);
    glutAddMenuEntry("8",8);
    glutAddMenuEntry("9",9);
    glutAddMenuEntry("10",10);
    int menuExit = glutCreateMenu(processMenuExit);
    glutAddMenuEntry("Yes",0);
    glutAddMenuEntry("No",1);
    glutCreateMenu(processMenuMain1);
    glutAddSubMenu("Number of Disks",menu);
    glutAddSubMenu("Exit",menuExit);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

```
void strokeString(float x,float y,float sx,float sy,char *string,int width)
{
    char *c;
    glLineWidth(width);
    glPushMatrix();
    glTranslatef(x,y,0);
    glScalef(sx,sy,0);
    for(c=string; *c != '\0'; c++)
    {
            glutStrokeCharacter(GLUT_STROKE_ROMAN, *c);
    }
    glPopMatrix();
}

void initfirst()
{
    glClearColor(0.7,1,0,1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0,1000,0,1000,-1,1);
    glMatrixMode(GL_MODELVIEW);
}

void first()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1,0,0);
    strokeString(50,850,0.15,0.15,"GLOBAL ACADEMY OF TECHNOLOGY",2);
    strokeString(100,750,0.3,0.3,"DEPARTMENT OF COMPUTER SCIENCE",4);
    strokeString(300,670,0.3,0.3,"AND ENGINEERING",4);
    strokeString(200,500,0.3,0.3,"A CG MINI PROJECT ON",2);
```

```
    glColor3f(0,0,0);
    strokeString(250,420,0.4,0.4,"TOWER OF HANOI",6);
    strokeString(150,220,0.2,0.2,"NUMBER OF DISKS:",3);
    glColor3f(0,1,0);
    char str[5];
    sprintf(str, "%d", NUM_DISKS);
    strokeString(450,220,0.2,0.2,str,3);
    glColor3f(1,0,0);
    strokeString(50,100,0.17,0.17,"1 . Set the number of disks using the menu",2);
    strokeString(50,50,0.17,0.17,"2 . Press (Enter) to start the simulation",2);
    strokeString(650,200,0.15,0.15,"By:",2);
    glColor3f(0,0,0);
    strokeString(650,160,0.18,0.18,"SANDEEP V Y",2);
    strokeString(600,30,0.2,0.2,"Ms.SUSHMITHA S(Ass.Professor).",2);
    glColor3f(1,0,0);
    strokeString(650,120,0.18,0.18,"1GA17CS134",2);
    strokeString(650,60,0.15,0.15,"Under the guidance of.",2);
    glutSwapBuffers();
}

void keyboard2(unsigned char c, int x, int y){}
void keyboard(unsigned char c, int x, int y)
{
    switch(c)
    {
            case 13:
                    restart();
                    init();
                    glutDisplayFunc(display);
                    createGLUTMenus2();
                    glutKeyboardFunc(keyboard2);
                    glutMouseFunc(mouse);
```

```
            break;
        }
        glutPostRedisplay();
    }

    int main(int argc,char** argv)
    {
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
        glutInitWindowSize(1024,720);
        glutInitWindowPosition(100,100);
        glutCreateWindow("tower of hannoi");
        initfirst();
        glutDisplayFunc(first);
        createGLUTMenus1();
        glutKeyboardFunc(keyboard);
        glutMainLoop();
        return 0;
}
```

CHAPTER 5

TESTING AND RESULTS

5.1 DIFFERENT TYPES OF TESTING

1. Unit Testing

Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components.

2. Module Testing

A module is a collection of dependent components such as a object class, an abstract Data type or some looser collection of procedures and functions. A module related Components, so can be tested without other system modules.

3. System Testing

This is concerned with finding errors that result from unanticipated interaction between Sub-system interface problems.

4. Acceptance Testing

The system is tested with data supplied by the system customer rather than simulated test data.

CHAPTER 6

SNAPSHOTS

Figure 5.1: Introduction Screen

Description: Choosing Number Of Disks using the menu

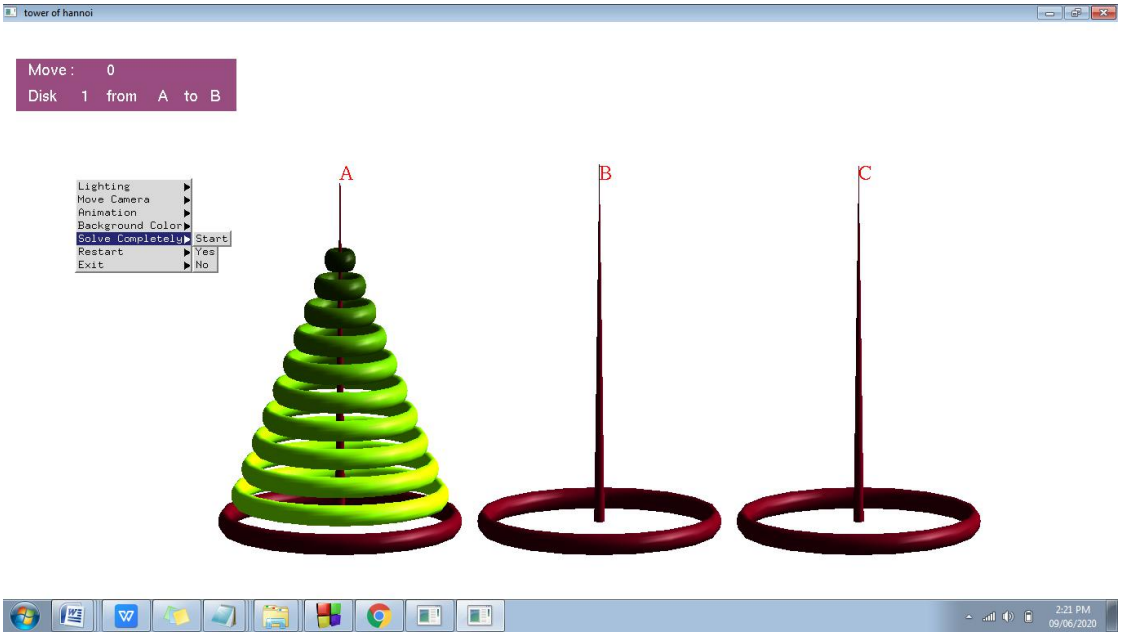

Figure 5.2: Initial View of Tower Of Hanoi

Description: Clicking Start button by using menu

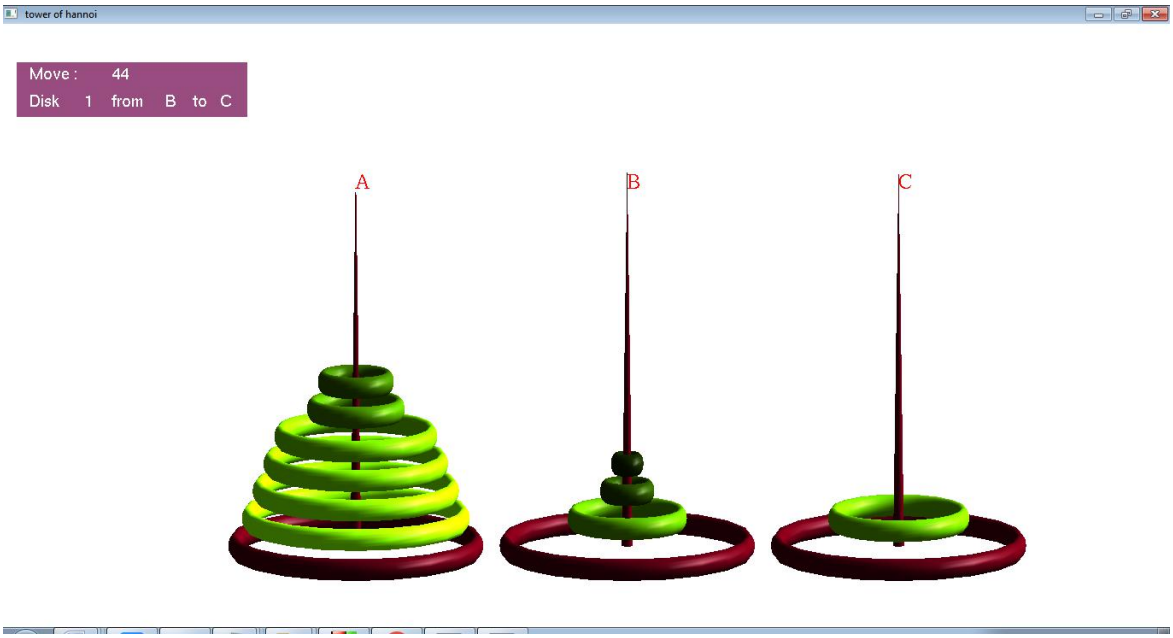

Figure 5.3: Solving the problem

Description: Disks from A to C are getting shifted through auxiliary B

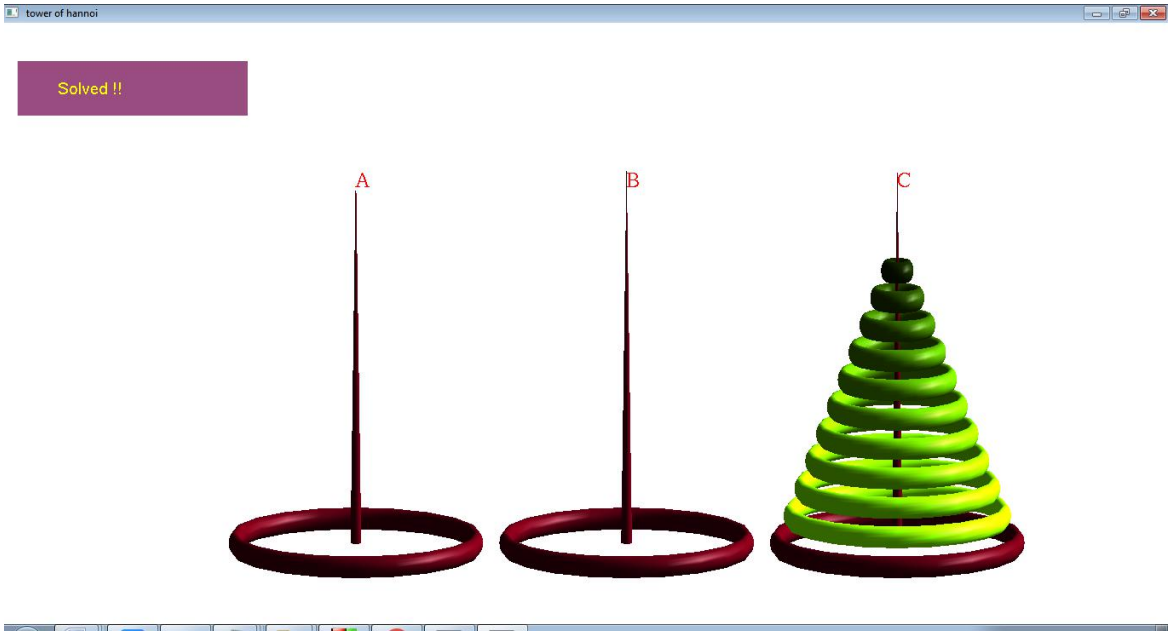

Figure 5.4: Problem got solved

Description: Disks for A to C are successfully shifted

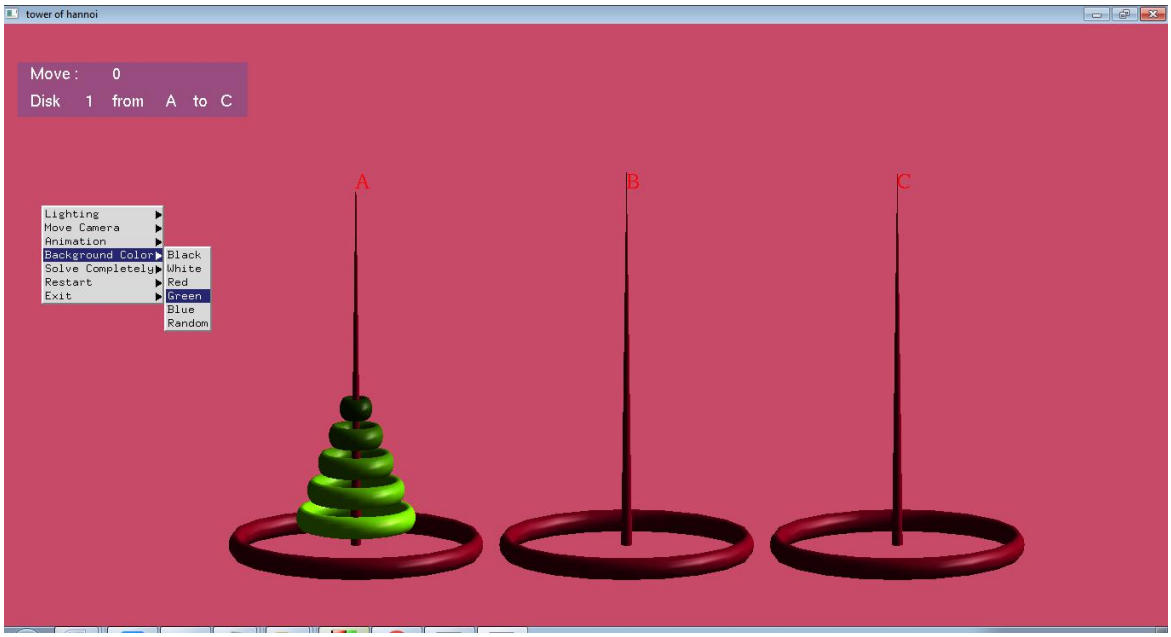

Figure 5.4: Choosing Background Color

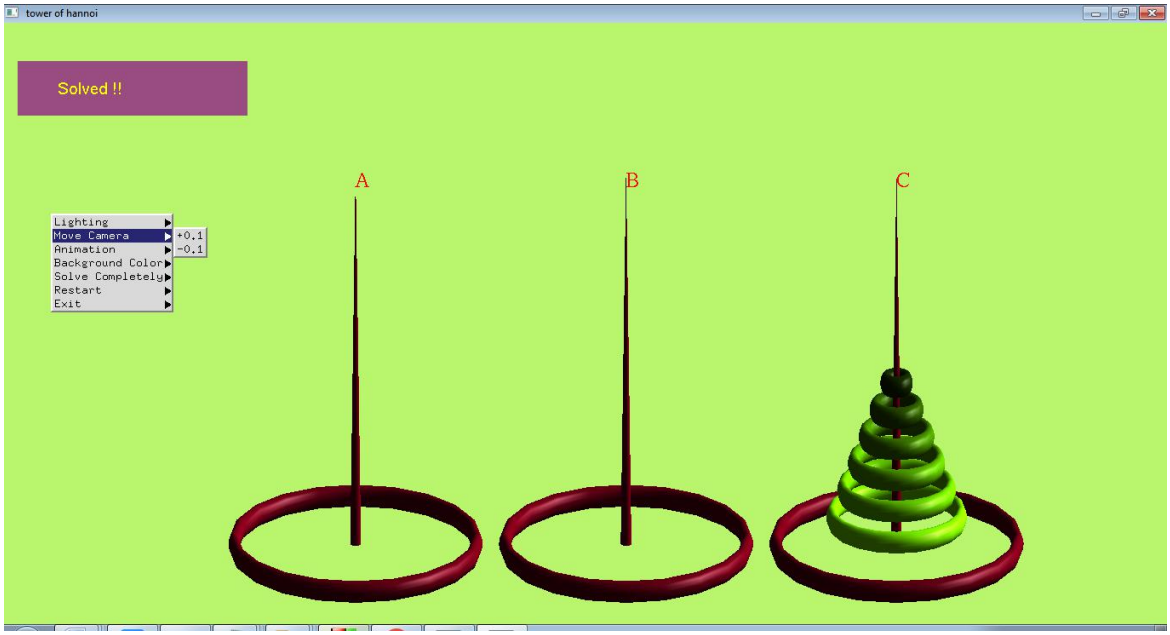

Figure 5.4: Choosing Camera Angle

CONCLUSION

It was a wonderful learning experience for me working on this project. This project took me through various phases of project development and gave me real insight into the world of software engineering. Simulation of tower of hanoi is designed and implemented using graphics software called OpenGL which has become widely accepted standard for developing graphic application. Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics.

The user-friendly interface allows the user to interact with it very effectively. So, I conclude on note that this project has given me a great exposure to the OpenGL and computer graphics. This is very reliable graphics package supporting various primitive objects like polygon, line loops, etc. Also, color selection, menu and mouse-based interface are included. Transformations like translation, rotation, scaling is also provided. The joy of working and the thrill involved while tackling the various problems and challenges gave me a feel of developers industry.

BIBLIOGRAPHY

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

[3]. Official OPENGL Documentation at https://www.**opengl**.org/**documentation**/

[4]. https://open.gl/

[5]. https://en.wikipedia.org/wiki/FreeGLUT

[6]. https://en.wikipedia.org/wiki/Tower_of_Hanoi