VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi – 590 018

A

Mini Project Report

on

"Tower of Hanoi Animation"

Submitted in partial fulfillment of Computer Graphics Laboratory with Mini project 18CSL67 in Computer Science and Engineering for the Academic Year 2020-2021

Submitted by

Madhan M

1GA18CS079

Under the Guidance of

Mrs. Reshma S

Assistant Professor

GLOBAL ACADEMY OF TECHNOLOGY

Department of Computer Science and Engineering

Rajarajeshwarinagar, Bengaluru - 560 098

2020 – 2021

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 **"Tower of Hanoi Animation"** carried out by **Mr. Madhan M** , bearing **USN 1GA18CS079** 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 2020-2021.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.

Mrs. Reshma S Assistant Professor, Dept of CSE, GAT, Bengaluru.	**Dr. Bhagyashri R Hanji** 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 . Bhagyashri R Hanji. ,** 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 guide **Prof. Reshma S**, 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.

NAME : Madhan M

USN : 1GA18CS079

ABSTRACT

The project uses the OpenGL and C/C++ library functions to simulate the steps while solving Towers of Hanoi problem.

The project has been implemented by efficiently using the data structures to obtain the optimized results and various functions and features are available by the OpenGL software package have been utilized effectively. Computer graphics can do many things, including modeling, simulation and visualization of an object or a problem. Modeling is a representation of how people describe or explain an object, system, or a concept, which is usually manifested by simplification or idealization. This can be represented by physical models (mockups, prototypes), the model image (design drawings, computer images), or mathematical formulas.

OpenGL support this modeling capability as OpenGL has additional features to better produce something more realistic. OpenGL allows us to create a graph that can be run on any operating system only minor adjustment. The 3-D graphics package designed here provides an interface for the users for handling the display and manipulation of Tower of Hanoi. The Keyboard and mouse are the main input devices used here.

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

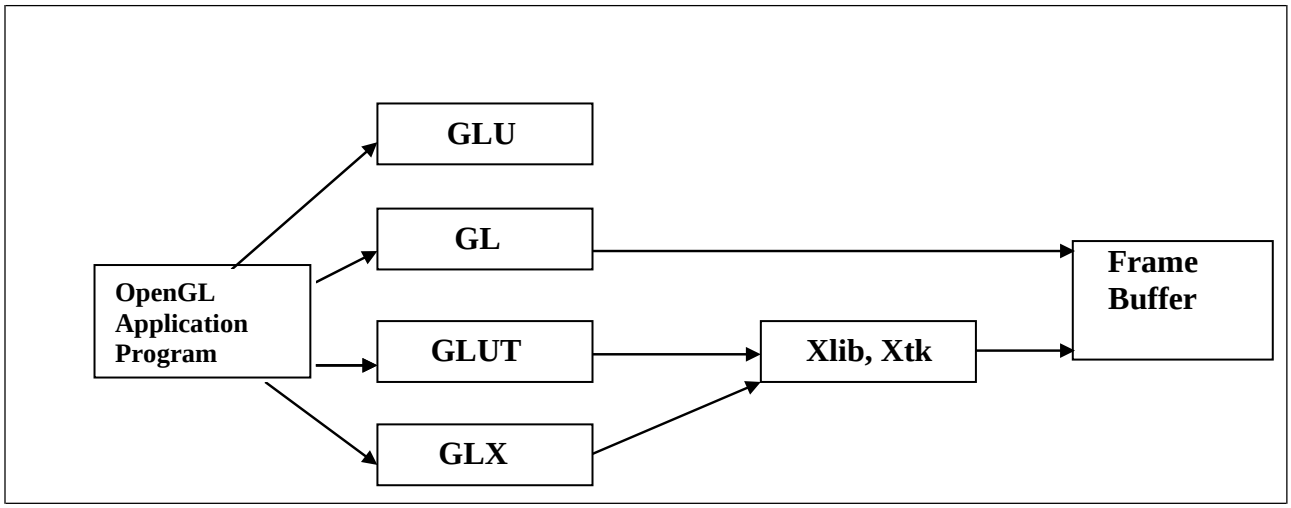

Figure 1.2: OpenGL Library organization

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.

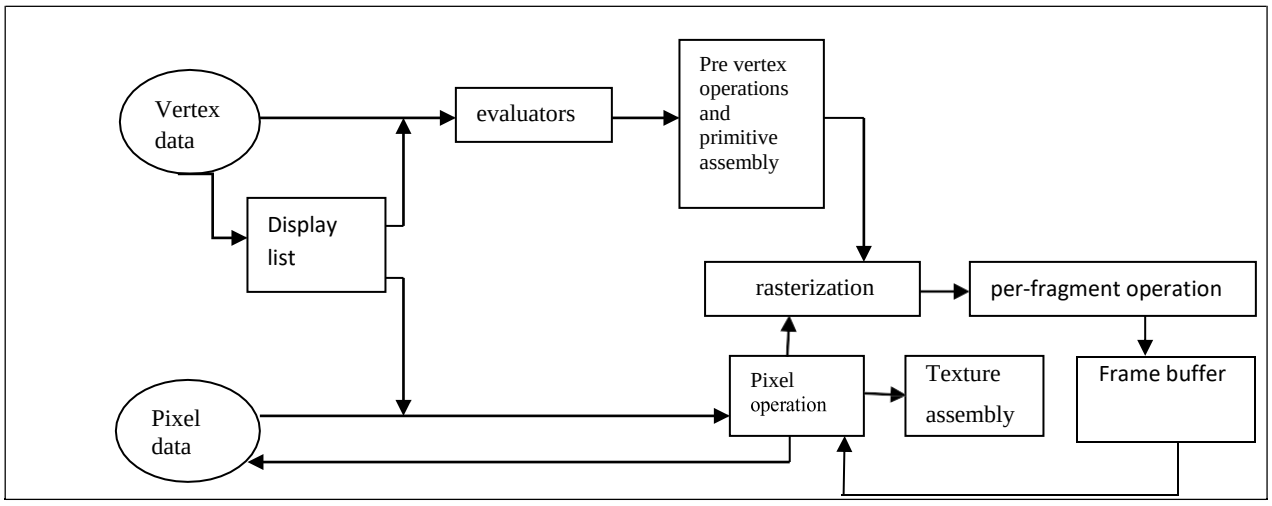

Figure 1.3: OpenGL Order of Operations

CHAPTER 2

REQUIREMENTS SPECIFICATION

2.1 SOFTWARE REQUIREMENTS

- Operating system – Windows 10
- Visual Studio Code Community 2019 16.1
- OPENGL library files – GL, GLU, GLUT
- Language used is C/C++

2.2 HARDWARE REQUIREMENTS

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

CHAPTER 3

SYSTEM DEFINITION

3.1 PROJECT DESCRIPTION

OpenGL is software which provides a graphical interface. It is an interface between the application program and the graphics hardware.

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. 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.

In this project we simulate how the problem is solved.

3.2 USER DEFINED FUNCTIONS

- **void push() and void pop() :** This function is used for push and pop operation.
- **void tower() :** This function is used to generate towers.
- **void drawPegs() and void drawText():** This is used create towers and pegs in position.
- **void drawSolved() :** This function is used to show the text after the problem is solved
- **void display() :** This function is used to create the view.
- **void lighting() :** This function is used to add lighting effects to the tower and pegs.
- **void animate () :** This function is used to animate the movement of pegs.
- **void mouse() :** This function is used to add mouse events.
- **void restart() :** This function is used to set the pegs to initial position and reset the values.
- **void createGLUTMenus1() :** This function is used to add menu events in first screen.
- **void createGLUTMenus2() :** This function is used to add menu events in second screen
- **void strokeString () :** This function is used to add text in a specified position on the screen.
- **void first() :** This function is used to generate the first screen.
- **void keyboard() and void keyboard2 :** This function is used for keyboard events.

CHAPTER 4

IMPLEMENTATION

4.1 SOURCE CODE

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.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 = 5;
int cnt, counter, speed = 5;
int line1 = 90, line2 = 85;
int lightflag = 1;

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(3, 70, 20, 20);
            glutSolidTorus(3, 45, 20, 20);
            glPopMatrix();
    }

}

void printString(const 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_s(str, "%d", counter);
    glRasterPos3f(-40, line1, 0);
    printString(str);
    glRasterPos3f(-70, line2, 0);
    printString("Disk");
    char str1[10];
    sprintf_s(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_s(src, "A");
    else if (moves[counter][1] == 1)strcpy_s(src, "B");
    else strcpy_s(src, "C");
    glRasterPos3f(-20, line2, 0);
    printString(src);
```

```
    glRasterPos3f(-10, line2, 0);
    printString("to");
    char dst[2];
    if (moves[counter][2] == 0)strcpy_s(dst, "A");
    else if (moves[counter][2] == 1)strcpy_s(dst, "B");
    else strcpy_s(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;

    //glClearColor((rand() % 100) / 100.0, (rand() % 100) / 100.0, (rand() % 100) / 100.0, 0);
    glClearColor(0.5, 0.5, 0.1, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_LIGHTING);
    glPushMatrix();
    gluLookAt(0, 0.1, 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[] = { 500 };
    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[] = { 100,160, 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 == GLUT_KEY_LEFT && mode == GLUT_DOWN)
    {
        if (counter < max_moves)
        {
            pop(moves[counter][1]);
            animate(moves[counter][0], moves[counter][1], moves[counter][2]);
            push(moves[counter][2], moves[counter][0]);
            counter++;
        }
    }
    if (btn == GLUT_KEY_RIGHT && mode == GLUT_DOWN)
    {
        if (counter > 0)
        {
            counter--;
            pop(moves[counter][2]);
            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;
    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 processMenuRestart(int option)
{
    if (option == 0)
    {
            restart();
            glutPostRedisplay();
    }
}

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

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

void createGLUTMenus2()
{
    int menuExit = glutCreateMenu(processMenuExit);
    glutAddMenuEntry("Yes", 0);
    glutAddMenuEntry("No", 1);
    int menuRestart = glutCreateMenu(processMenuRestart);
    glutAddMenuEntry("Yes", 0);
    glutAddMenuEntry("No", 1);
    int menuSolveCompletely = glutCreateMenu(processMenuSolveCompletely);
    glutAddMenuEntry("Start", 0);
    glutCreateMenu(processMenuMain2);
    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,const char* string, int width)
{
    const 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, 0, 0, 0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, 1000, 0, 1000, -1, 1);
    glMatrixMode(GL_MODELVIEW);
}

void first()
{
    glClearColor(0.5, 0.5, 0.1, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1, 1, 1);
    strokeString(15, 900, 0.4, 0.4, "Tower of hanoi Animation for", 3);
    glColor3f(1, 1, 0);
    char str[5];
    sprintf_s(str, "%d", NUM_DISKS);
    strokeString(820, 900, 0.4, 0.4, str , 3);
    glColor3f(1, 1, 1);
```

```
    strokeString(870, 900, 0.4, 0.4, "Disks", 3);
    glColor3f(1, 1, 0);
    strokeString(100, 750, 0.4, 0.4, "Click Enter to start animation!", 3);
    glColor3f(1, 0, 0);
    strokeString(800, 700, 0.22, 0.22, "Madhan M", 2);
    glColor3f(1, 0, 0);
    strokeString(800, 660, 0.22, 0.22, "1GA18CS079", 2);
    glutSwapBuffers();
}

void keyboard2(unsigned char c, int x, int y)
{
    if (c == 'a')
            mouse(GLUT_KEY_LEFT, GLUT_DOWN, 0, 0);
    else if (c == 's')
            mouse(GLUT_KEY_RIGHT, GLUT_DOWN, 0, 0);
}
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 Hanoi");
    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.

5.2 TEST CASES

The test cases provided here test the most important features of the project.

Table 5.2.1: Test Case

Sl No	Test Input	Expected Results	Observed Results	Remarks
1	Right click	Menu to be displayed	Menu displayed	Pass
2	Left click	Menu not to be displayed	Menu not displayed	Pass
3	Click on menu item “Number of Disks”	Shows all the available number of disks options	Options are displayed	Pass
4	Click key ‘a’ and ‘b’	Shows animation of pegs moving to next tower	Animation is visible	Pass
5	Click on menu item “Solve Completely”	The Tower of Hanoi is solved with animation	Animation is visible	Pass
6	Click on menu item “Restart”	The rings must be back to the First tower	The rings are back to the First tower	Pass
8	Click on menu item “Exit”	Exit	Exit	Pass

CHAPTER 6

SNAPSHOTS

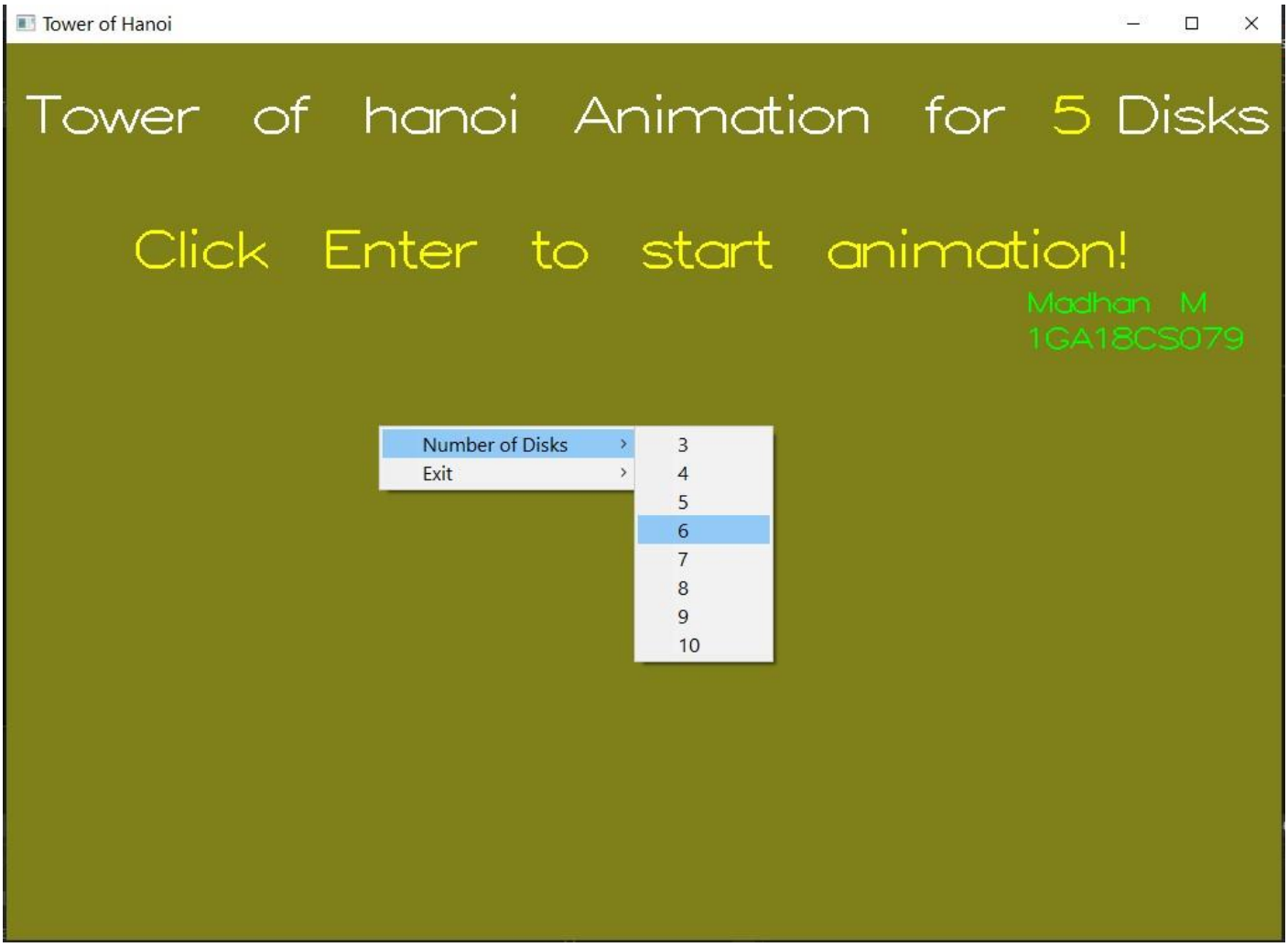

Figure 5.1: Main Screen with menu options on mouse right click

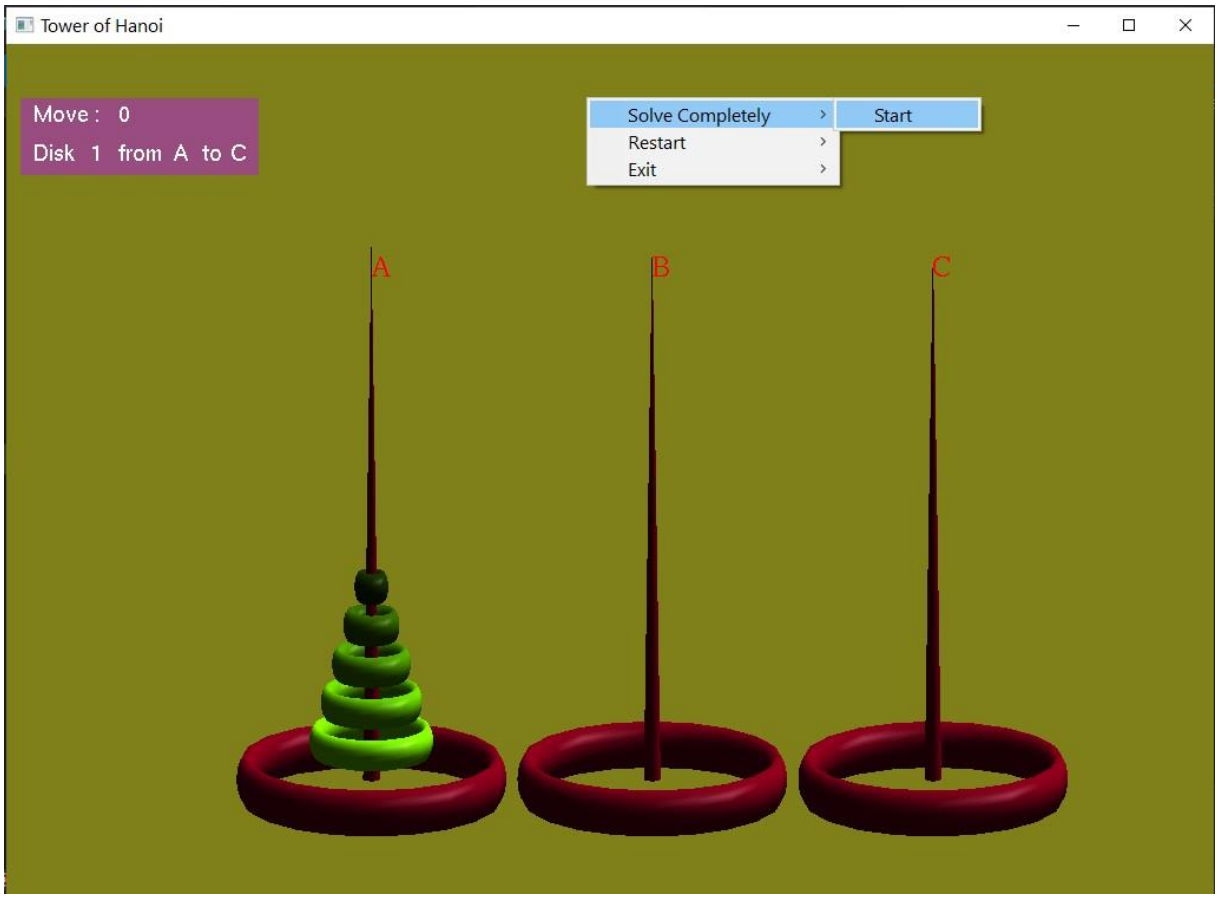

Figure 5.2: Tower of Hanoi Screen with menu on right click

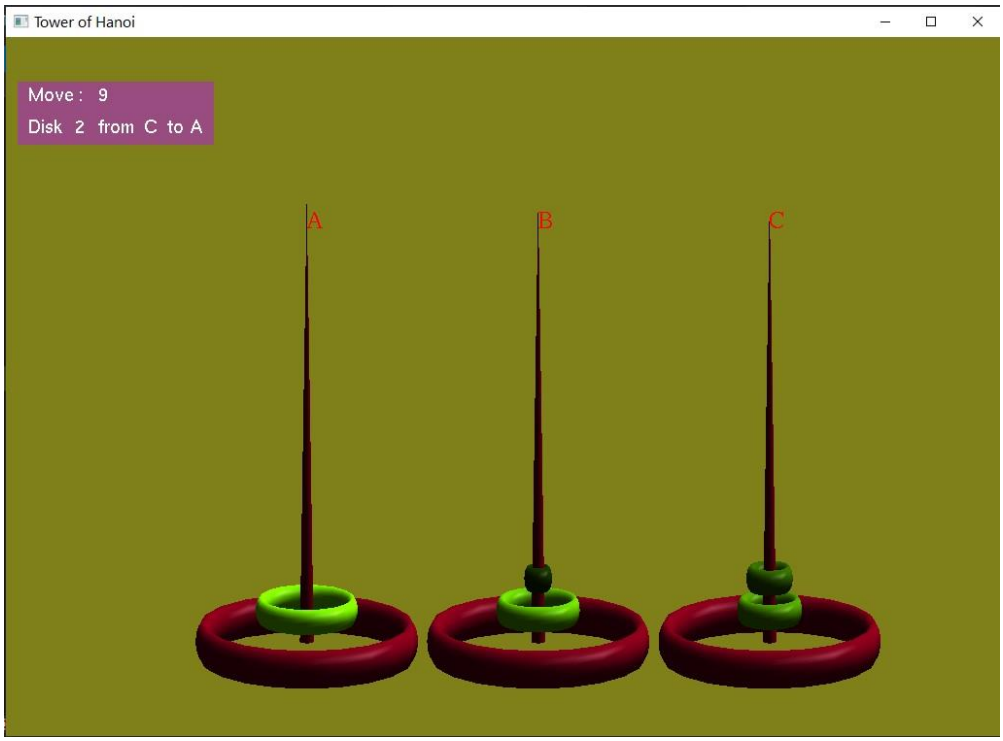

Figure 5.3: Screen during animation

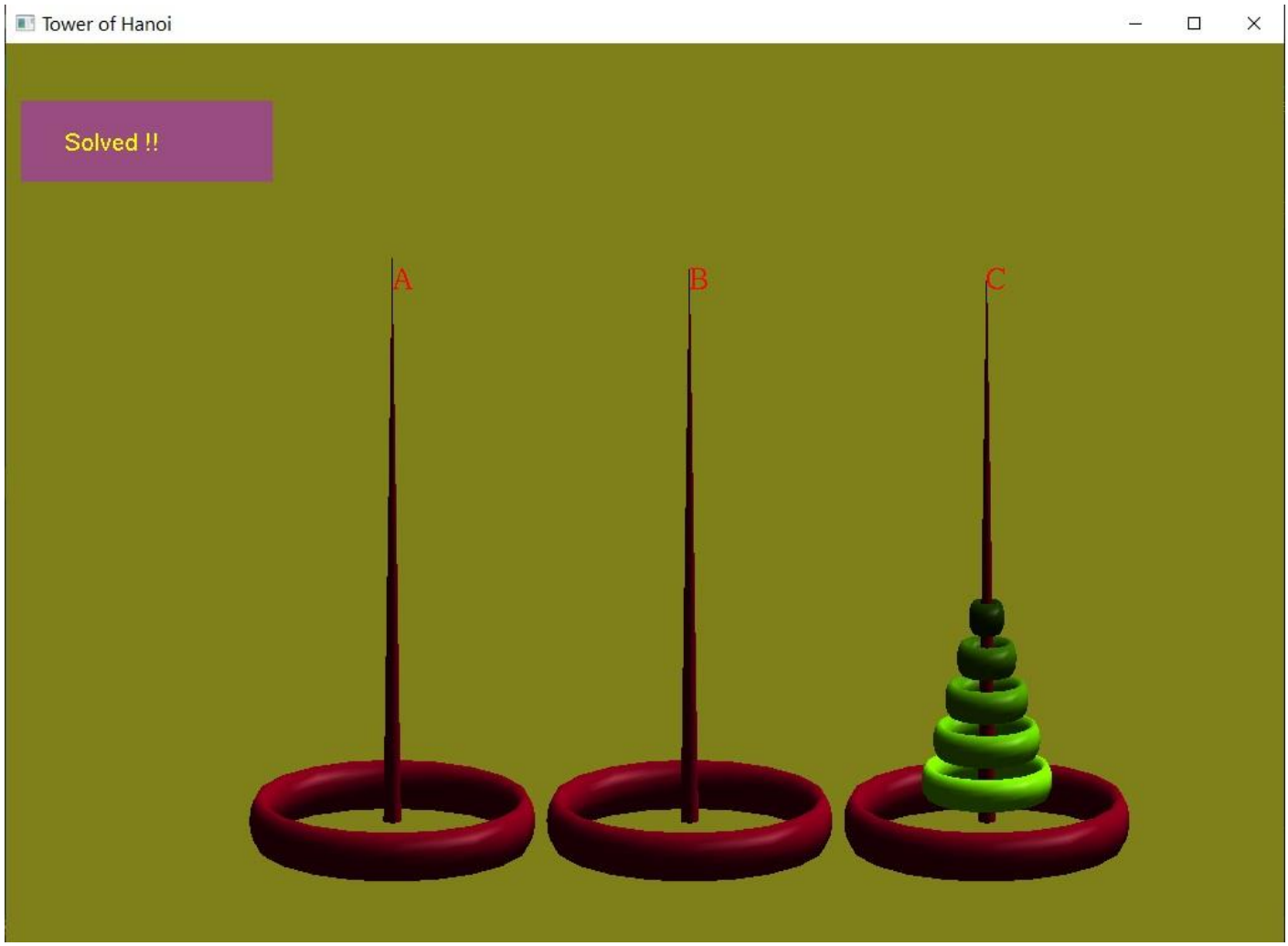

Figure 5.3: Screen after animation

CONCLUSION

Tower of Hanoi Animation 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 development of Tower of Hanoi Animation has given a good exposure to OpenGL by which I have learnt some of the techniques. This mini project on data structure using OpenGL is a graphics package that provides the user the platform to go through the solution of Tower of Hanoi problem.

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.

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]. https://open.gl/

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