CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Graphics provides one of the most natural means of communicating with a computer, since highly developed 2D and 3D pattern recognition abilities allow to perceive and process pictorial data rapidly and efficiently.

Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, can make pictures not only of concrete real-world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

1.2 PROBLEM STATEMENT

The aim of this project is to show the simulation of Towers of Hanoi using OpenGL which include changing visual properties, redisplay functionality and keyboard button press actions. The package must also have a user-friendly interface.

The input cannot be given by the user and hence the project is not dynamic in nature. Few parameters can be changed in the #define constructs.

1.3 MOTIVATION

Computer Graphics involves the usage of the libraries provided by OpenGL in order to visually represent the logic that we program. This can give us a better idea as to how Algorithms and Data Structures function. This was the motivation behind identifying the logic of Towers of Hanoi. Recursion can be given a rigorous mathematical formalism with the theory of dynamic programming and Towers of Hanoi can be used as an example of recursion when teaching programming.

1.4 COMPUTER GRAPHICS

Computer graphics and multimedia technologies are becoming widely used in educational applications because they facilitate non-linear, self-learning environments that particularly suited to abstract concepts and technical information.

Computer graphics are pictures and films created using computers. Usually, the term refers to computer-generated image data created with help from specialized graphical hardware and software. It is a vast and recent area in computer science. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, though sometimes erroneously referred to as CGI.

Important topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics.

Computer graphics is responsible for displaying art and image data effectively and meaningfully to the user. It is also used for processing image data received from the physical world. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design generally.

1.5 OPEN GL

OpenGL (Open Graphics Library) is a standard specification defining a cross language cross platform API for writing applications that produce 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 Window Platforms.

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

OpenGL Graphics Architecture

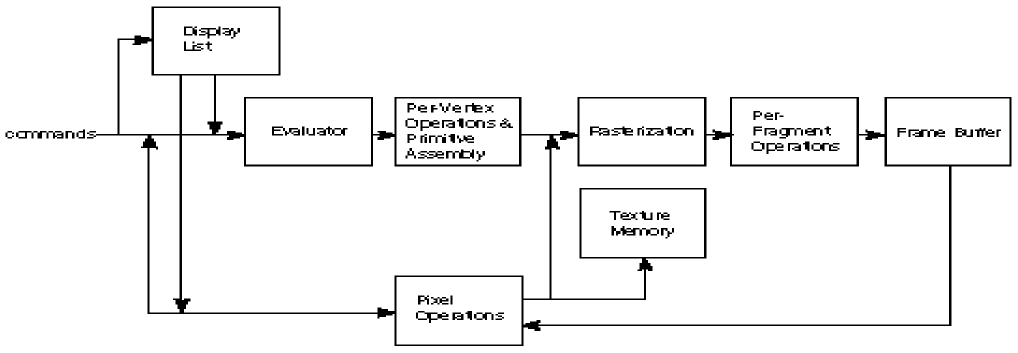

Figure 1.1 OpenGL Graphics Architecture

Display Lists:

All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators:

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions.

Per Vertex Operations:

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

Primitive Assembly:

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half space, defined by a plane.

Pixel Operation:

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step.

Rasterization:

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Color and depth values are assigned for each fragment square.

Fragment Operations:

Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

1.6 APPLICATION OF COMPUTER GRAPHICS

We can classify applications of computer graphics into four main areas:

1.Display of Information
2.Design
3.Simulation
4.Interfaces

Although many applications span two, three, or even all of these areas, the development of the field was based, for the most part, on separate work in each domain.

1.Display of Information

Graphics has always been associated with the display of information. Examples of the use of orthographic projections to display floorplans of buildings can be found on 4000-year-old Babylonian stone tablets. Mechanical methods for creating perspective drawings were developed during the Renaissance. Countless engineering students have become familiar with interpreting data plotted on log paper. More recently, software packages that allow interactive design of charts incorporating color, multiple data sets, and alternate plotting methods have become the norm. In fields such as architecture and mechanical design, hand drafting is being replaced by computer-based drafting systems using plotters and workstations. Medical imaging uses computer graphics in a number of exciting ways.

2. Design

Professions such as engineering and architecture are concerned with design. Although their applications vary, most designers face similar difficulties and use similar methodologies. One of the principal characteristics of most design problems is the lack of a unique solution. Hence, the designer will examine a potential design and then will modify it, possibly many times, in an attempt to achieve a better solution. Computer graphics has become an indispensable element in this iterative process.

3. Simulation

Some of the most impressive and familiar uses of computer graphics can be classified as simulations. Video games demonstrate both the visual appeal of computer graphics and our ability to generate complex imagery in real time. Computer-generated images are also the heart of flight simulators, which have become the standard method for training pilots.

4. User Interfaces

The interface between the human and the computer has been radically altered by the use of computer graphics. Consider the electronic office. The figures in this book were produced through just such an interface. A secretary sits at a workstation, rather than at a desk equipped with a typewriter. This user has a pointing device, such as a mouse, that allows him to communicate with the workstation.

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 HARDWARE REQUIREMENTS

- Processor: i3 or i5 processor
- VGA 640x480 or higher-resolution screen supported by Microsoft Windows.
- Recommended 2GB RAM or Higher
- 100GB SATA (Serial Advanced Technology Attachment) Hard Drive
- 5400 RPM hard disk drive
- DirectX 9 capable video card running at 1024 x 768
- DVD-ROM Drive
- Input devices: Keyboard, Mouse
- Output devices: Monitor
- Visual Display Unit
- Display Processor – AMD

2.2 SOFTWARE REQUIREMENTS

- Operating System- Windows 10
- GLUT library and OpenGL package
- Microsoft Visual basic C++ 2010 version

CHAPTER 3

SYSTEM DESIGN

3.1 PROPOSED SYSTEM

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:

- Only one disk can be moved at a time.
- 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.
- 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

3.1.1 ALGORITHM:

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 as A, B and 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.

3.1 DATA FLOW DIAGRAM

The interaction with the windows is initialized using glutInit() OpenGL API. The display mode-double buffer and depth buffer is, various callback functions for drawing and redrawing, for mouse and keyboard interfaces, input and calculate functions for various mathematical calculations, the window position and size are also initialized and create the window to display the output.

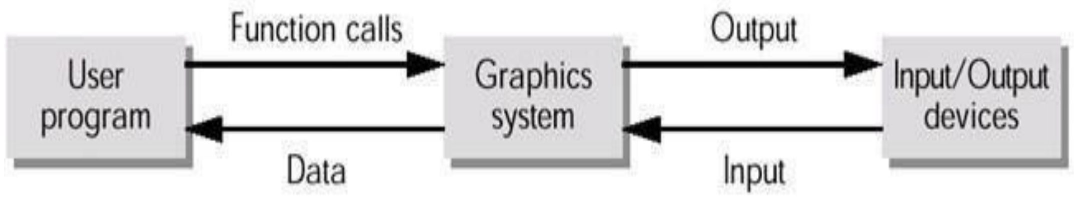

Fig 3.1 Graphics function flow

3.1 FLOWCHART

A simple flow chart describing the events occurring is described below.

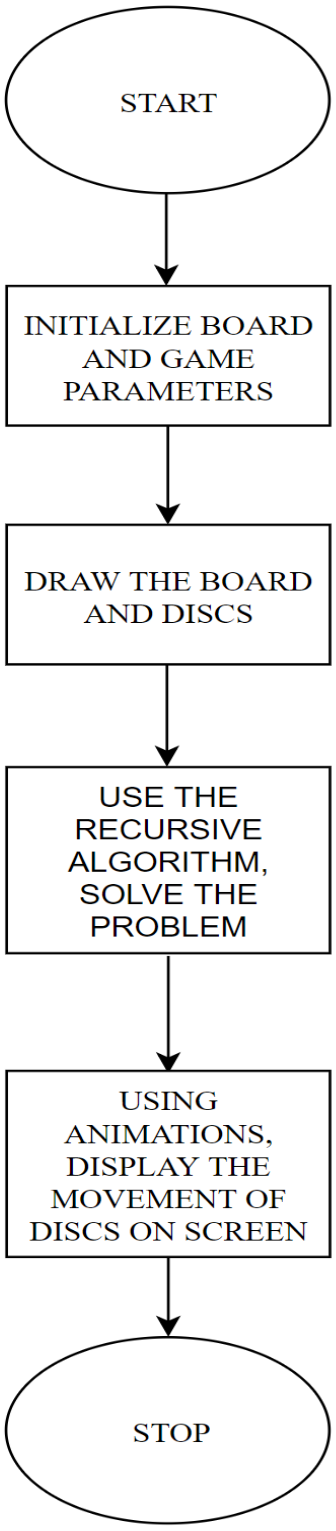
START
INITIALIZE BOARD AND GAME PARAMETERS
DRAW THE BOARD AND DISCS
USE THE RECURSIVE ALGORITHM, SOLVE THE PROBLEM
USING ANIMATIONS, DISPLAY THE MOVEMENT OF DISCS ON SCREEN
STOP

Figure 3.2 – Flow chart of the application

CHAPTER 4

IMPLEMENTATION

4.1 MODULE DESCRIPTION

- void initialize() – This function initializes the display parameters
- void initialize_game() – This function initializes the parameters required for Pegs and Game board.
- void display_handler() – This is the display handler for the second window.
- void anim_handler() – This function handles all the animation required for the game.
- void move_disc(int from_rod , int to_rod) – This function moves the disc from the first peg(from_rod) to second_peg(to_rod).
- void move_stack(int n, int f, int t) – This fuction is used to obtain the recursive solution to the problem.
- int main2() – The function provides the diver code for the second output window.
- void solve() – This function starts solving the problem from the initial state.
- int main() – This function provides the driver code for the first output window.

4.2 HIGH LEVEL CODE

4.2.1 BUILT-IN FUNCTIONS

- **void glClear(glEnum mode);**
  Clears the buffers namely color buffer and depth buffer. Mode refers to GL_COLOR_BUFFER_BIT or DEPTH_BUFFER_BIT.

- **void glutBitmapCharacter(void *font, int character);**
  Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font. The fonts used are GLUT_BITMAP_HELVETICA_18.

- **void glutInit(int *argc, char **argv);**
  Initializes GLUT; the arguments from main are passed in and can be used by the application.

- **void glutCreateWindow(char *title);**
  Creates a window on display; the string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

- **void glutMainLoop();**
  Causes the program to enter an event-processing loop.

- **void glutDisplayFunc(void (*func)(void))**
  Registers the display function func that is executed when the window needs to be redrawn.

- **void glColor3[b I f d ub us ui](TYPE r, TYPE g, TYPE b)**
  Sets the present RGB colors. Valid types are bytes(b), int(i), float(f), double(d), unsigned byte(ub), unsigned short(us), and unsigned int(ui). The maximum and minimum value for floating point types are 1.0 and 0.0 respectively, whereas the maximum and minimum values of the discrete types are those of the type, for eg, 255 and 0 for unsigned bytes.

- **void glutInitWindowSize(int width, int height);**
  Specifies the initial height and width of the window in pixels.

4.2.2 User implementation code

- **Initializing the OpenGL parameters – void initialize()**

```
void initialize()
{
        glClearColor(0,0,0,0);
        //Setting the clear color
        //glShadeModel(GL_SMOOTH);
        //SMOOTH Shading
        glEnable(GL_DEPTH_TEST);            //Enabling Depth Test
        //Setting Light0 parameters
        GLfloat light0_pos[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        // A positional light
```

```
                glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);
                glEnable(GL_LIGHTING);
                //Enabling Lighting
                glEnable(GL_LIGHT0);
                //Enabling Light0
                initialize_game();
        }
```

- **Initializing the game parameters and other variables – void initialize_game()**

```
void initialize_game()
{
    //Initializing 1)GameBoard t_board 2) Discs discs  3) ActiveDisc active_disc
    //1) Initializing GameBoard
    t_board.rod_base_rad = 1.0;
    t_board.x_min = 0.0;
    t_board.x_max = BOARD_X * t_board.rod_base_rad;
    t_board.y_min = 0.0;
    t_board.y_max = BOARD_Y * t_board.rod_base_rad;

    double x_center = (t_board.x_max - t_board.x_min) / 2.0;
    double y_center = (t_board.y_max - t_board.y_min) / 2.0;

    double dx = (t_board.x_max - t_board.x_min) / 3.0; //Since 3 rods
    double r = t_board.rod_base_rad;

   //Initializing Rods Occupancy value
    for (int i = 0; i < 3; i++)
    {
            for (int h = 0; h < NUM_DISCS; h++)
            {
                    if (i == 0)
                    {
                            t_board.rods[i].occupancy_val[h]= NUM_DISCS - 1 - h;
                    }
                    else
                            t_board.rods[i].occupancy_val[h] = -1;
    }
}
//Initializing Rod positions
    for (int i = 0; i < 3; i ++)
    {
            for (int h = 0; h < NUM_DISCS; h++)
            {
                    double x = x_center + ((int)i - 1) * dx;
                    double y = y_center;
                    double z = (h + 1) * DISC_SPACING;
                    Vector3& pos_to_set = t_board.rods[i].positions[h];
                    pos_to_set.x = x;
                    pos_to_set.y = y;
                    pos_to_set.z = z;
                    printf("%f %f %f \n",x,y,z);
            }
    }

//2) Initializing Discs
    for (size_t i = 0; i < NUM_DISCS; i++)
    {
            discs[i].position = t_board.rods[0].positions[NUM_DISCS - i - 1];
            //Normals are initialized whie creating a Disc object - ie in constructor of Disc
    }
//3) Initializing Active Disc
    active_disc.disc_index = -1;
    active_disc.is_in_motion = false;
    active_disc.step_u = 0.015;
    active_disc.u = 0.0;
    active_disc.direction = 0;
}
```

- **Function for drawing a solid cylinder at a given position, radius and height.**

```
void draw_solid_cylinder(double x, double y, double r, double h)
{
        GLUquadric* q = gluNewQuadric();
        GLint slices = 50;
        GLint stacks = 10;
        glPushMatrix();
        glTranslatef(x, y, 0.0f);
        gluCylinder(q, r, r, h, slices, stacks);
        glTranslatef(0, 0, h);
        gluDisk(q, 0, r, slices, stacks);
        glPopMatrix();
        gluDeleteQuadric(q);
}
```

- **Functions for drawing the game boards and rods**

```
void draw_board_and_rods(GameBoard const& board)
{
        //Materials,
        GLfloat mat_white[] = { 1.0f, 1.0f, 1.0f, 1.0f };
        GLfloat mat_yellow[] = { 1.0f, 1.0f, 0.0f, 1.0f };

        glPushMatrix();
        //Drawing the Base Rectangle [where the rods are placed]
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE, mat_white);
        glBegin(GL_QUADS);
                glNormal3f(0.0f, 0.0f, 1.0f);
                glVertex2f(board.x_min, board.y_min);
                glVertex2f(board.x_min, board.y_max);
                glVertex2f(board.x_max, board.y_max);
                glVertex2f(board.x_max, board.y_min);
        glEnd();
        //Drawing Rods and Pedestals
        glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE, mat_yellow);

        double r = board.rod_base_rad;
        for (int i = 0; i < 3; i++)
        {
                Vector3 const& p = board.rods[i].positions[0];
                draw_solid_cylinder(p.x, p.y, r * 0.1, ROD_HEIGHT - 0.1);
                draw_solid_cylinder(p.x, p.y, r, 0.1);
        }

        glPopMatrix();
}
```

- **Functions for drawing the discs**

```
void draw_discs()
{
        int slices = 100;
        int stacks = 10;

        double rad;

        GLfloat r, g, b;
        GLfloat emission[] = { 0.4f, 0.4f, 0.4f, 1.0f };
        GLfloat no_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat material[] = { 1.0f, 1.0f, 1.0f, 1.0f };
        for (size_t i = 0; i < NUM_DISCS; i++)
        {
                switch (i)
                {
                case 0: r = 0; g = 0; b = 1;
                        break;
                case 1: r = 0; g = 1; b = 0;
```

```
                    break;
        case 2: r = 0, g = 1; b = 1;
                    break;
        case 3 : r = 1; g = 0; b =0 ;
                    break ;
        case 4 : r = 1; g = 0; b = 1;
                    break;
        case 5 : r = 1; g = 1; b = 0;
                    break;
        case 6 : r = 1 ; g = 1 ; b = 1 ;
                    break ;
        default: r = g = b = 1.0f;
                    break;
        };

        material[0] = r;
        material[1] = g;
        material[2] = b;
        glMaterialfv(GL_FRONT,          GL_AMBIENT_AND_DIFFUSE,
material);

        GLfloat u = 0.0f;
        //This part is written to highlight the disc in motion
        if (i == active_disc.disc_index)
        {
                    glMaterialfv(GL_FRONT, GL_EMISSION, emission);
                    u = active_disc.u;
        }

        GLfloat factor = 1.0f;
        switch (i) {
                    case 0: factor = 0.2;
                                break;
                    case 1: factor = 0.4;
```

```
                                break;
                        case 2: factor = 0.6;
                                break;
                        case 3: factor = 0.8;
                                break;
                        case 4 : factor = 1.2 ;
                                break ;
                        case 5 : factor = 1.4 ;
                                break ;
                        case 6 : factor = 1.6 ;
                                break ;
                        case 7 : factor = 1.8 ;
                                break ;
                        default: break;
                };
                rad = factor * t_board.rod_base_rad;
                int d = active_disc.direction;



                glPushMatrix();
                glTranslatef(discs[i].position.x, discs[i].position.y, discs[i].position.z);
                double theta = acos(discs[i].normal.z);
                theta *= 180.0f / PI;
                glRotatef(d * theta , 0.0f, 1.0f, 0.0f);
                glutSolidTorus(0.2 * t_board.rod_base_rad, rad, stacks, slices);
                glPopMatrix();

                glMaterialfv(GL_FRONT, GL_EMISSION, no_emission);
        }
}
```

- **The Recursive Solution to Towers of Hanoi Code**

```
void move_stack(int n, int f, int t)
{
        if (n == 1) {
                solution_pair s;
                s.f = f;
                s.t = t;
                sol.push_back(s);        //pushing the (from, to) pair of solution to a list
    [so that it can be animated later]
                moves++;
                cout << "From rod " << f << " to Rod " << t << endl;
                return;
        }
        move_stack(n - 1, f, 3 - t - f);
        move_stack(1, f, t);
        move_stack(n - 1, 3 - t - f, t);
    }
```

- **Keyboard Handler for the both the windows**

```
void keyboard_handler_for_intro(unsigned char key, int x, int y)
{

   //Console Outputs
        switch (key)
        {
        case 27:
        case 'q' :
        case 'Q' :
                exit(0);
                break;

        case 'h':
        case 'H':
                cout << "ESC: Quit" << endl;
                cout << "S: Solve from Initial State" << endl ;
                cout << "H: Help" << endl;
                break;
        case 'n':
        case 'N' :
                glutDisplayFunc(display_handler);
                glutReshapeFunc(reshape_handler);
                glutKeyboardFunc(keyboard_handler);
                glutIdleFunc(display_handler);
                main2();
        default:
                break;
        }
}
```

CHAPTER 5

RESULTS AND SNAPSHOTS

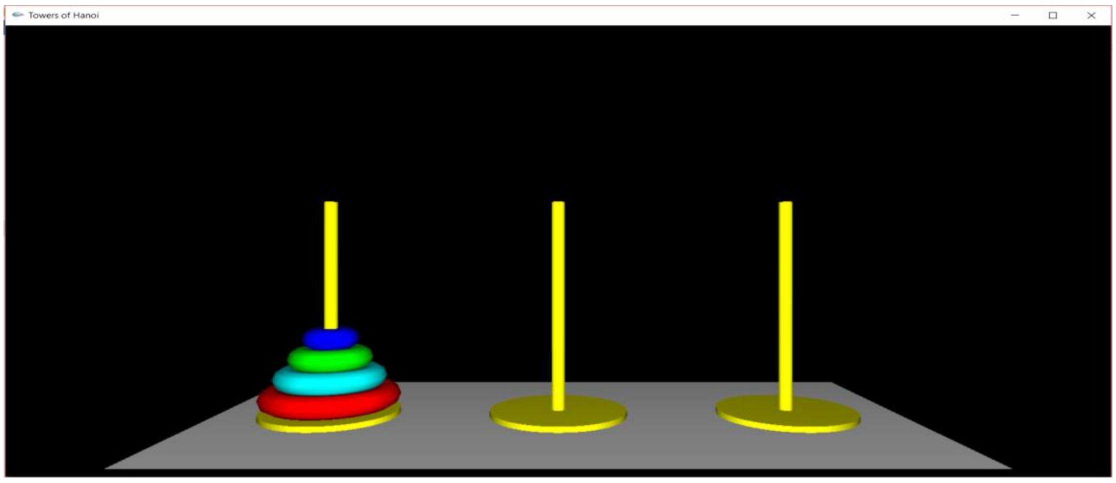

Figure 5.1 – Starting Position

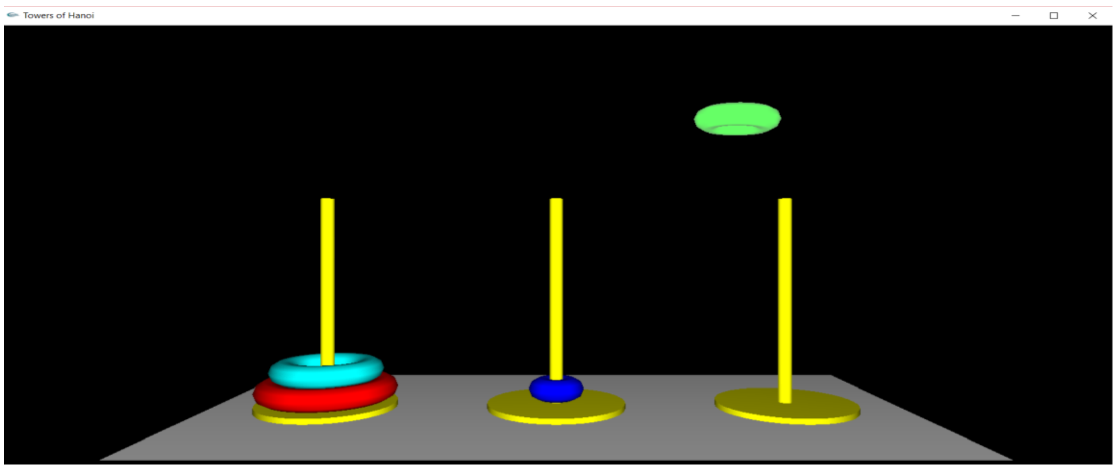

Figure 5.2 – Disc in Motion

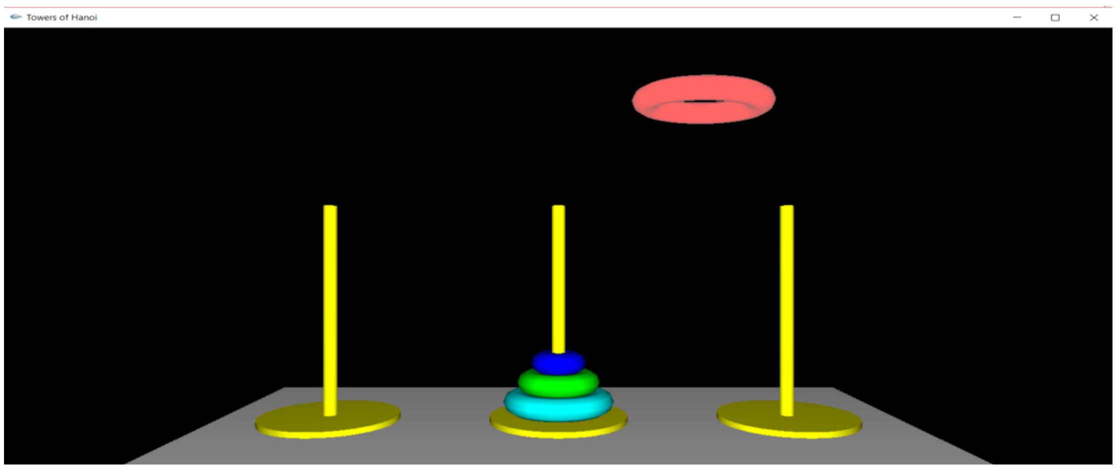

Figure 5.3 – Disc in Motion

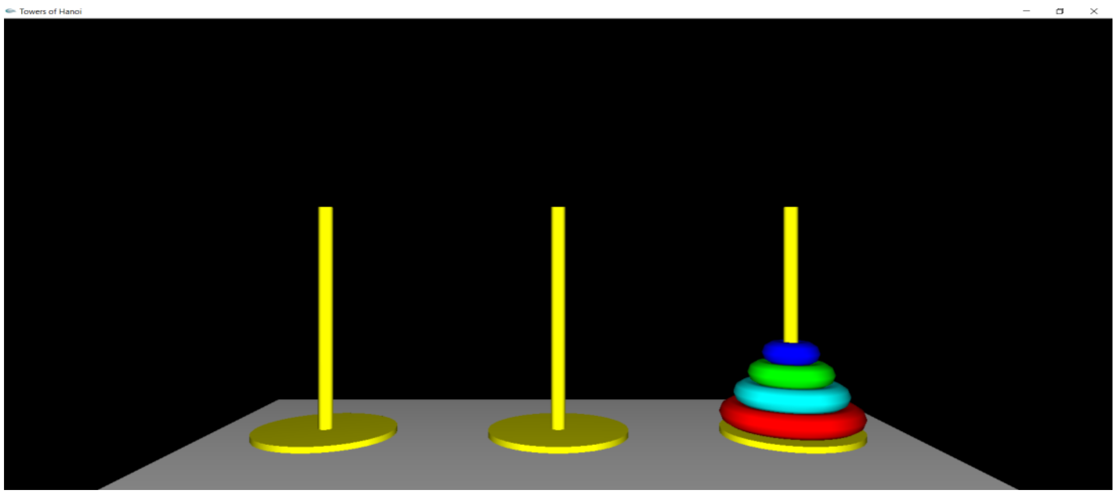

Figure 5.4 – Ending Position

CHAPTER 6

CONCLUSION

Towers of Hanoi Simulation is designed and implemented using a graphics software system called Open GL which became a widely accepted standard for developing graphics application.

Usage of Open GL functions and primitives are well understood and henceforth can be applied for real time applications.

This project is both informative and entertaining. This project provided an opportunity to learn the various concepts of the subject in detail and provided a platform to express creativity and imagination come true.

6.1 FUTURE ENHANCEMENTS

Further animation can be included to enhance the project’s look and feel.

CHAPTER 7

BIBLIOGRAPHY

[1] Edward Angel: Interactive Computer Graphics: A Top-Down Approach with 5th Edition,Addison-Wesley,2008.

[2] Donald Hearn and Pauline Baker: OpenGLVersion 3, 3rd Edition, Pearson Education, 2004.

[3] F.S.Hill Jr.: Computer Graphics Using OpenGL, 3rd Edition, PHI, 2009.

[4] James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes, Computer Graphics, Pearson Education, 1997

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

[6] OpenGL Documentation – https://www.opengl.org/documentation