VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI-590018

COMPUTER GRAPHICS AND IMAGE PROCESSING (21CSL66)

Mini Project Report

on

3D Graphical Simulation of the Tower of Hanoi

Submitted in partial fulfillment of the requirements for the VI semester

Computer Science and Engineering of Visvesvaraya Technological University, Belagavi

Submitted by:

Eshwar K 1RN21CS056

Under the Guidance of:

Dr. Sudhamani M J
Associate Professor

Department of Computer Science and Engineering
RNS Institute of Technology
Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098

2024

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ESTD : 2001

CERTIFICATE

This is to certify that the mini project work entitled **3D Graphical Simulation of the Tower of Hanoi** has been successfully carried out by **Eshwar k** bearing USN **1RN21CS056**, bonafide student of **RNS Institute of Technology** in partial fulfillment of the requirements for the 6th semester **Computer Science and Engineering of Visvesvaraya Technological University"**, Belagavi, during academic year 2024. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the COMPUTER GRAPHICS AND IMAGE PROCESSING laboratory requirements of 6th semester BE, CSE.

Signature of the Guide **Dr. Sudhamani M J** Associate Professor Dept. of CSE	Signature of the HoD **Dr. Kiran P** Professor & Head Dept. of CSE	Signature of the Principal **Dr. Ramesh Babu H S** Principal

PRINCIPAL
R.N.S.I.T., Bengaluru-98

External Viva:

Name of the Examiners	Signature with Date
1. Dr. Vinayath K.P	5/8/24
2. Savitha. T	5/8/24

Acknowledgement

The successful completion of any achievement is not solely dependent on individual efforts but also on the guidance, encouragement, and cooperation of intellectuals, elders, and friends. We would like to take this opportunity to express our heartfelt gratitude to all those who have contributed to the successful execution of this project.

First and foremost, we extend our profound thanks to **Sri. Satish R Shetty**, Managing Trustee of R N Shetty Trust and Chairman of RNS Group of Institutions, and **Sri. Karan S Shetty** , CEO of RNS Group of Institutions, Bengaluru, for providing a conducive environment that facilitated the successful completion of this project.

We would also like to express our sincere appreciation to our esteemed Director, **Dr. M K Venkatesha**, for providing us with the necessary facilities and support throughout the duration of this work.

Our heartfelt thanks go to our respected Principal, **Dr. Ramesh Babu H S**, for his unwavering support, guidance, and encouragement that played a vital role in the completion of this project.

We would like to extend our wholehearted gratitude to our HOD, **Dr. Kiran P**, Professor, and Head of the Department of Computer Science & Engineering, RNSIT, Bangalore, for his valuable suggestions and expert advice, which greatly contributed to the success of this endeavor.

A special word of thanks is due to our project guide,**Dr. Sudhamani M J**, Associate Professor in the Department of CSE, RNSIT, Bangalore, for her exceptional guidance, constant encouragement, and unwavering assistance throughout the project.

We would also like to express our sincere appreciation to all the teaching and non-teaching staff of the Department of Computer Science & Engineering, RNSIT, for their consistent support and encouragement.

Once again, we express our deepest gratitude to everyone involved, as their support and cooperation were instrumental in the successful completion of this project.

Abstract

The project implements a graphical simulation of the Tower of Hanoi problem using OpenGL and GLUT, providing an educational and interactive 3D visualization. Users can input the number of disks, and the program calculates and displays the total number of moves required to solve the puzzle. The simulation features dynamic real-time animation of the disk movements, demonstrating the algorithm's steps. Users can control the viewing angles via mouse interactions and access a menu to toggle the solution animation or quit the application. The code efficiently manages stack operations for disk handling and employs a recursive algorithm to solve the Tower of Hanoi problem. The program includes enhanced visual features such as distinct colors for disks and pegs, smooth shading, and lighting effects for improved clarity. This project, developed by Eshwar K (1RN21CS056) as a Computer Graphics mini-project, aims to provide a comprehensive understanding of both the algorithm and the graphical capabilities of OpenGL.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of Computer Graphics

The term computer graphics has been used in a broad sense to describe almost everything on computers that is not text or sound. Typically, the term computer graphics refers to several different things:

- The representation and manipulation of image data by a computer.
- The various technologies used to create and manipulate images.
- The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

Today, computers and computer-generated images touch many aspects of daily life. Computer images is found on television, in newspapers, for example in weather reports, in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media such graphs are used to illustrate papers, reports, thesis, and other presentation material.Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D, 3D, 4D, 7D, and animated graphics.

As technology has improved, 3D computer graphics have become more common. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis

is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic component.

1.2 History of Computer Graphics

In 1959, the TX-2 computer was developed at MIT's Lincoln Laboratory. The TX-2 integrated a number of new man-machine interfaces. A light pen could be used to draw sketches on the computer using Ivan Sutherland's revolutionary Sketchpad software.Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location.

In 1961 another student at MIT, Steve Russell, created the first video game, E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963. During 1970s, the first major advance in 3D computer graphics was created at University of Utah by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image.

In the 1980s, artists and graphic designers began to see the personal computer, particularly the Commodore Amiga and Macintosh, as a serious design tool, one that could save time and draw more accurately than other methods. In the late 1980s, SGI computers were used to create some of the first fully computer-generated short films at Pixar. The Macintosh remains a highly popular tool for computer graphics among graphic design studios and businesses. Modern computers, dating from the 1980s often use graphical user interfaces (GUI) to present data and information with symbols, icons and pictures, rather than text. Graphics are one of the five key elements of multimedia technology.

3D graphics became more popular in the 1990s in gaming, multimedia and animation. In 1996, Quake, one of the first fully 3D games, was released. In 1995, Toy Story, the first full-length computer-generated animation film, was released in cinemas worldwide. Since then, computer graphics have only become more detailed and realistic, due to more powerful graphics hardware and 3D modeling software.

1.3 Applications of Computer Graphics

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

- Display of information
- Design
- Simulation and animation
- User interfaces

1.3.1 Display of information

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

1.3.2 Design

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

1.3.3 Simulation and Animation

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

1.3.4 User interfaces

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

1.4 Overview of Image Processing

Image processing is a method to perform operations on an image to enhance it or extract useful information. It is a type of signal processing where the input is an image and the output can be either an image or a set of characteristics/features related to the image. It involves various steps like image acquisition, enhancement, restoration, segmentation, and more. Following subsections describe the basic steps in in Image Processing

1.4.1 Image Acquisition

Image acquisition is the first step in image processing. It involves capturing an image using a sensor (such as a camera) and converting it into a digital form that can be processed by a computer. The quality of the acquired image significantly affects the subsequent processing steps. Examples include capturing a photograph using a digital camera or scanning a document using a scanner.

1.4.2 Image Enhancement

Image enhancement involves improving the visual appearance of an image or converting the image to a form better suited for analysis by a human or machine. Techniques used for image enhancement include:

- **Contrast Adjustment**: Modifying the contrast of an image to make it more suitable for visual interpretation.
- **Histogram Equalization**: Improving the contrast of an image by redistributing the intensity values.
- **Noise Reduction**: Removing noise from an image using filters such as Gaussian, median, or adaptive filters.

1.4.3 Image Restoration

Image restoration aims to recover an image that has been degraded by known factors. This step involves reversing the degradation to retrieve the original image. Common techniques include:

- **De-blurring**: Removing blurriness caused by motion or out-of-focus lenses.
- **Noise Filtering**: Reducing noise while preserving important details and edges in the image.
- **Inverse Filtering**: Applying the inverse of the degradation function to restore the image.

1.4.4 Color Image Processing

Color image processing involves the manipulation of color images and is a crucial area in image processing as it adds an extra dimension to the visual content. The primary goals are color correction, color enhancement, and color space transformation.

Color Models

Different color models represent colors in various ways, each suitable for different applications. Common color models include:

- **RGB (Red, Green, Blue)**: Used in electronic displays and cameras.
- **CMY(K) (Cyan, Magenta, Yellow, Black)**: Used in color printing.
- **HSV (Hue, Saturation, Value)**: Useful for color analysis and manipulation.

Color Space Transformations

Transforming an image from one color space to another is essential for various processing techniques. For instance:

- **RGB to Grayscale**: Converts a color image to grayscale by removing hue and saturation information while retaining luminance.
- **RGB to HSV**: Helps in separating color information (hue) from intensity information (value).

Color Correction and Enhancement

Improving the color quality of an image involves techniques like:

- **White Balance Adjustment**: Corrects color casts due to different lighting conditions.

- **Histogram Equalization**: Enhances contrast by adjusting the intensity distribution.

1.4.5 Image Segmentation

Dividing an image into meaningful segments or regions is crucial for further analysis and interpretation. Techniques include:

- **Thresholding**: Simple and effective for binary segmentation based on pixel intensity.
- **Edge Detection**: Identifies object boundaries using operators like Sobel, Canny, and Prewitt.
- **Clustering**: Groups pixels with similar attributes using algorithms like K-means and Mean Shift.

1.4.6 Image Compression

Reducing the size of an image file without significantly degrading its quality is vital for storage and transmission. There are two main types of compression:

- **Lossless Compression**: Reduces file size without losing any data (e.g., PNG, GIF).
- **Lossy Compression**: Reduces file size by sacrificing some quality (e.g., JPEG).

1.4.7 Image Representation and Description

Transforming image data into a form suitable for computer processing involves various techniques:

- **Shape Representation**: Uses boundaries and regions to describe object shapes.
- **Boundary Descriptors**: Includes curvature, Fourier descriptors, and chain codes.
- **Regional Descriptors**: Characterizes objects based on their region properties like area, centroid, and texture.

1.4.8 Image Recognition

Identifying and classifying objects within an image is essential for many applications. Techniques include:

- **Pattern Recognition**: Utilizes statistical methods to recognize patterns in data.
- **Neural Networks**: Employs deep learning for highly accurate image recognition.

- **Machine Learning Algorithms**: Uses algorithms like Support Vector Machines (SVM) and Random Forest for classification tasks.

1.5 Applications

1.5.1 Medical Imaging

Medical imaging involves enhancing and analyzing medical images for diagnostic and treatment purposes. Techniques such as MRI, CT scans, and X-rays are processed to improve clarity and assist in accurate diagnosis.

1.5.2 Remote Sensing

Remote sensing involves processing satellite or aerial images to monitor and analyze earth's surface. Applications include environmental monitoring, agricultural assessment, and urban planning.

1.5.3 Computer Vision

Computer vision enables machines to interpret and make decisions based on visual data. Applications include self-driving cars, facial recognition, and automated inspection systems.

1.5.4 Industrial Inspection

Industrial inspection uses image processing for quality control and fault detection in manufacturing processes. Techniques include checking for defects, verifying dimensions, and ensuring product quality.

1.5.5 Security and Surveillance

Security and surveillance applications involve using image processing for tasks such as face recognition, motion detection, and behavior analysis to enhance security measures.

1.6 Conclusion

Image processing is integral to modern technology, driving advancements in numerous fields. With continuous advancements in computational power and algorithms, the capabilities and applications of image processing continue to expand, offering innovative solutions across various industries.

Chapter 2

OpenGL

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 three-dimensional 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. OpenGL provides a set of commands to render a three dimensional scene. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop OpenGL-applications without licensing. OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation. Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

2.1 OpenGL Libraries

Computer Graphics are created using OpenGL, which became a widely accepted standard software system for developing graphics applications. 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. OpenGL stands for 'open graphics library' graphics library is a collection of API's (Applications Programming Interface). Graphics library functions are:

- GL library (OpenGL in windows) – Main functions for windows.

- GLU (OpenGL utility library) - Creating and viewing objects.
- GLUT (OpenGL utility toolkit)- Functions that help in creating interface of windows

OpenGL draws primitives—points, line segments, or polygons—subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. These libraries are included in the application program using preprocessor directives OpenGL User Interface Library (GLUI) is a C++ user interface library based on the OpenGL Utility Toolkit (GLUT) which provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications. It is window and operating system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management. The OpenGL Utility Library (GLU) is a computer graphics library. It consists of a number of functions that use the base OpenGL library to provide higher-level drawing routines from the more primitive routines that OpenGL provides. It is usually distributed with the base OpenGL package.

2.2 OpenGL Contributions

It is very popular in the video games development industry where it competes with Direct3D (on Microsoft Windows).OpenGL is also used in CAD, virtual reality, and scientific visualization programs.OpenGL is very portable. It will run for nearly every platform in existence, and it will run well. It even runs on Windows NT 4.0 etc. The reason OpenGL runs for so many platforms is because of its Open Standard. OpenGL has a wide range of features, both in its core and through extensions. Its extension feature allows it to stay immediately current with new hardware features, despite the mess it can cause.

2.3 Limitations

- OpenGL is case sensitive.
- Line Color, Filled Faces and Fill Color not supported.
- Shadow plane is not supported.

Chapter 3

Resource Requirements

3.1 Hardware Requirements

The Hardware requirements are very minimal and the program can be run on most of the machines.Table 3.1 gives details of hardware requirements.

Table 3.1: Hardware Requirements

Processor	Intel Core i3 processor
Processor Speed	1.70 GHz
RAM	4 GB
Storage Space	40 GB
Monitor Resolution	1024*768 or 1336*768 or 1280*1024

3.2 Software Requirements

The software requirements are description of features and functionalities of the system.Table 3.2 gives details of software requirements.

Table 3.2: Software Requirements

Operating System	Windows 8.1
IDE	Microsoft Visual Studio with C++ 2022
OpenGL libraries	glut.h,glu32.lib,opengl32.lib,glut32.lib glu32.dll,glut32.dll,opengl32.dll.

Chapter 4

System Design

The description of all the functions used in the program is given below:

- **void glutInitDisplayMode(unsigned int mode):**
  This function requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE).

- **void glutInitWindowPosition(int x, int y):** This specifies the initial position of top-left corner of the windows in pixels.

- **void glutInitWindowSize(int width, int height):** This function specifies the initial height and width of the window in pixels.

- **void glutCreateWindow(char *title):** This function creates a window on the 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 glutDisplayFunc(void (*func)(void)):** This function registers the display func that is executed when the window needs to be redrawn.

- **void glClearColor(GLclampfr,GLclampfg GLclampfb,GLclampf a):** This sets the present RGBA clear colour used when clearing the colour buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

Chapter 5

Implementation

```
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Global variables for window dimensions and number of disks
GLfloat WIDTH = 1360;
GLfloat HEIGHT = 768;
GLint NUM_DISKS;

GLboolean motion = GL_FALSE;
GLfloat xangle = 0, yangle = 0;

// Macro for calculating the third peg index
#define other(i,j) (6-(i+j))
#define DISK_HEIGHT 35
#define CONE NUM_DISKS+1
#define HANOI_SOLVE 0
#define HANOI_QUIT 1

// Light colors for different lights
GLfloat lightTwoColor[] = { 1.0, 0.0, 1, 1.0 };
GLfloat lightZeroColor[] = { .3, .3, .3, .3 };

// Colors for disks and poles
```

```
GLfloat diskColor[] = { 0.0, 0.0, 1.0 };
GLfloat poleColor[] = { 1.0, 1.0, 1.0 };
GLfloat destinationPoleColor[] = { 1.0, 0.0, 0.0 };

// Definition of a stack node and stack structure for poles
typedef struct stack_node {
    int size;
    struct stack_node* next;
} stack_node;

typedef struct stack {
    struct stack_node* head;
    int depth;
} stack;

stack poles[4];

// Function to push a disk onto a pole
int push(int which, int size) {
    stack_node* next = (stack_node*)malloc(sizeof(stack_node));
    if (!next) {
        //standard error
        fprintf(stderr, "out of memory!\n");
        exit(-1);
    }
    next->size = size;
    next->next = poles[which].head;
    poles[which].head = next;
    poles[which].depth++;
    return 0;
}

// Function to pop a disk from a pole
int pop(int which) {
    int retval = poles[which].head->size;
    stack_node* temp = poles[which].head;
```

```
    poles[which].head = poles[which].head->next;
    poles[which].depth--;
    free(temp);
    return retval;
}

// Definition of a move node and move stack structure for moves
typedef struct move_node {
    int t, f;
    struct move_node* next;
} move_node;

typedef struct move_stack {
    int depth;
    struct move_node* head, * tail;
} move_stack;

move_stack moves;

// Function to initialize OpenGL and tower of Hanoi setup
void init(void) {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glShadeModel(GL_SMOOTH);
    int i;
    for (i = 0; i < 4; i++) {
        poles[i].head = NULL;
        poles[i].depth = 0;
    }
    moves.head = NULL;
    moves.tail = NULL;
    moves.depth = 0;

    // Create display lists for disks
    for (i = 1; i <= NUM_DISKS; i++) {
        glNewList(i, GL_COMPILE);
        {
```

```
            // here we are increasing the desk shape and size
            glutSolidTorus(DISK_HEIGHT / 2 + 2, 14 * i + 2, 3, 20);
        }
        glEndList();
    }

    // Create display list for cone (representing pegs)
    glNewList(CONE, GL_COMPILE);
    {
        glutSolidCone(5, (NUM_DISKS + 2) * DISK_HEIGHT, 50, 50);
    }
    glEndList();
}

// Function to pop a move from the move stack
void mpop(void) {
    move_node* temp = moves.head;
    moves.head = moves.head->next;
    free(temp);
    moves.depth--;
}

// Function to push a move onto the move stack
void mpush(int t, int f) {
    move_node* new1 = (move_node*)malloc(sizeof(move_node));
    new1->t = t;
    new1->f = f;
    new1->next = NULL;
    if (moves.tail)
        moves.tail->next = new1;
    moves.tail = new1;
    if (!moves.head)
        moves.head = moves.tail;
    moves.depth++;
}
```

```
// Function to update the display
void update(void) {
    while (motion == GL_TRUE && motion++) {
        glutPostRedisplay();
    }
}

// Function to draw a single peg
void DrawPost(GLfloat xcenter, int pegIndex) {
    glPushMatrix();
    {
        glTranslatef(xcenter, 0, 0);
        glRotatef(90, -1, 0, 0);
        if (pegIndex == 3) {
            glColor3fv(destinationPoleColor);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, destinationPoleColor);
        }
        else {
            glColor3fv(poleColor);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, poleColor);
        }
        glCallList(CONE);
    }
    glPopMatrix();
}

// Function to draw all three pegs
void DrawPosts(void) {
    glLineWidth(10);
    DrawPost((int)(WIDTH / 4), 1);
    DrawPost((int)(2 * WIDTH / 4), 2);
    DrawPost((int)(3 * WIDTH / 4), 3);
}

// Function to draw a single disk
void DrawDisk(GLfloat xcenter, GLfloat ycenter, GLfloat size) {
```

```
    glPushMatrix();
    {
        glTranslatef(xcenter, ycenter, 0);
        glRotatef(90, 1, 0, 0);
        glColor3fv(diskColor);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, diskColor);
        glCallList(size);
    }
    glPopMatrix();
}

// Function to draw all disks on all pegs
void DrawDisks(void) {
    int i;
    stack_node* temp;
    int xcenter, ycenter;
    for (i = 1; i <= 3; i++) {
        xcenter = i * WIDTH / 4;
        for (temp = poles[i].head, ycenter = DISK_HEIGHT * poles[i].depth -
            DISK_HEIGHT / 2; temp; temp = temp->next, ycenter -= DISK_HEIGHT) {
            DrawDisk(xcenter, ycenter, temp->size);
        }
    }
}

// Macro for pushing a move onto the move stack
#define MOVE(t,f) mpush((t),(f))

// Recursive function to solve Tower of Hanoi
static void mov(int n, int f, int t) {
    int o;
    if (n == 1) {
        MOVE(t, f);
        printf("\nDisk moves From Peg: %d -> Peg: %d", f, t);
        return;
    }
```

```
    o = other(f, t);
    mov(n - 1, f, o);
    mov(1, f, t);
    mov(n - 1, o, t);
}

// Function to draw the entire scene
void draw(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    {
        glTranslatef(WIDTH / 2, HEIGHT / 2, 0);
        glRotatef(xangle, 0, 1, 0);
        glRotatef(yangle, 1, 0, 0);
        glTranslatef(-WIDTH / 2, -HEIGHT / 2, 0);
        DrawPosts();
        DrawDisks();
    }
    glPopMatrix();

    if (motion && moves.depth) {
        int t = moves.head->t;
        int f = moves.head->f;
        push(t, pop(f));
        mpop();
    }
    glutSwapBuffers();
    update();
}

// Function to handle visibility events in GLUT
void hanoi_visibility(int state) {
    if (state == GLUT_VISIBLE && motion)
        update();
}
```

```
// Variables for mouse interaction
int moving = 0;
int startx, starty;

// Function to handle mouse button events in GLUT
void hanoi_mouse(int but, int state, int x, int y) {
    if (but == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        moving = 1;
        startx = x;
        starty = y;
    }
    if (but == GLUT_LEFT_BUTTON && state == GLUT_UP) {
        moving = 0;
    }
    if (but == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) {
        motion = !motion;
        update();
    }
}

// Function to handle mouse motion events in GLUT
void hanoi_motion(int x, int y) {
    if (moving) {
        xangle += (x - startx);
        yangle += (y - starty);
        startx = x;
        starty = y;
        glutPostRedisplay();
    }
}

// Function to handle menu selection in GLUT
void hanoi_menu(int val) {
    switch (val) {
    case HANOI_SOLVE: motion = !motion; update(); break;
    case HANOI_QUIT: exit(0);
```

```
    }
}

// Main function
int main(int argc, char* argv[]) {
    int i;

    printf("Enter the number of Disks: ");
    scanf_s("%d", &NUM_DISKS);

    double DISK_MOVES = pow(2.0, NUM_DISKS) - 1;
    printf("\nGuide: Dr Sudhamani MJ, Associate Profssor, CSE Dept");
    printf("\nThis is CG Mini Project, Made By Eshwar k (1RN21CS056)\n");
    printf("\nTotal number of Disk movements for %d disks : %.0lf\n", NUM_DISKS,
        DISK_MOVES); // Calculate and print total moves

    if (NUM_DISKS > 0) {
        glutInit(&argc, argv);
        glutInitWindowSize(1000, 600);
        glutInitWindowPosition(100, 100);
        glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
        glutCreateWindow("Tower of Hanoi");
        glutDisplayFunc(draw);

        glViewport(0, 0, (int)WIDTH, (int)HEIGHT);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0, WIDTH, -70.0, HEIGHT, -10000, 10000);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glClearColor(0, 1.0, 0, 0);
        glClearDepth(1.0);
        glEnable(GL_DEPTH_TEST);

        glLightfv(GL_LIGHT2, GL_DIFFUSE, lightTwoColor);
        glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, 10);
```

```
        glEnable(GL_LIGHT2);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
        glEnable(GL_LIGHT0);
        glEnable(GL_LIGHTING);

        glutMouseFunc(hanoi_mouse);
        glutMotionFunc(hanoi_motion);
        glutVisibilityFunc(hanoi_visibility);

        glutCreateMenu(hanoi_menu);
        glutAddMenuEntry("Solve", HANOI_SOLVE);
        glutAddMenuEntry("Quit", HANOI_QUIT);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        init();

        for (i = 0; i < NUM_DISKS; i++)
            push(1, NUM_DISKS - i);
        mov(NUM_DISKS, 1, 3);

        glutMainLoop();
        return 0;
    }
    else {
        exit(0);
    }
}
```

Chapter 6

Testing

In unit testing, the program modules that make up the system are tested individually. Unit testing focuses on locating errors in the working modules that are independent of each other. This enables the detection of errors in coding and the logic within the module alone. This testing is also used to ensure the integrity of the data stored. The various routines were checked by passing the inputs and the corresponding output is tested. Table 6.1 gives details of validation. Test cases used in the project are as follows:

Table 6.1: Test Case Validation

No.	Metric	Description	Observation
1.	Disk Movement	Verify disk movement according to Tower of Hanoi rules.	Correct disk movements observed.
2.	Display Function	Verify correct display of pegs and disks.	Correct rendering of pegs and disks.
3.	User Interaction	Test mouse clicks for motion control and view rotation.	Correct response to mouse inputs.
4.	Recursive Algorithm	Ensure accurate Tower of Hanoi solution.	Correct move sequence produced.
5.	Memory Management	Verify proper memory allocation/deallocation.	Efficient memory management.
6.	Menu Function	Test menu options for solving and quitting.	Menu options function correctly.
7.	Motion Toggle	Verify toggling motion with middle mouse button.	Motion toggles correctly.

Chapter 7

Results & Snapshots

Figure 7.1: User Input and Output Display in Debug Console

This figure shows the main screen of the command-line interface where user inputs are entered and corresponding outputs are displayed. The console accepts the number of disks for the Tower of Hanoi problem and initializes the simulation parameters. It displays real-time logs of the disk movements, ensuring that the steps taken by the recursive algorithm are accurately tracked. This helps in verifying the correctness of the solution by matching the moves with the expected sequence according to the Tower of Hanoi rules .

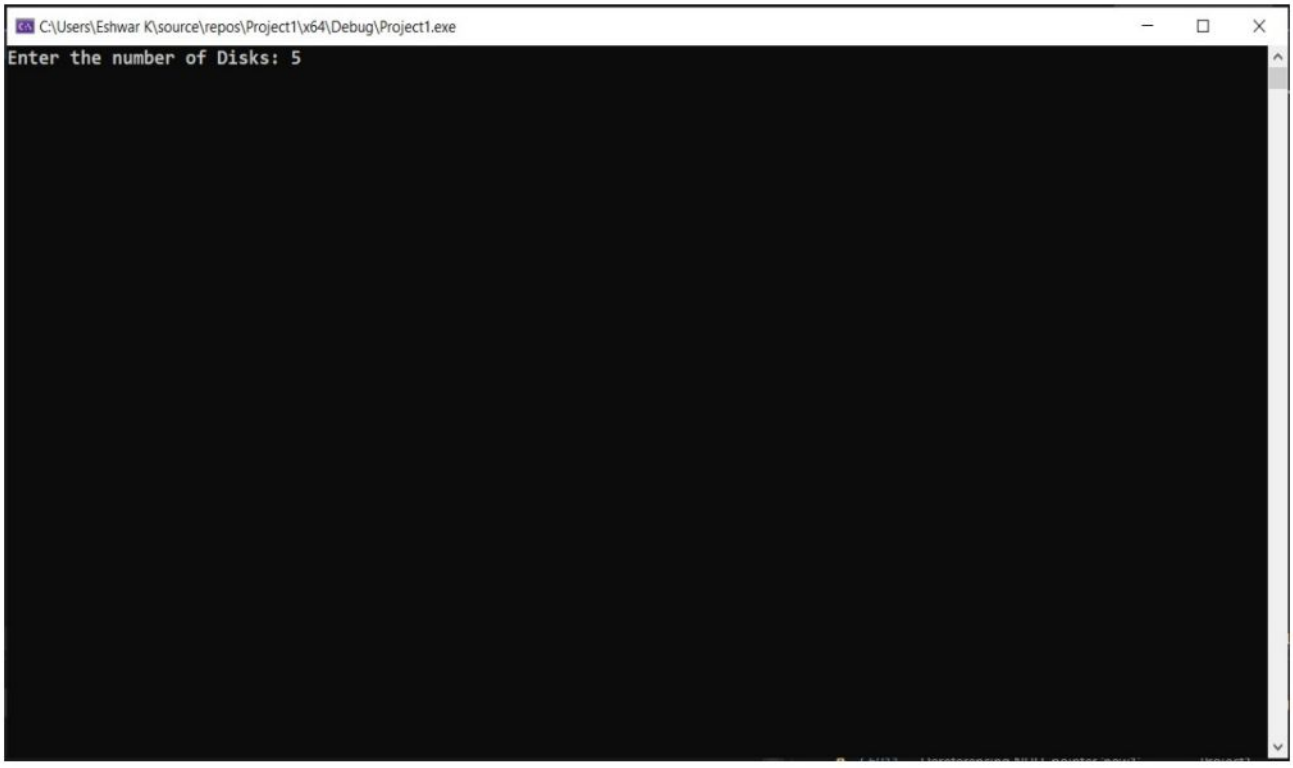

Figure 7.1: User Input and Output Display in Debug Console

Figure 7.2: Tower of Hanoi Solution Steps

This window presents the visual representation of the Tower of Hanoi solution steps. As the recursive algorithm progresses, the disks are moved from the source peg to the destination peg, following the rules of the puzzle. Each step is animated, allowing users to observe the transitions and understand how the algorithm solves the problem step-by-step. This visualization aids in comprehending the recursive nature and efficiency of the solution by providing a clear and interactive demonstration .

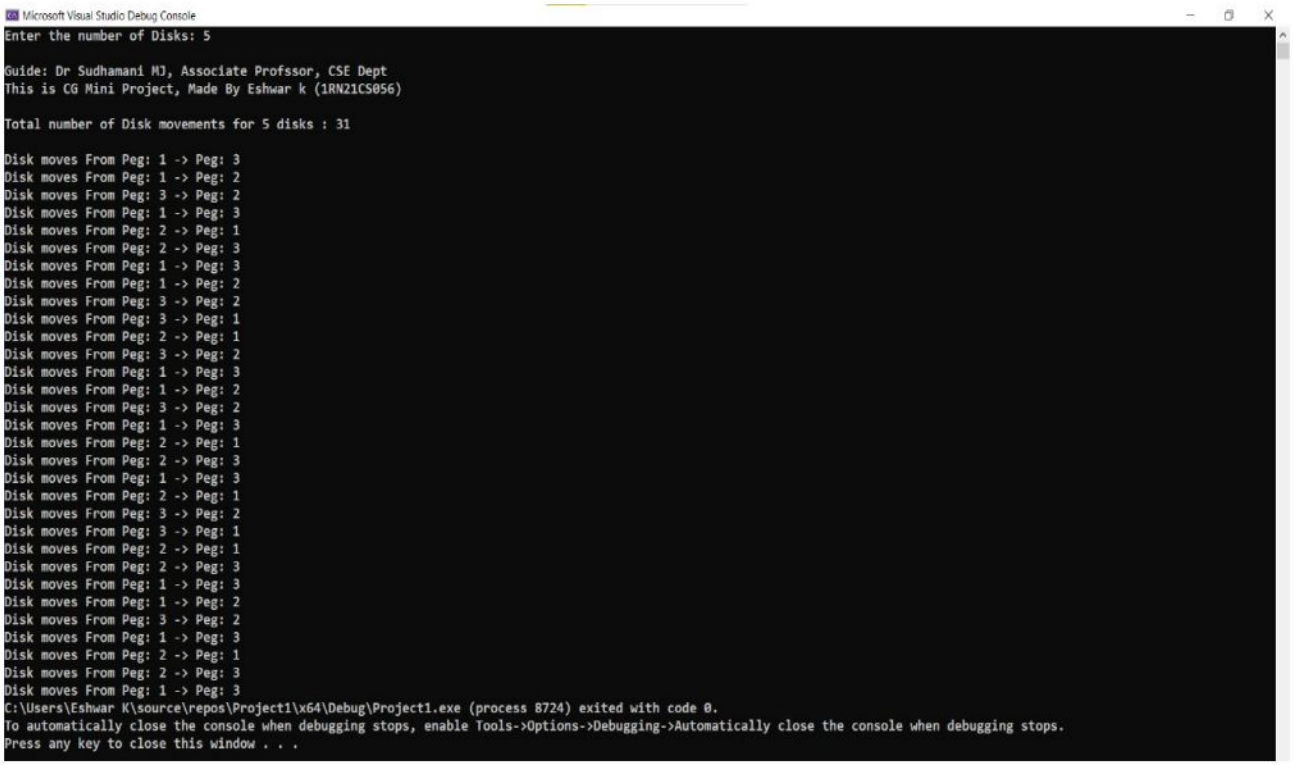

Figure 7.2: Tower of Hanoi Solution Steps

Figure 7.3: Initial Main Screen Post Disk Creation Based on User Input

After the user inputs the number of disks, this initial main screen shows the graphical setup with all the disks stacked on the source peg. The graphical user interface is prepared for the simulation, with the disks rendered according to the specified input. This stage sets up the visual context for the Tower of Hanoi problem, ensuring that the user can see the starting state of the puzzle before the algorithm begins its execution .

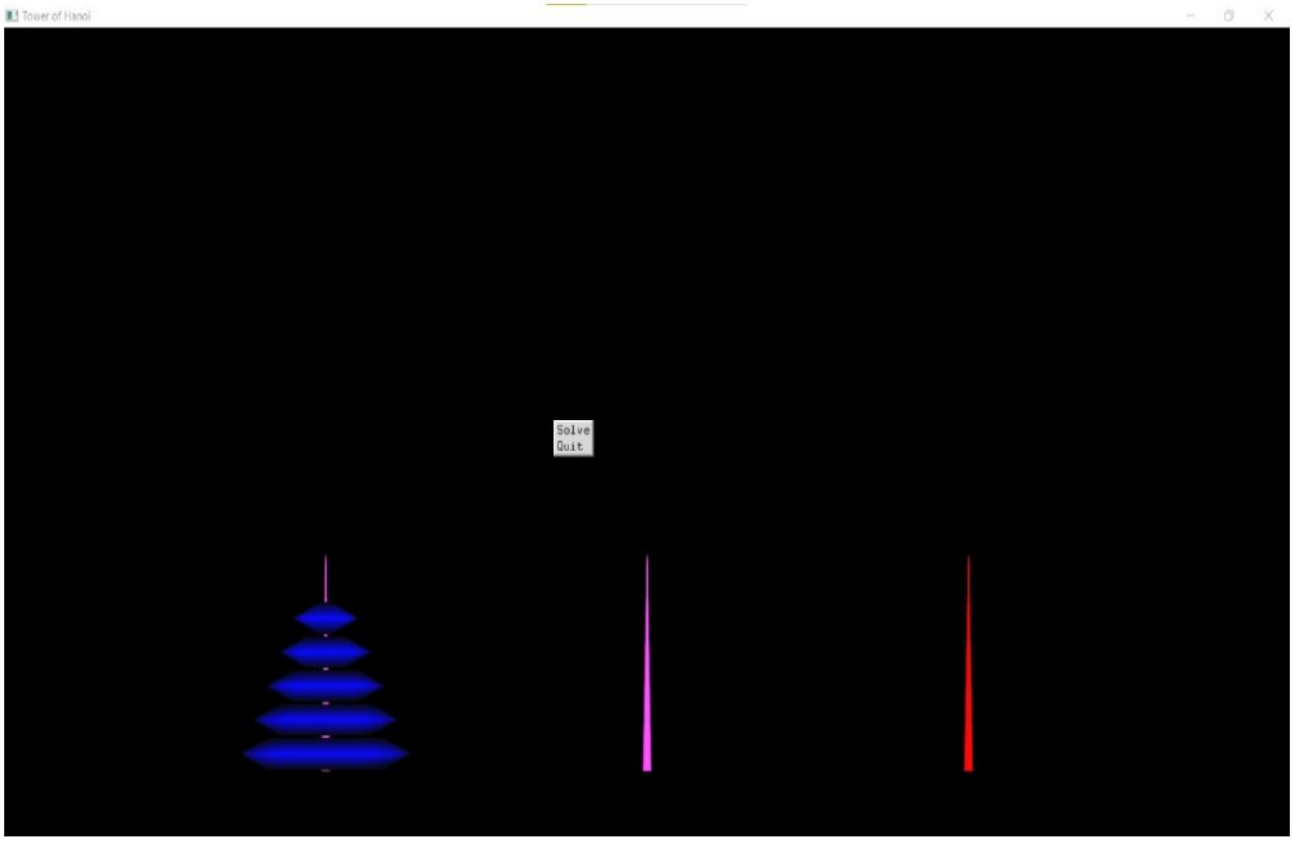

Figure 7.3: Initial Main Screen Post Disk Creation Based on User Input

Figure 7.4: Second Screen Post-Resolution

Once the Tower of Hanoi problem is resolved, this screen shows the final state with all disks moved to the destination peg. The screen captures the successful completion of the puzzle, highlighting the effectiveness of the implemented algorithm. It also demonstrates the graphical capabilities of the program, including smooth transitions and accurate rendering of the final positions of the disks, reinforcing the correctness of the solution .

Figure 7.5: Final 3D View Display

The final display provides a 3D view of the Tower of Hanoi setup, enhancing the visual appeal and providing a more immersive experience. This view includes dynamic lighting and shading effects, which add depth and realism to the simulation. The 3D perspective allows users to better appreciate the spatial arrangement of the pegs and disks, and the graphical sophistication of the application, showcasing the integration of computer graphics principles in solving and visualizing the Tower of Hanoi problem .

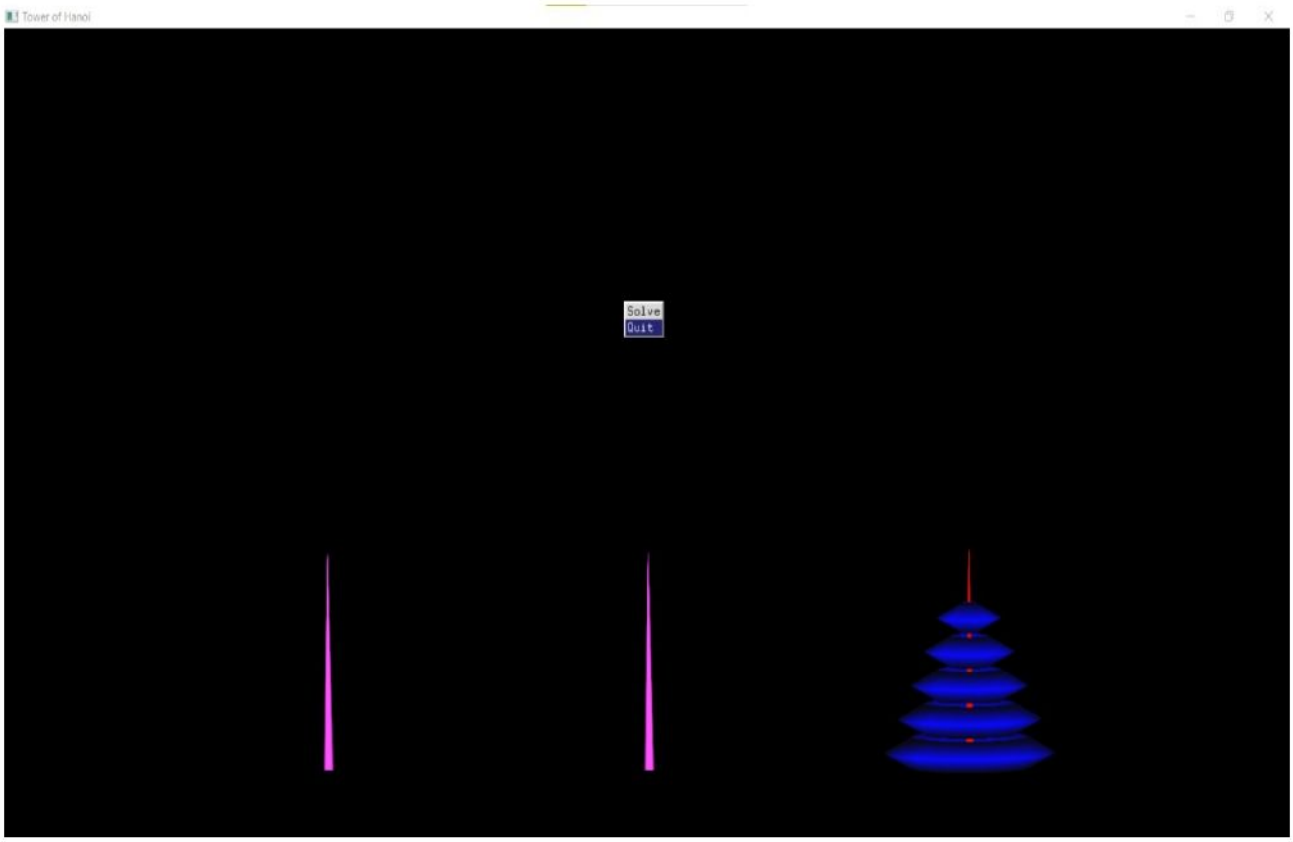

Figure 7.4: Second Screen Post-Resolution

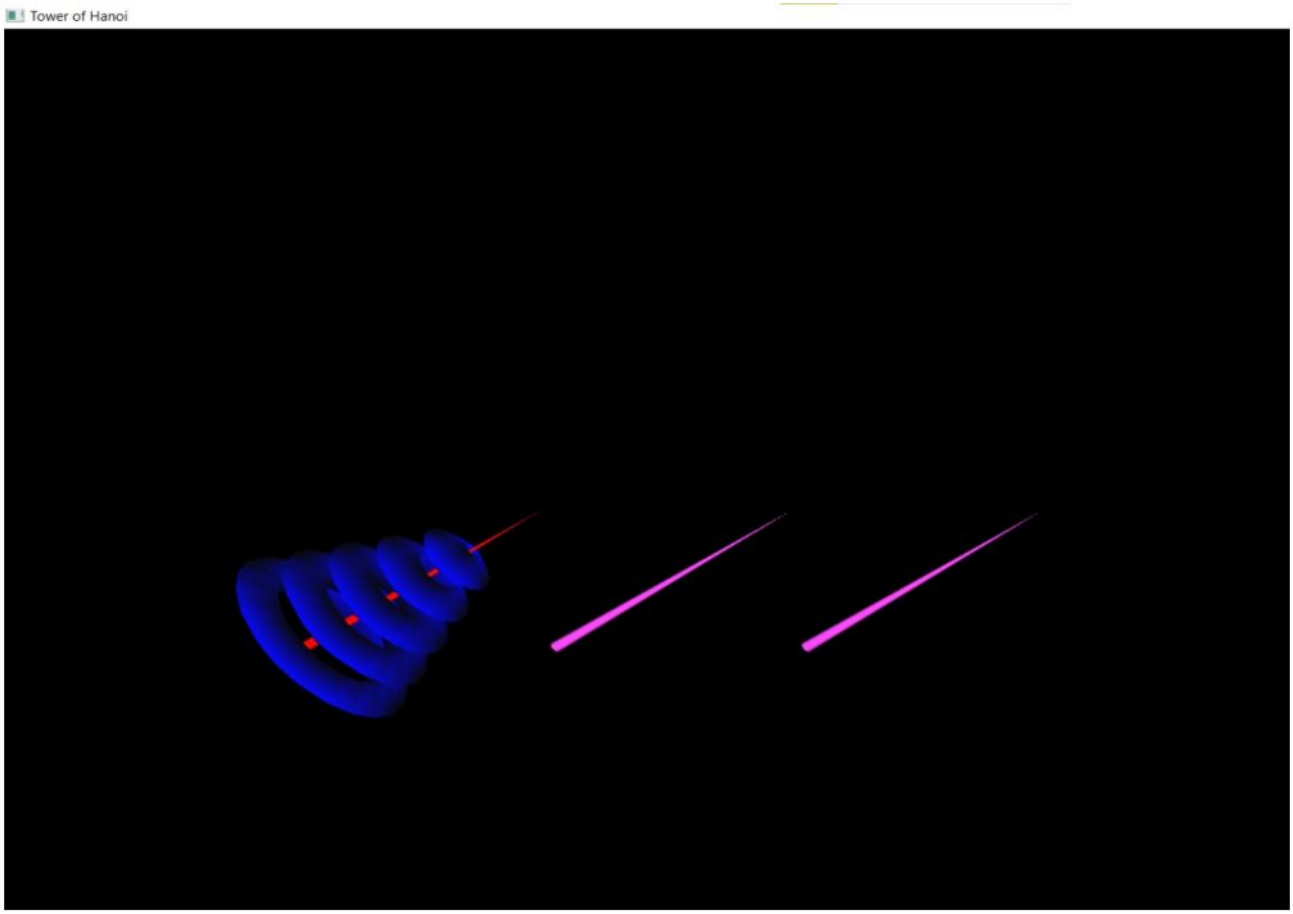

Figure 7.5: Final 3D View Display

Chapter 8

Conclusion & Future Enhancements

8.1 Conclusion

This project successfully implements the Tower of Hanoi problem using OpenGL for visualization, featuring smooth shading and dynamic lighting to enhance visual appeal. The recursive algorithm efficiently solves the puzzle by calculating and executing the minimum number of disk moves required. The graphical user interface allows for intuitive interaction through mouse controls and menu options, making it easy to start, pause, and quit the animation. The use of display lists for disks and cones ensures optimized rendering, contributing to the overall performance and responsiveness of the application. This project demonstrates a practical application of computer graphics principles and provides a valuable learning tool for understanding recursion and problem-solving in computer science.

8.2 Future Enhancements

Future enhancements for the Tower of Hanoi project can significantly improve both functionality and user experience. One major enhancement could be implementing texture mapping and shadows to add more detail and realism to the disks and pegs. and Enhanced interactivity is another crucial improvement. Adding features such as drag-and-drop functionality for the disks, different viewpoints, and camera angles can make the simulation more interactive and educational. Performance optimization is also important to ensure smooth performance, even with a higher number of disks or more complex graphical features. Lastly, integrating educational features such as step-by-step guides, explanations of the recursive algorithm, and visualizations of recursive calls can provide a deeper understanding of the problem and the solution.

References

[1] Edward Angel, *"Interactive Computer Graphics A Top-Down Approach With OpenGL"* 5th Edition, Addison-Wesley, 2008.

[2] F.S. Hill,*"Computer Graphics Using OpenGL"*, 2nd Edition, Pearson Education, 2001.

[3] James D.Foley, Andries Van Dam, Steven K. Feiner, John F Hughes, *"Computer Graphics"*, Second Edition, Addison-Wesley Professional, August 14,1995.

[4] @online OpenGL Official ,`https://www.opengl.org/`

[5] @online OpenGL Overview , `https://https://www.khronos.org/opengl/`