VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum: 590018

A Mini Project Report (18CSL67)
on

TOWER OF HANOI

A mini project report submitted in partial fulfillment of the requirement for the award of the degree of

Bachelor of Engineering
in
Computer Science & Engineering

Submitted by
ANUBHAV TEKRIWAL (1AY20CS018)

Under the guidance of
Mrs VARALAKSHMI B D
Department of Computer Science & Engineering

ACHARYA

Acharya Institute of Technology
Department of Computer Science & Engineering
Soladevanahalli, Bangalore-560107

ACHARYA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belgaum)
Soladevanahalli, Bangalore – 560 107

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Certificate

Certified that the Computer Graphics Mini Project entitled **"Tower Of Hanoi"** is a bonafide work carried out by **Anubhav Tekriwal (1AY20CS018)** in partial fulfillment for the award of degree of **Bachelor of Engineering in Computer Science & Engineering of the Visvesvaraya Technological University**, Belgaum during the academic year **2022-2023.** It is certified that all corrections/ suggestions indicated for internal assessments have been incorporated in the Report deposited in the departmental library. The Mini project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the **Bachelor of Engineering Degree**.

Signature of Guides **Signature of H.O.D**

Name of the Examiners **Signature with date**

1.

2.

ACKNOWLEDGEMENT

I express my gratitude to my institution and management for providing us with good infrastructure, laboratory facilities and inspiring staff, and whose gratitude was of immense help in completion of this mini project successfully.

I express my sincere gratitude to our principal, **Dr. M M Rajath Hegde** and vice principal, **Prof. Marigowda C K** for providing us the required environment and for their valuable suggestions.

My sincere thanks to **Dr. Ajith Padyana,** Professor and Head of the Department, Computer Science and Engineering, Acharya Institute of Technology for his valuable support and for rendering me the resources for this mini project.

I heartily thank **Prof. Varalakshmi B D, Prof. Kamala K, Prof. Jawahar Jonathan** and **Prof. Reshma**, Assistant Professors, Department of Computer Science and Engineering, Acharya Institute of Technology who guided me with valuable suggestions in completing this mini project at every stage.

My gratitude thanks should be rendered to many people who helped me in all possible ways.

ANUBHAV TEKRIWAL
(1AY20CS018)

ABSTRACT

A Computer Graphics project based on the concept of the classic puzzle game “Tower Of Hanoi”. This project focuses on creating a visually appealing and interactive representation of the Tower of Hanoi problem. This Tower of Hanoi consists of three pegs and three disks of different sizes. The API glutSolidCone() has been used to render the pegs as cone shapes, and glutSolidTorus() is used for the disks. An animation has been implemented which shows the transfer of the disks between the pegs. Lighting has been implemented by the inbuilt OpenGL lighting functions. This project implements the orthographic view. The glutCreateMenu() API has been used to provide a menu to interact with various features, such as changing the background color, toggle lighting, animation, etc. An on-screen legend has been provided to help users explore all the features.

The following concepts are implemented:

1. Translation
2. Scaling
3. Rotation
4. Reflection
5. Shearing
6. Lighting
7. Animation
8. Mouse and Keyboard Interactions

CONTENTS

List of Figures

Chapter 1

INTRODUCTION

1.1 Computer Graphics

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. It has the added advantage that, with the computer, we 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.

Computer graphics started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computers themselves. It has grown to include the creation, storage, and manipulation of models and images of objects. These models come from a diverse and expanding set of fields, and include physical, mathematical, engineering, architectural, and even conceptual structures, natural phenomena, and so on.

Computer graphics today is largely interactive. The user controls the contents, structure, and appearance of the objects and of their displayed images by using input devices, such as keyboard, mouse, or touch-screen.

Due to close relationships between the input devices and the display, the handling of such devices is included in the study of computer graphics. The advantages of the Interactive graphics are many in number. Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D patter-recognition abilities allow us to perceive and process data rapidly and efficiently. In many design, implementation, and construction processes today, the information pictures can give is virtually indispensable. Scientific visualization became an important field in the 1980s when the scientists and engineers realized that they could not interpret the prodigious quantities of data produced in supercomputer runs without summarizing the data and highlighting trends and phenomena in various kinds of graphical representations.

1.2 OpenGL

OpenGL Interface:

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects.

Most of our applications will be designed to access OpenGL directly through functions in three libraries. They are:

1. **Main GL**: Library has names that begin with the letter "gl" and are stored in a library usually referred to as GL.
2. **OpenGL Utility Library (GLU)**: This library uses only GL functions but contains code for creating common objects and simplifying viewing.
3. **OpenGL Utility Toolkit (GLUT)**: This provides the minimum functionality that should be accepted in any modern windowing system.

OpenGL Overview:

- OpenGL (Open Graphics Library) is the interface between a graphic program and graphics hardware. It is streamlined. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.
- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- It is system-independent: It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- It is a state machine. At any moment during the execution of a program there is a current model transformation.
- It is a rendering pipeline. The rendering pipeline consists of the following steps:
    - Defines objects mathematically.
    - Arranges objects in space relative to a viewpoint.
    - Calculates the color of the objects.
    - Rasterizes the objects.

Chapter 2

SYSTEM REQUIREMENTS SPECIFICATION

2.1 SOFTWARE REQUIREMENTS

- Programming language – C/C++ using OpenGL
- Operating system – Linux operating system
- Compiler – C Compiler
- Graphics library – GL/glut.h
- OpenGL 2.0

2.2 HARDWARE REQUIREMENTS

- Dual Core Processor
- 2GB RAM
- 40GB Hard disk
- Mouse and other pointing devices
- Keyboard

2.3 FUNCTIONAL REQUIREMENTS

OpenGL APIs:

If we want to have a control on the flow of program and if we want to interact with the window system then we use OpenGL API'S. Vertices are represented in the same manner internally, whether they are specified as two-dimensional or three- dimensional entities, everything that we do are here will be equally valid in three dimensions. Although OpenGL is easy to learn, compared with other APIs, it is nevertheless powerful. It supports the simple three dimensional programs and also supports the advanced rendering techniques.

GL/glut.h:

We use a readily available library called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system. The application program uses only GLUT functions and can be recompiled with the GLUT library for other window system. OpenGL makes a heavy use of macros to increase code readability and avoid the use of magic numbers. In most implementation, one of the include lines.

Chapter 3

ABOUT THE PROJECT

3.1 INTRODUCTION

A Computer Graphics project based on the concept of the classic puzzle game “Tower Of Hanoi”. This project focuses on creating a visually appealing and interactive representation of the Tower of Hanoi problem. This Tower of Hanoi consists of three pegs and three disks of different sizes. The API glutSolidCone() has been used to render the pegs as cone shapes, and glutSolidTorus() is used for the disks. An animation has been implemented which shows the transfer of the disks between the pegs. Lighting has been implemented by the inbuilt OpenGL lighting functions.

This project implements the orthographic view. The glutCreateMenu() API has been used to provide a menu to interact with various features, such as changing the background color, toggle lighting, animation, etc. An on-screen legend has been provided to help users explore all the features.

3.2 OBJECTIVES:

The aim of this project is to develop a graphics package which supports basic operations which include building a simulation of the “Tower of Hanoi” problem using Open GL. The package must also has a user-friendly interface. The objective of developing this model was to design and apply the skills we learnt in class.

3.3 BUILT-IN FUNCTIONS:

- **glColor3f (float, float, float)** : This function will set the current drawing color.
- **gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)** : which defines a two dimensional viewing rectangle in the plane z=0.
- **glClear()** : Takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.
- **glClearColor() :** Specifies the red, green, blue, and alpha values used by glClear to clear the color buffers.

- **glLoadIdentity() :** the current matrix with the identity matrix.
- **glMatrixMode(mode) :** Sets the current matrix mode, mode can be GL_MODELVIEW, GL_PROJECTION or GL_TEXTURE.
- **void glutInit (int *argc, char**argv) :** Initializes GLUT, the arguments from main are passed in and can be used by the application.
- **void glutInitDisplayMode (unsigned int mode) :** Requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model and buffering.
- **void glutInitWindowSize (int width, int height) :** Specifies the initial position of the topleft corner of the window in pixels.
- **int glutCreateWindow (char *title) :** A window on the display. The string title can be used to label the window. The return value provides references to the window that can be used when there are multiple windows.
- **void glutMouseFunc(void *f(int button, int state, int x, int y) :** Register the mouse callback function f. The callback function returns the button, the state of button after the event and the position of the mouse relative to the top-left corner of the window.
- **void glutKeyboardFunc(void(*func) (void)) :** This function is called every time when you press enter key to resume the game or when you press ‘b’ or ‘B’ key to go back to the initial screen or when you press esc key to exit from the application.
- **void glutDisplayFunc (void (*func) (void)) :** Register the display function func that is executed when the window needs to be redrawn.
- **void glutSpecialFunc(void(*func)(void)) :** This function is called when you press the special keys in the keyboard like arrow keys, function keys etc. In our program, the func is invoked when the up arrow or down arrow key is pressed for selecting the options in the main menu and when the left or right arrow key is pressed for moving the object(car) accordingly.
- **glutPostReDisplay ( ) :** which requests that the display callback be executed after the current callback returns.
- **void MouseFunc (void (*func) void)) :** This function is invoked when mouse keys are pressed. This function is used as an alternative to the previous function i.e., it is used to

move the object(car) to right or left in our program by clicking left and right button respectively.

- **void glutMainLoop () :** Cause the program to enter an event-processing loop. It should be the last statement in main function.

3.4 USER DEFINED FUNCTIONS:

- **void push(int p, int disk) :** This function is responsible for pushing a disk onto a pole. It takes the parameters int disk (representing the disk number) and int pole (representing the pole number). It updates the pos array to reflect the new disk position on the specified pole.
- **void pop(int p, int disk) :** This function removes the top disk from a pole. It takes the parameter int pole (representing the pole number). It updates the pos array to remove the top disk from the specified pole.
- **void tower(int n,int src,int temp,int dst) :** This recursive function implements the Tower of Hanoi algorithm. It takes four parameters: int n (the number of disks), int src (the source pole), int temp (the temporary pole), and int dst (the destination pole). It uses recursion to move the disks from the source pole to the destination pole, following the rules of the Tower of Hanoi puzzle. It also records the moves in the moves array.
- **void drawPegs() :** This function is responsible for drawing the pegs of the Tower of Hanoi. It uses OpenGL commands to draw the pegs on the screen. It does not take any parameters.
- **void printString(char *text) :** This function is used to print a string on the screen using the GLUT bitmap fonts. It takes the parameters char* string (the string to be printed) and int x and int y (the coordinates of the position where the string should be printed).
- **void drawText() :** This function is responsible for drawing the text and labels on the screen. It uses the GLUT bitmap fonts to print the required text at specified positions. It is called by the display() function to draw the text elements of the Tower of Hanoi puzzle.
- **void smallLargeEffect() :** This function creates a small and large effect by scaling the objects on the screen. It is used to provide a visual effect when the user interacts with the puzzle. This function is called when a key is pressed or the mouse is clicked.
- **void drawSolved() :** This function draws the final configuration of the Tower of Hanoi

puzzle when it is solved. It is called by the display() function to display the solved state of the puzzle. It draws the disks on the destination pole in their final order.

- **void animate(int n, int src, int dest) :** This function animates the movement of a disk from the source pole to the destination pole. It takes three parameters: n (the disk number), src (the source pole), and dest (the destination pole). It updates the position of the disk in the pos array and triggers the animation of the disk movement.
- **void spinScene() :** This function rotates the entire scene in a circular motion. It is responsible for creating the spinning effect of the Tower of Hanoi puzzle. It is called by the display() function to update the rotation angle and redraw the scene.
- **void restart() :** This function resets the Tower of Hanoi puzzle to its initial state. It is called when the user chooses to restart the puzzle. It resets the positions of the disks and clears the moves array to start the puzzle from the beginning.

3.5 DATA FLOW DIAGRAM

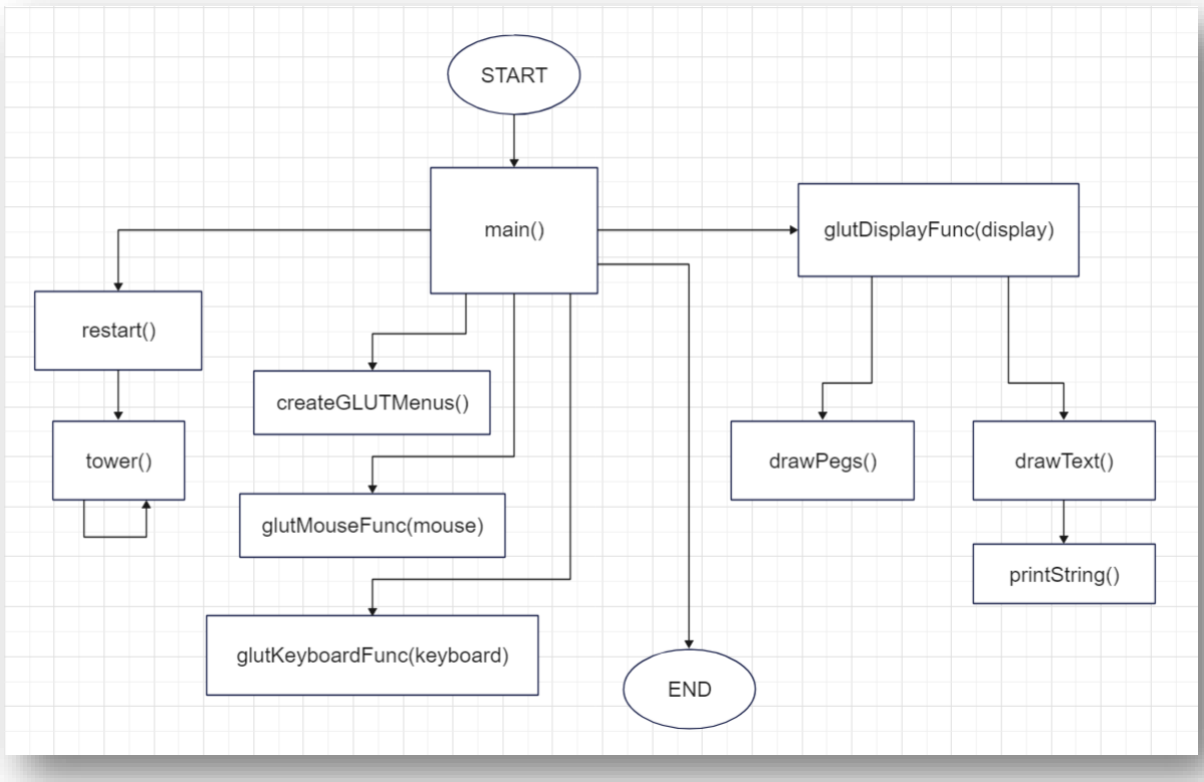

Fig 3.1: Data Flow Diagram

3.6 SOURCE CODE:

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

#define LIGHT_ON 0
#define LIGHT_OFF 1
#define NUM_DISKS 3

int pos[16] = {10,15,20,25,30,35,40,45,50};
int peg[3] = {50,150,250};
int moves[10000][3];
int max_moves;
int POLES[3][10];
int top[3]={-1,-1,-1};

static GLfloat theta=0.0;
static GLfloat axis=2;
float scale=1.00f;
int reflect=0;
float tx=0, ty=0, a=150;
int shCounter=0;
int se1 = 0, se2;
int baseXcoord=250, baseYcoord=95, dif=4;
int legend=1;
// int delayTime = 100;

int cnt,counter,speed=7;
float ycoordinate, xcoordinate, zcoordinate;
int lightflag=1, animationFlag=1;
```

```
GLfloat shearMatrix[16] = {
   1.0f, 0.0f, 0.0f, 0.0f,
   0.25f, 1.0f, 0.0f, 0.0f,
   0.25f, 0.0f, 1.0f, 0.0f,
   0.0f, 0.0f, 0.0f, 1.0f
};
GLfloat inverseShearMatrix[16] = {
   1.0f, 0.0f, 0.0f, 0.0f,
   -0.25f, 1.0f, 0.0f, 0.0f,
   -0.25f, 0.0f, 1.0f, 0.0f,
   0.0f, 0.0f, 0.0f, 1.0f
};

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;

        for(i=0;i<3;i++)
        {
                glColor3f(0.25,0.41, 0.88);
                glPushMatrix();
                glTranslatef(peg[i],5,0);
                glRotatef(-90,1,0,0);
                glutSolidCone(3,50,20,20);

                //for drawing the circular ring at the bottom of each peg.
                glColor3f(0,0,0.5);
                glutSolidTorus(2,45, 20, 20);

                glPopMatrix();
        }


}

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

void drawText()
{
        glColor3f(0, 0.79, 1);
```

```
glBegin(GL_POLYGON); // Yellow BOX
        glVertex3f(-73.8,93.3,0);
        glVertex3f(-73.8,89.8,0);
        glVertex3f(10.8,89.8,0);
        glVertex3f(10.8,93.3,0);
glEnd();

glColor3f(1.0,1.0,0.0);
glBegin(GL_POLYGON); // Yellow BOX
        glVertex3f(-73,93,0);
        glVertex3f(-73,90,0);
        glVertex3f(10,90,0);
        glVertex3f(10,93,0);
glEnd();
glColor3f(0,0,0);
glRasterPos3f(-45,90.5,2);
printString("Move : ");                    // Move :
glColor3f(1,0,1);
char str[5];
sprintf(str, "%d", counter);
glRasterPos3f(-17,90.5,2);
printString(str);                          // $counter
glRasterPos3f(-70,80,0);
printString("NEXT: ");                     // NEXT:
glColor3f(0,0,0);
glRasterPos3f(-44,80,0);
printString("Disk");                  // DISK

glColor3f(0,0.5,0);
char str1[10];
sprintf(str1, "%d", moves[counter][0]); // disk no. = moves[counter][0]
glRasterPos3f(-25,80,0);
printString(str1);                         // disk no.
```

```
glColor3f(0,0,0);
glRasterPos3f(-15,80,0);
printString("from");                    // from
glColor3f(0,0,0.5);
char src[2];
if(moves[counter][1]==0)strcpy(src,"A");          // src = A or B or C
else if(moves[counter][1]==1)strcpy(src,"B");
else strcpy(src,"C");
glRasterPos3f(5,80,0);
printString(src);                          // src
glColor3f(0,0,0);
glRasterPos3f(15,80,0);
printString("to");                        // to
glColor3f(0,0,0.5);
char dst[2];
if(moves[counter][2]==0)strcpy(dst,"A");          // dst = A or B or C
else if(moves[counter][2]==1)strcpy(dst,"B");
else strcpy(dst,"C");
glRasterPos3f(25,80,0);
printString(dst);                          // dst

// Drawing A, B and C on top of each peg
glColor3f(1,0,0);
glRasterPos3f(peg[0],1,0);
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'A');
glRasterPos3f(peg[1],1,0);
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'B');
glRasterPos3f(peg[2],1,0);
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'C');


glColor3f(0.502, 0.502, 0.502);
glBegin(GL_POLYGON); // Yellow BOX
        glVertex3f(-73,73.5,0);
```

```
                glVertex3f(-73,49,0);
                glVertex3f(-10,49,0);
                glVertex3f(-10,73.5,0);
        glEnd();
        glColor3f(1,1,1);
        glRasterPos3f(-70,70.5,0);
        printString("Animation : ");
        if(animationFlag) {
                glColor3f(0,1,0);
                glRasterPos3f(-30,70.5,0);
                printString("ON");}
        else {
                glColor3f(1,0,0);
                glRasterPos3f(-30,70.5,0);
                printString("OFF");}

        glColor3f(1,1,1);
        glRasterPos3f(-70,66.5,0);
        printString("Lights : ");
        if(lightflag) {
                glColor3f(0,1,0);
                glRasterPos3f(-30,66.5,0);
                printString("ON");}
        else {
                glColor3f(1,0,0);
                glRasterPos3f(-30,66.5,0);
                printString("OFF");}

        glColor3f(1,1,1);
        glRasterPos3f(-70,66.5-4,0);
        printString("Scale: ");
        if (scale > 0.99f && scale < 1.01f) glColor3f(0,1,0); else glColor3f(1,0,0);          //if
then green else red
        glRasterPos3f(-30,66.5-4,0);
```

```
char printScale[5];
sprintf(printScale,"%.2f",scale);
printString(printScale);

glColor3f(1,1,1);
glRasterPos3f(-70,66.5-8,0);
printString("Tx: ");
if (tx==0) glColor3f(1,0,0); else glColor3f(0,1,0);              //if then green else red
glRasterPos3f(-30,66.5-8,0);
char printTx[5];
sprintf(printTx,"%.0f",tx);
printString(printTx);

glColor3f(1,1,1);
glRasterPos3f(-70,66.5-12,0);
printString("Ty: ");
if (ty==0) glColor3f(1,0,0); else glColor3f(0,1,0);
glRasterPos3f(-30,66.5-12,0);
char printTy[5];
sprintf(printTy,"%.0f",ty);
printString(printTy);

glColor3f(1,1,1);
glRasterPos3f(-70,66.5-16,0);
printString("Speed: ");
if (((speed/7)+1) != 1) glColor3f(0,1,0); else glColor3f(1,0,0);
glRasterPos3f(-30,66.5-16,0);
char printSpeed[5];
sprintf(printSpeed,"%d",(speed/7)+1);
printString(printSpeed);
printString("x");


glColor3f(0, 0, 0);
```

```
        glLineWidth(4);
        glBegin(GL_LINE_LOOP); // Yellow BOX
                glVertex3f(121,-7.7,0);
                glVertex3f(148,-7.7,0);
        glEnd();

        glColor3f(0.878, 0.066, 0.372);
        glRasterPos3f(100,-7,0);
        printString("Press SPACE to start solving.");

if(legend){
        glColor3f(0, 0.79, 1);
        glBegin(GL_POLYGON); // blue BOX
                glVertex3f(baseXcoord-50,baseYcoord+4,0);
                glVertex3f(baseXcoord-50,baseYcoord,0);
                glVertex3f(baseXcoord+40,baseYcoord,0);
                glVertex3f(baseXcoord+40,baseYcoord+4,0);
        glEnd();
        glColor3f(0.1, 0.2, 0.2);
        glBegin(GL_POLYGON); // dark green BOX
                glVertex3f(baseXcoord-103,baseYcoord,0);
                glVertex3f(baseXcoord-103,baseYcoord-26.8,0);
                glVertex3f(baseXcoord+96,baseYcoord-26.8,0);
                glVertex3f(baseXcoord+96,baseYcoord,0);
        glEnd();

        glColor3f(0, 0, 0);
        glRasterPos3f(baseXcoord-25.5,baseYcoord+1,0);
        printString("CONTROLS");

        glColor3f(1,1,1);
        glRasterPos3f(baseXcoord-100,baseYcoord-dif,0);
        printString("Translation: ");
        glColor3f(0, 0.79, 1);
```

```
glRasterPos3f(baseXcoord-58,baseYcoord-dif,0);
printString("WASD");

glColor3f(1,1,1);
glRasterPos3f(baseXcoord-100,baseYcoord-2*dif,0);
printString("Scaling: ");
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord-58,baseYcoord-2*dif,0);
printString("QE");

glColor3f(1,1,1);
glRasterPos3f(baseXcoord-100,baseYcoord-3*dif,0);
printString("Rotation: ");
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord-58,baseYcoord-3*dif,0);
printString("C/Middle Mouse");

glColor3f(1,1,1);
glRasterPos3f(baseXcoord-100,baseYcoord-4*dif,0);
printString("Reflection: ");
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord-58,baseYcoord-4*dif,0);
printString("R");

glColor3f(1,1,1);
glRasterPos3f(baseXcoord-100,baseYcoord-5*dif,0);
printString("Shearing: ");
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord-58,baseYcoord-5*dif,0);
printString("ZX");

glColor3f(1,1,1);
glRasterPos3f(baseXcoord-100,baseYcoord-6*dif,0);
printString("Breathing Effect: ");
```

```
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord-39.5,baseYcoord-6*dif,0);
printString("F");



//COLUMN 2
glColor3f(1,1,1);
glRasterPos3f(baseXcoord+5,baseYcoord-dif,0);
printString("Perspective: ");
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord+58,baseYcoord-dif,0);
printString("IJKLUO");

glColor3f(1,1,1);
glRasterPos3f(baseXcoord+5,baseYcoord-2*dif,0);
printString("Animation Speed: ");
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord+79.5,baseYcoord-2*dif,0);
printString("+-");

glColor3f(1,1,1);
glRasterPos3f(baseXcoord+5,baseYcoord-3*dif,0);
printString("Random Background: ");
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord+85,baseYcoord-3*dif,0);
printString("B");

        glColor3f(1,1,1);
glRasterPos3f(baseXcoord+5,baseYcoord-4*dif,0);
printString("Next Move: ");
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord+49,baseYcoord-4*dif,0);
printString("Scroll Down");
```

```
        glColor3f(1,1,1);
glRasterPos3f(baseXcoord+5,baseYcoord-5*dif,0);
printString("Undo Move: ");
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord+59,baseYcoord-5*dif,0);
printString("Scroll Up");

            glColor3f(1,1,1);
glRasterPos3f(baseXcoord+5,baseYcoord-6*dif,0);
printString("Disable Legend: ");
glColor3f(0, 0.79, 1);
glRasterPos3f(baseXcoord+85,baseYcoord-6*dif,0);
printString("\\");
}
else{

glColor3f(0,0,0);
glBegin(GL_POLYGON); // dark green BOX
        glVertex3f(baseXcoord+10.4,baseYcoord+0.1,0);
        glVertex3f(baseXcoord+10.4,baseYcoord-9.4,0);
        glVertex3f(baseXcoord+96.1,baseYcoord-9.4,0);
        glVertex3f(baseXcoord+96.1,baseYcoord+0.1,0);
glEnd();

glColor3f(0.80*1.0, 1.0, 1);
glBegin(GL_POLYGON); // dark green BOX
        glVertex3f(baseXcoord+11,baseYcoord,0);
        glVertex3f(baseXcoord+11,baseYcoord-9,0);
        glVertex3f(baseXcoord+96,baseYcoord-9,0);
        glVertex3f(baseXcoord+96,baseYcoord,0);
glEnd();

glColor3f(0, 0.55, 1);
glRasterPos3f(baseXcoord+15,baseYcoord-4,0);
```

```
        printString("ANUBHAV TEKRIWAL");

        glColor3f(0.133, 0.545, 0.133);
        glRasterPos3f(baseXcoord+30,baseYcoord-7,0);
        printString("1AY20CS018");
        }
}

void smallLargeEffect()
{

        if(se2==2) {scale+=0.03; se2=0;glutIdleFunc(NULL);}

        se1++;
        if(se1<4) scale-=0.03;
        else if(se1<14) scale+=0.03;
        else if(se1<21) scale-=0.03;
        else {se1=0;se2++;}

        for(int m=0; m<10000;m++)
                for(int n=0; n<3500;n++);

        glutPostRedisplay();
}

void drawSolved()
{
                glColor3f(0.4, 0.4, 0.4);
        glBegin(GL_POLYGON); //yellow box
                glVertex3f(99.8,75.1,0);
                glVertex3f(99.8,67.4,0);
                glVertex3f(200.3,67.4,0);
                glVertex3f(200.3,75.1,0);
        glEnd();
```

```
glColor3f(0.80*1.0, 1.0, 1);
glBegin(GL_POLYGON); //yellow box
        glVertex3f(100,75,0);
        glVertex3f(100,68,0);
        glVertex3f(200,68,0);
        glVertex3f(200,75,0);
glEnd();

glColor3f(0, 0.55, 1);
glRasterPos3f(113,72,0);
printString("ANUBHAV TEKRIWAL");

glColor3f(0.133, 0.545, 0.133);
glRasterPos3f(127,69,0);
printString("1AY20CS018");

glColor3f(0.35, 0.35, 0.35);
// glColor3f(1.0f, 0.843f, 0.0f);
glBegin(GL_POLYGON);
        glVertex3f(-73,94,0);
        glVertex3f(-73,89,0);
        glVertex3f(15,89,0);
        glVertex3f(15,94,0);
glEnd();
glColor3f(0.678, 0.847, 0.902);
glRasterPos3f(-65,90.5,2);
printString("TOTAL MOVES :");
glColor3f(0.9,0.9,0.9);
char str[5];
sprintf(str, "%d", counter);
glRasterPos3f(-1,90.5,2);
printString(str);
```

```
        glColor3f(1.0,1.0,0.0);
        glBegin(GL_POLYGON);
                glVertex3f(50,85,-5);
                glVertex3f(50,80,-5);
                glVertex3f(250,80,-5);
                glVertex3f(250,85,-5);
        glEnd();
        glColor3f(1,0,1);
        glRasterPos3f(135,81.5,2);
        printString("Solved !!");

        smallLargeEffect();
}

void display()
{
        int i,j,k;

        if(shCounter>0) {shCounter=0; glMultMatrixf(shearMatrix);}
        else if(shCounter<0) {shCounter=0; glMultMatrixf(inverseShearMatrix);}

        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        if(lightflag)glEnable(GL_LIGHTING);
        glPushMatrix();

        glTranslatef(a,0,0);
        if(reflect) glScalef(-scale,scale,scale);
        else glScalef(scale,scale,scale);

        if(axis==1) glRotatef(theta,1,0,0);
        if(axis==2) glRotatef(theta,0,1,0);
        if(axis==3) glRotatef(theta,0,0,1);

        glTranslatef(tx,ty,0);
```

```
        gluLookAt(xcoordinate,ycoordinate,zcoordinate,0,0,-1,0,1,0);          //PERSPECITVE
VIEWING
        glTranslatef(-a,0,0);

        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(1.0,1.0,1.0,0);
        glColor3f(1,0,0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-80,350,-10,100,-500,500);
        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<70;i+=speed)
        {
                glPushMatrix();
                glTranslatef(peg[src],i,0);
```

```
            glRotatef(85,1,0,0);
            glColor3f(0.1*n+0.5,0.2*n+0.5,0+0.5);
            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,70,0);
                    glRotatef(85,1,0,0);
                    glColor3f(0.1*n+0.5,0.2*n+0.5,0+0.5);
                    glutSolidTorus(2.0, 4*n, 20, 20);
                    glPopMatrix();
                    glutSwapBuffers();
                    display();
            }
    else
            for(i=peg[src];i>=peg[dest];i-=speed)
            {
                    glPushMatrix();
                    glTranslatef(i,70,0);
                    glRotatef(85,1,0,0);
                    glColor3f(0.1*n+0.5,0.2*n+0.5,0+0.5);
                    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+0.5,0.2*n+0.5,0+0.5);
                glutSolidTorus(2.0, 4*n, 20, 20);
                glPopMatrix();
                glutSwapBuffers();
                display();
        }
}

void spinScene()
{
        theta+=4.0;
        if(theta>360.0) axis=1;
        if(theta>720.0) {axis =2;theta=0;glutIdleFunc(NULL);}
        glutPostRedisplay();
}

void mouse(int btn,int state,int x,int y)
{
        if(btn == 4 && state == GLUT_DOWN)
        {
                scale=1;
                reflect=0;
                tx=0;
                ty=0;
                xcoordinate=0;
                zcoordinate=3;
                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 && state == GLUT_DOWN)
        {
            scale=1;
            reflect=0;
            tx=0;
            ty=0;
            xcoordinate=0;
            zcoordinate=3;

            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]);
            }
        }
        if(btn == GLUT_MIDDLE_BUTTON && state==GLUT_DOWN){
            glutIdleFunc(spinScene);
        }


        glutPostRedisplay();
}

void restart()
{
        int i;
```

```
        memset(POLES,0,sizeof(POLES));
        memset(moves,0,sizeof(POLES));
        memset(top,-1,sizeof(top));
        theta=0.0;
        axis=2;
        scale=1.0;
        reflect=0;
        tx=0;
        ty=0;
        shCounter=0;
        se1=0;
        se2=0;


        cnt=0,counter=0;
        xcoordinate=0;
        ycoordinate=-0.3;
        zcoordinate=3;
        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)
{
        switch(option)
        {
                case 1: restart();
                                glutPostRedisplay();
                                break;

                case 10:        glClearColor(1,1,1,0);
                                        break;

                case 100:       exit(0);
                                        break;
        }
}
void processMenuCamera(int option)
{
        switch(option)
        {
                case 119:ycoordinate+=0.1;break;
                case 115:ycoordinate-=0.1;break;
                case 97:xcoordinate+=0.1;break;
                case 100:xcoordinate-=0.1;break;
                case 113:zcoordinate+=0.1;break;
                case 101:zcoordinate-=0.1;break;
        }
        glutPostRedisplay();
```

```
}

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

void processMenuSolveCompletely(int option)
{
	while(counter<max_moves)
	{
		mouse(4,GLUT_DOWN,0,0);
		display();
		for(int m=0; m<20000;m++)
			for(int n=0; n<10000;n++);
	}
}

void createGLUTMenus()
{
	int menu = glutCreateMenu(processMenuLighting);
	glutAddMenuEntry("On",LIGHT_ON);
	glutAddMenuEntry("Off",LIGHT_OFF);

	int menuAnimate = glutCreateMenu(processMenuAnimate);
	glutAddMenuEntry("On",0);
	glutAddMenuEntry("Off",1);
```

```
        glutCreateMenu(processMenuMain2);
        glutAddSubMenu("Lights",menu);
        glutAddSubMenu("Disk Animation",menuAnimate);
        glutAddMenuEntry("Restart",1);
        glutAddMenuEntry("Reset BG color",10);
        glutAddMenuEntry("Quit",100);
        glutAttachMenu(GLUT_RIGHT_BUTTON);

}

void keyboard(unsigned char c, int x, int y){
        switch(c)
        {
                case ' '://pressing SPACE means start solving till end
                        keyboard('\\',0,0); processMenuSolveCompletely(1); break;

                case 'I':
                case 'i'://Perspective viewing using kb
                        processMenuCamera(119);break;
                case 'K':
                case 'k':
                        processMenuCamera(115);break;
                case 'j':
                case 'J':
                        processMenuCamera(97);break;
                case 'L':
                case 'l':
                        processMenuCamera(100);break;
                case 'U':
                case 'u':
                        processMenuCamera(113);break;
                case 'O':
                case 'o':
```

```
                processMenuCamera(101);break;

        case 'B':
        case 'b'://random bg color on pressing 'b'

glClearColor((rand()%100)/100.0,(rand()%100)/100.0,(rand()%100)/100.0,0);
                break;

        case 'E':
        case 'e':
                scale+=0.05f;
                break;
        case 'Q':
        case 'q':
                scale-=0.05f;
                break;

        case 'A':
        case 'a':tx-=4;break;
        case 'D':
        case 'd':tx+=4;break;
        case 'S':
        case 's':ty-=2;break;
        case 'W':
        case 'w':ty+=2;break;

        case 'R':
        case 'r':reflect=(reflect+1)%2; break;

        case 'X':
        case 'x':shCounter=1;break;
        case 'Z':
        case 'z':shCounter=-1;break;
```

```
                case 'F':
                case 'f': glutIdleFunc(smallLargeEffect); break;
                case 'C':
                case 'c': mouse(GLUT_MIDDLE_BUTTON,GLUT_DOWN,0,0);break;

                case '+': if(speed<30) speed+=7; break;
                case '-': speed-=7; if(speed<=0) speed=1; break;

                case '\\': legend=(legend+1)%2; 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 - Anubhav");
        restart();
        init();
        glutDisplayFunc(display);
        createGLUTMenus();
        glutKeyboardFunc(keyboard);
        glutMouseFunc(mouse);
        glutMainLoop();
        return 0;
}
```

Chapter 4

RESULT

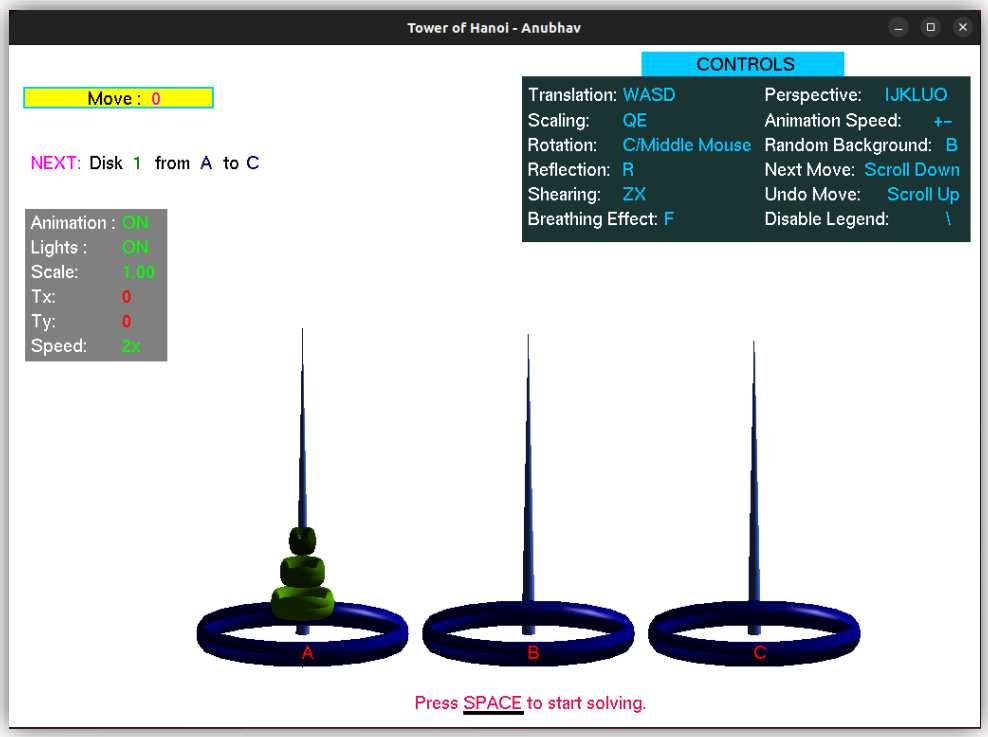

Fig 4.1: Final Output

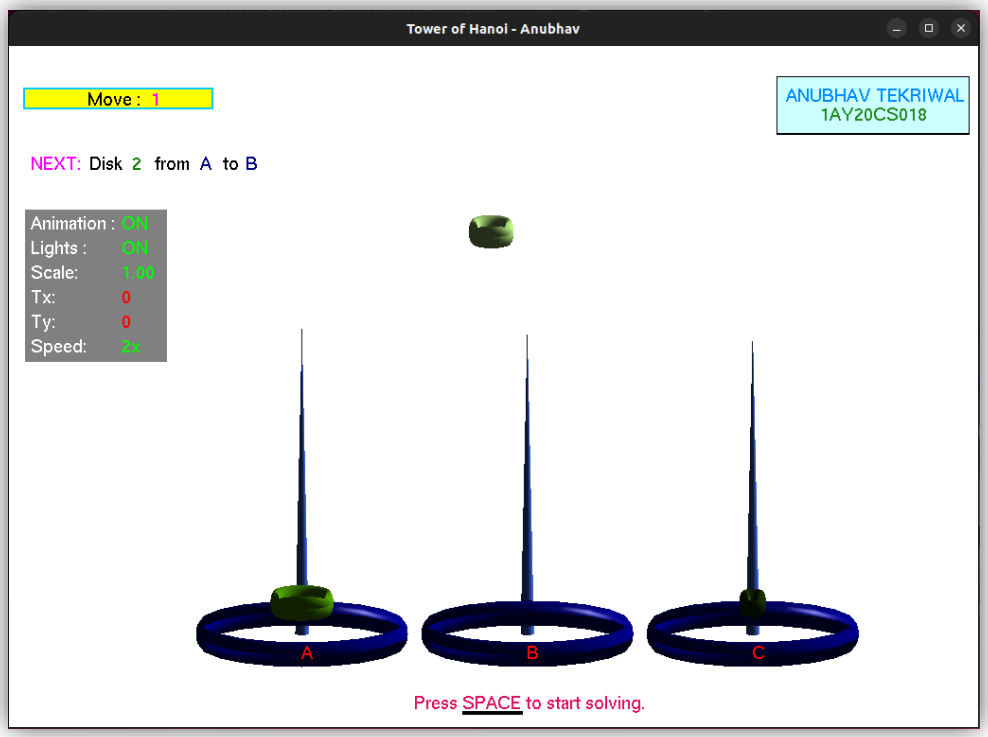

Fig 4.2: Animation of disks between pegs

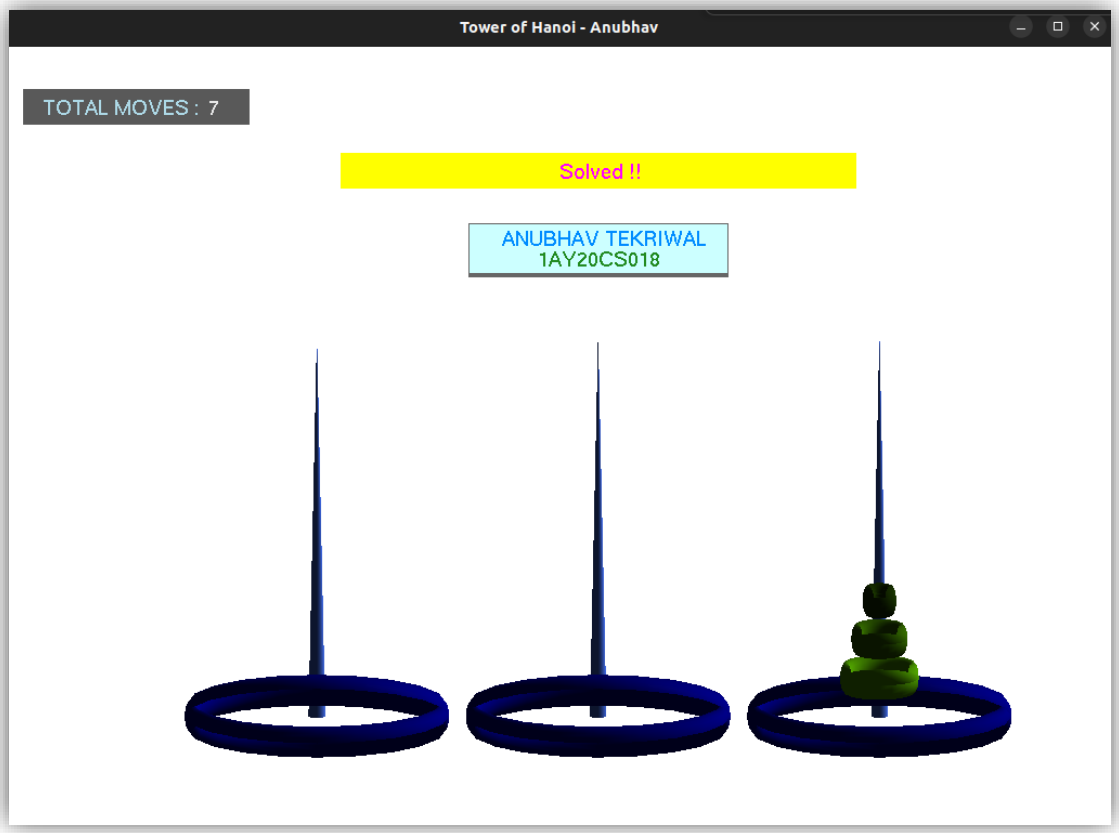

Fig 4.3: Finished Simulation

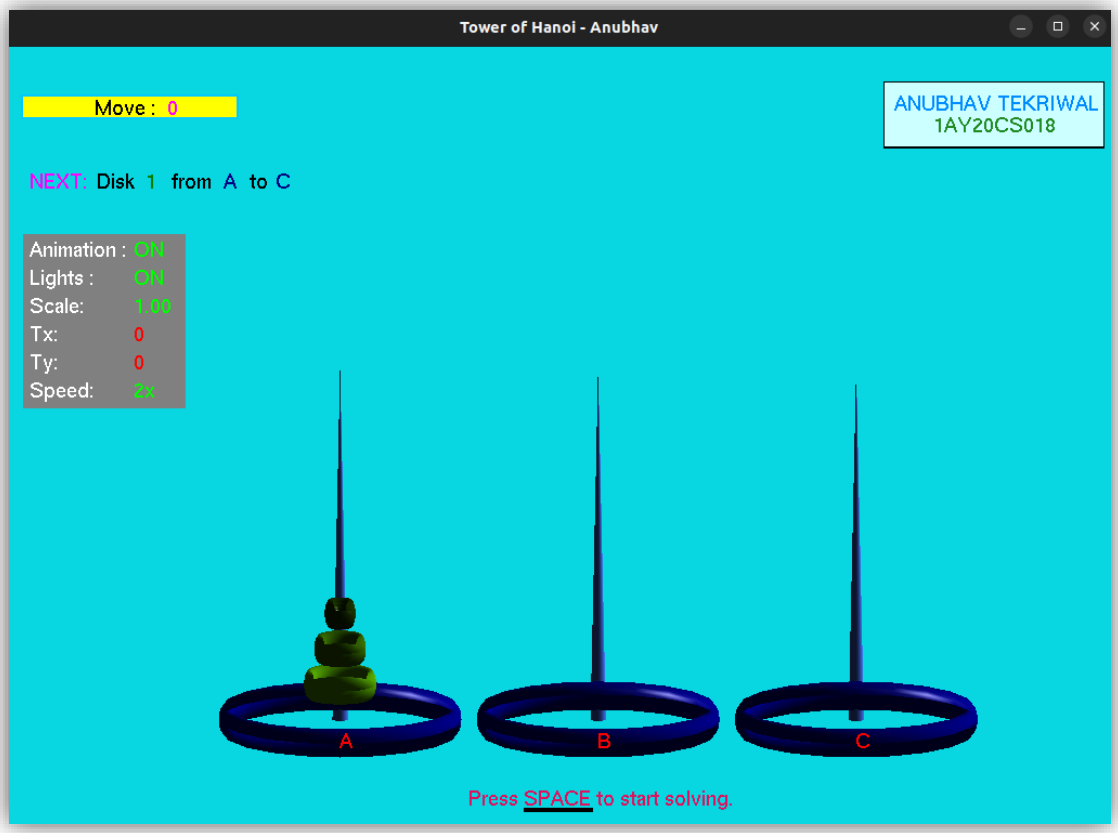

Fig 4.4: Changing Random Background Colors

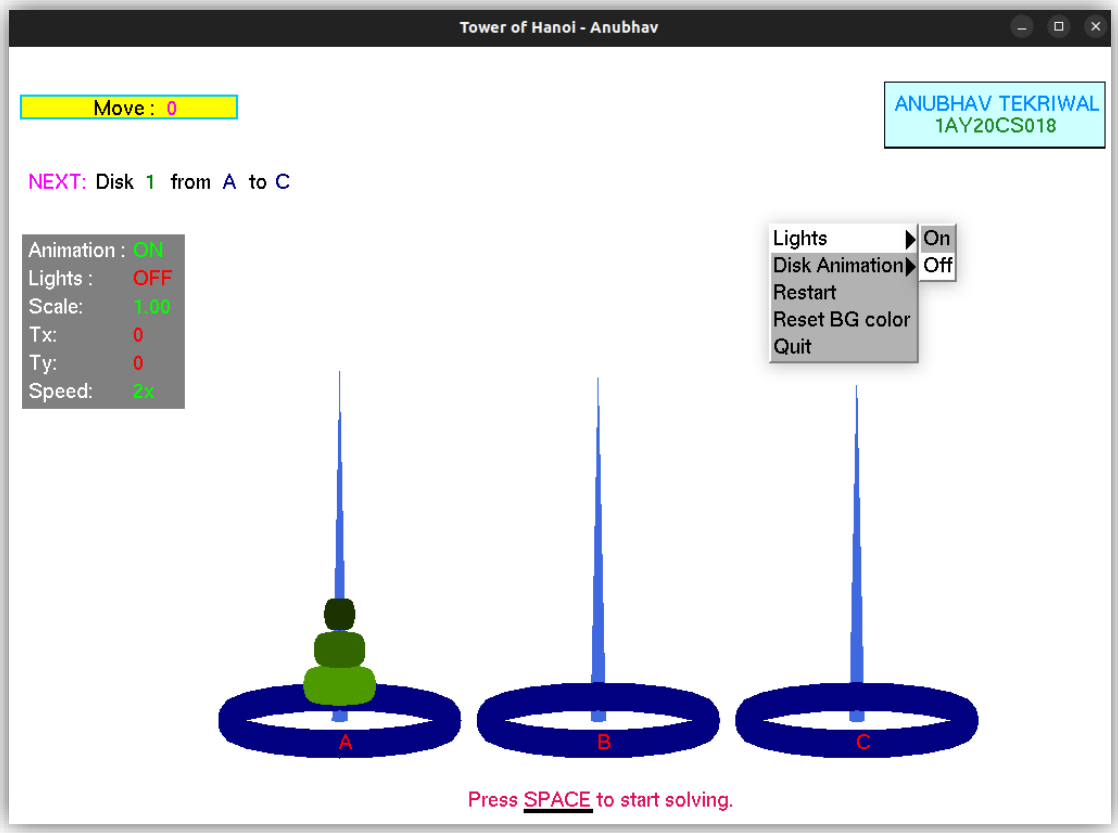

Fig 4.5: Lights On/Off and Main Menu

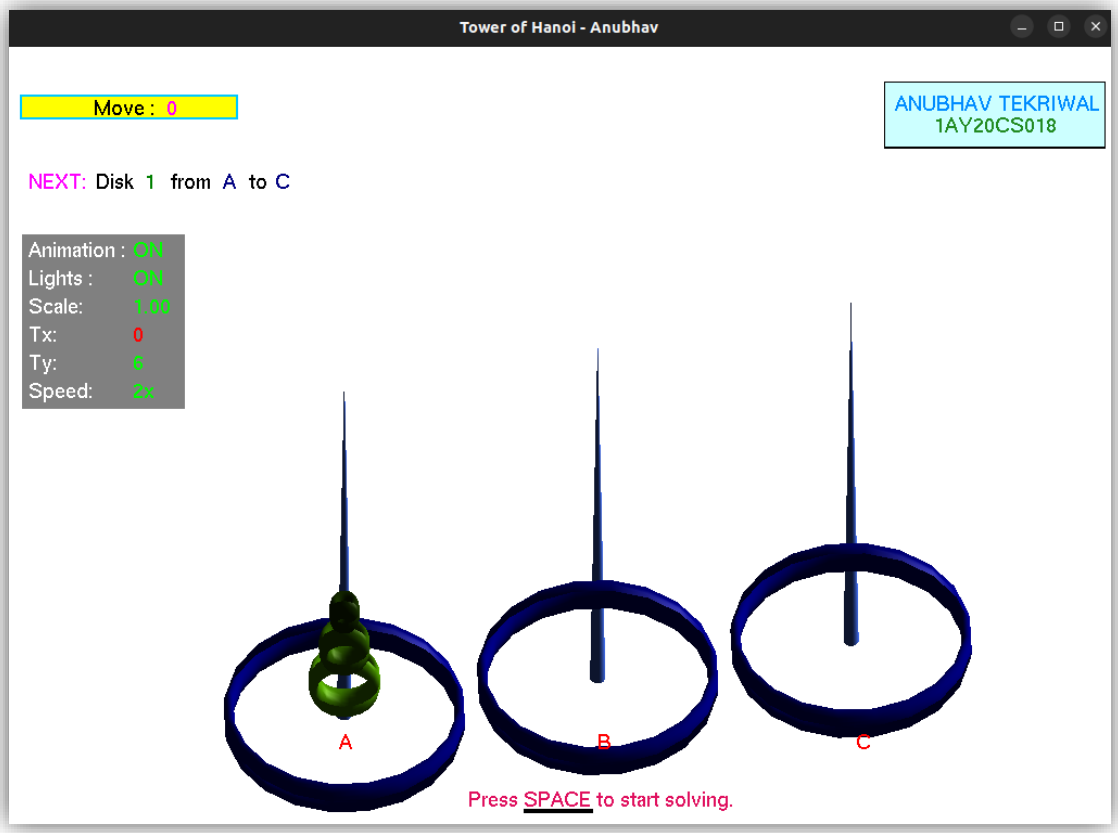

Fig 4.6: Perspective Viewing

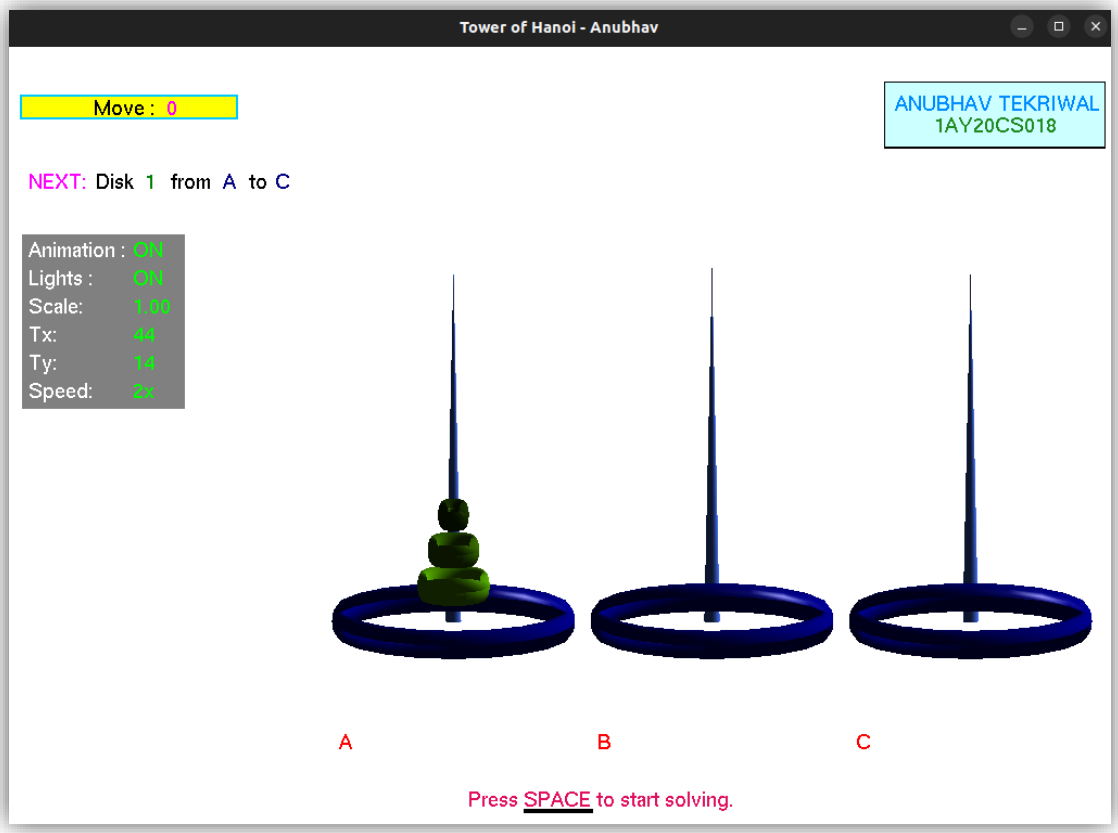

Fig 4.7: Translation

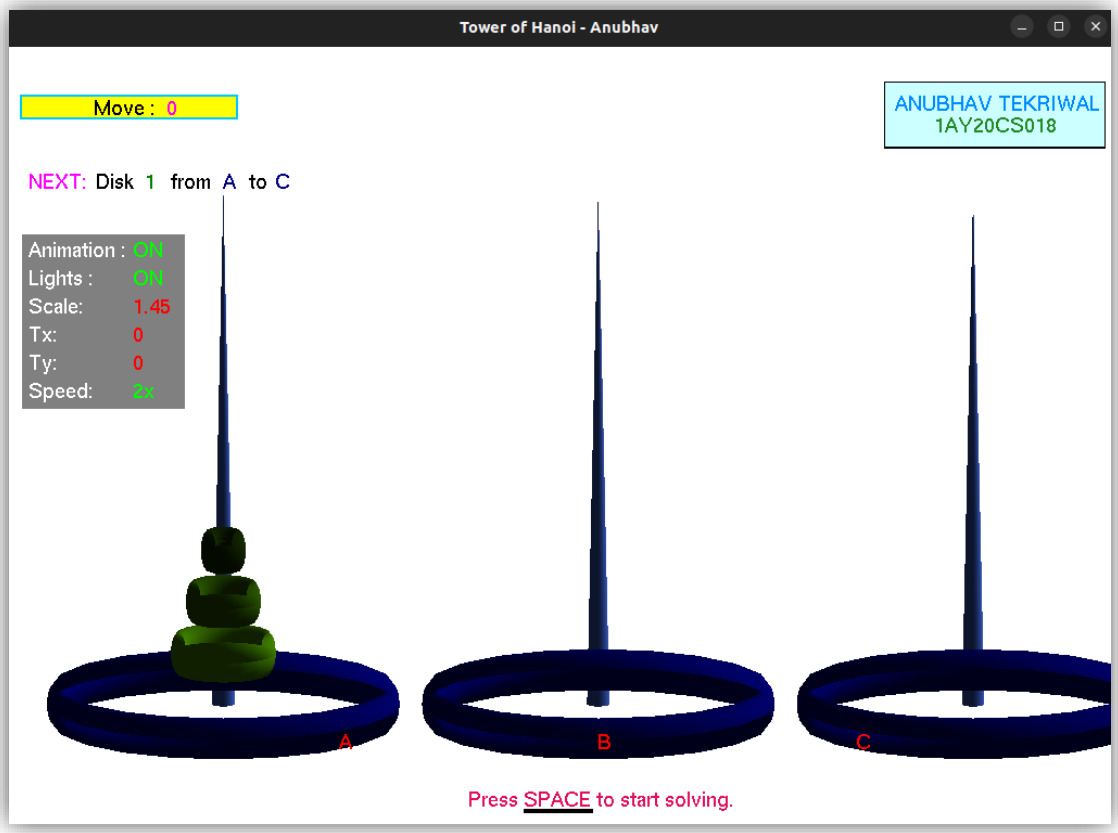

Fig 4.8: Scaling

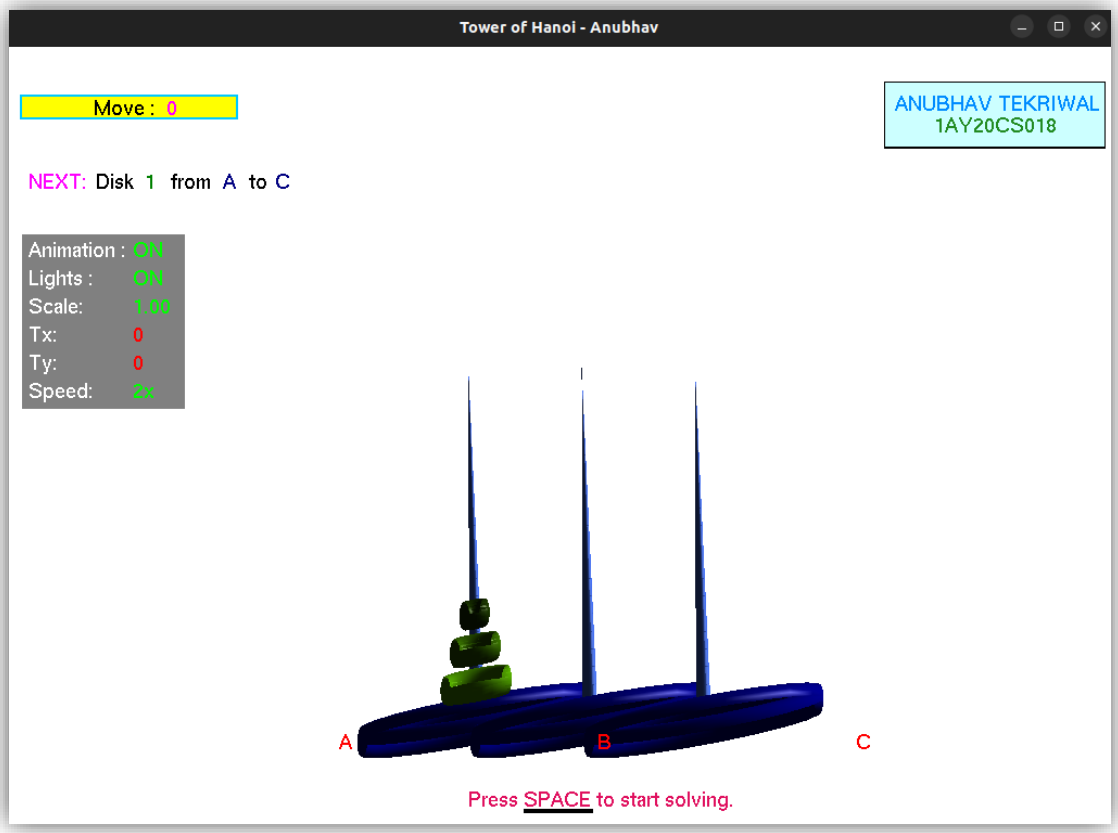

Fig 4.9: Rotation

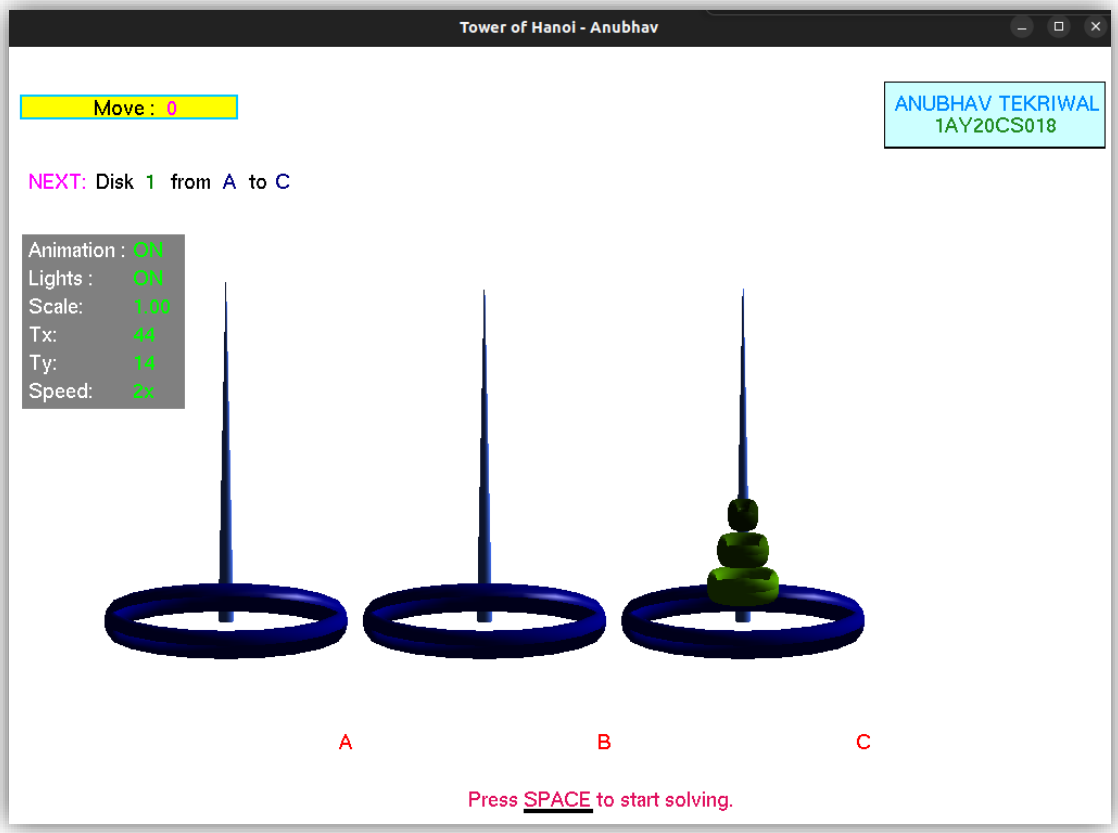

Fig 4.10: Reflection

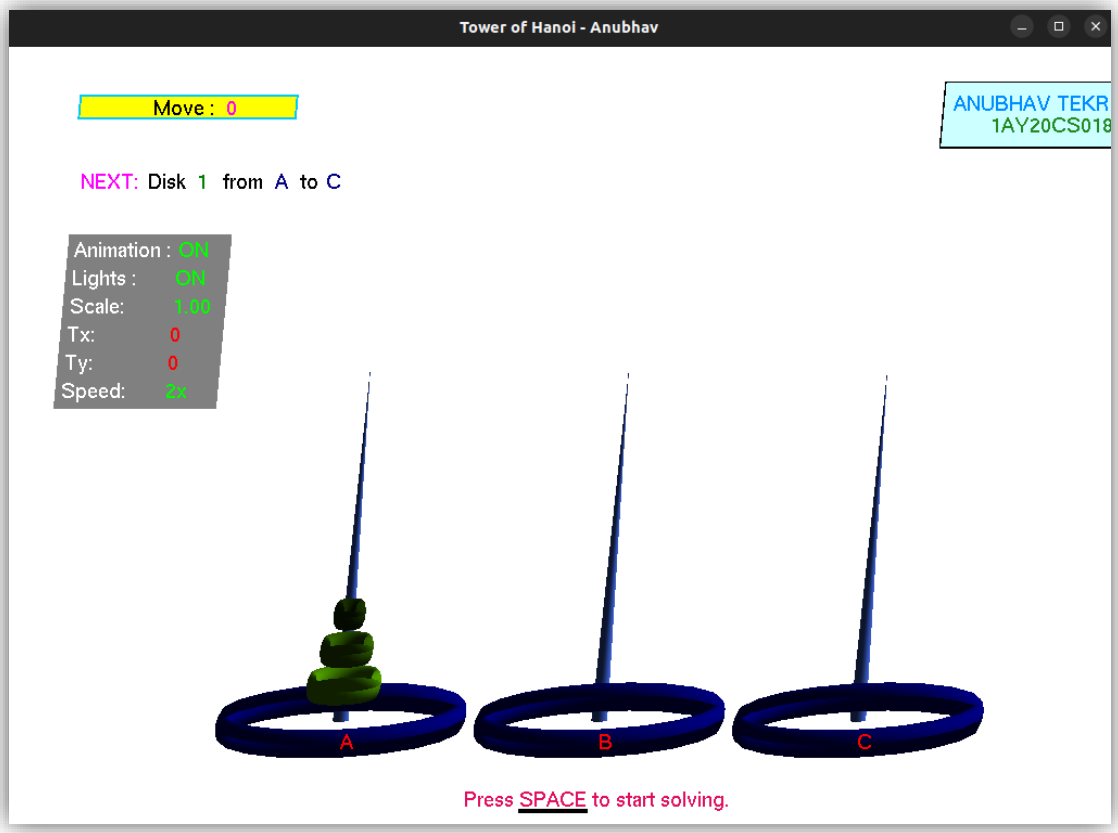

Fig 4.11: Shearing

Chapter 5

CONCLUSION AND FUTURE WORK

The 'Tower of Hanoi' simulation has been successfully implemented on the Linux operating system, utilizing the OpenGL library. This project offers users an easy-to-use and manipulative visual simulation, featuring various graphical elements such as lines, boxes, cones, tori, and polygons. The resulting interface is both simple and aesthetically pleasing.

Designing and developing this visual simulation of the 'Tower of Hanoi' proved to be an interesting and highly educational experience. It provided me with valuable insights into computer graphics, graphical user interface design, user interaction handling, and screen management. By exploring and implementing the capabilities of OpenGL, I was able to enhance the overall functionality and visual appeal of the simulation.

FUTURE WORK:

These are the features that are planned to be supported in the future:

- Customizable Number of Disks.
- Interactive Mode: Support for users to manually move the disks by dragging and dropping them between pegs.
- Advanced Analytics and Statistics: time taken to solve the puzzle, and comparisons with optimal or average solutions.
- Support for pattern filling.

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, 5 th edition. Pearson Education, 2008

[3] James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer graphics with OpenGL: pearson education

[4] Xiang, Plastock : Computer Graphics , sham's outline series, 2nd edition, TMG

[5] https://www.opengl.org/

[6] https://learnopengl.com/Getting-started/OpenGL

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

[8] https://www.khronos.org/opengl/

[9] https://learnopengl.com/Getting-started/OpenGL

[10] https://open.gl/

[11] https://www.opengl-tutorial.org/