CONTENTS

Chapter-1
Introduction

Introduction

What Is OpenGL?

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of *geometric primitives* - points, lines, and polygons.

A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS (Non-Uniform Rational B-Splines) curves and surfaces. GLU is a standard part of every OpenGL implementation. Also, there is a higher-level, object-oriented toolkit, Open Inventor, which is built atop OpenGL, and is available separately for many implementations of OpenGL.

OpenGL-Related Libraries

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow you to simplify your programming tasks, including the following:

- The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations

and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. GLU routines use the prefix **glu**.

- For every window system, there is a library that extends the functionality of that window system to support OpenGL rendering. For machines that use the X Window System, the OpenGL Extension to the X Window System (GLX) is provided as an adjunct to OpenGL. GLX routines use the prefix **glX**. For Microsoft Windows, the WGL routines provide the Windows to OpenGL interface. All WGL routines use the prefix **wgl**. For IBM OS/2, the PGL is the Presentation Manager to OpenGL interface, and its routines use the prefix **pgl**.

- The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window system APIs. GLUT routines use the prefix **glut.**

- Open Inventor is an object-oriented toolkit based on OpenGL which provides objects and methods for creating interactive three-dimensional graphics applications. Open Inventor, which is written in C++, provides prebuilt objects and a built-in event model for user interaction, high-level application components for creating and editing three-dimensional scenes, and the ability to print objects and exchange data in other graphics formats. Open Inventor is separate from OpenGL.

Include Files

For all OpenGL applications, you want to include the gl.h header file in every file. Almost all OpenGL applications use GLU, the aforementioned OpenGL Utility Library, which requires inclusion of the glu.h header file. So almost every OpenGL source file begins with

```
#include <GL/gl.h>

#include <GL/glu.h>
```

If you are directly accessing a window interface library to support OpenGL, such as GLX, AGL, PGL, or WGL, you must include additional header files. For example, if you are calling GLX, you may need to add these lines to your code

#include <X11/Xlib.h>

#include <GL/glx.h>

If you are using GLUT for managing your window manager tasks, you should include

#include <GL/glut.h>

Note that glut.h includes gl.h, glu.h, and glx.h automatically, so including all three files is redundant. GLUT for Microsoft Windows includes the appropriate header file to access WGL.

GLUT, the OpenGL Utility Toolkit

As you know, OpenGL contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse. Unfortunately, it's impossible to write a complete graphics program without at least opening a window, and most interesting programs require a bit of user input or other services from the operating system or window system.

In addition, since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects such as a sphere, a torus, and a teapot. This way, snapshots of program output can be interesting to look at. (Note that the OpenGL Utility Library, GLU, also has quadrics routines that create some of the same three-dimensional objects as GLUT, such as a sphere, cylinder, or cone.)

Important features of OpenGL Utility Toolkit (GLUT)

- Provides functionality common to all window systems.
- Open a window.
- Get input from mouse and keyboard.
- Menus.
- Event-driven.
- Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform.

- No slide bars.
- OpenGL is not object oriented so that there are multiple functions for a given logical function :
- glVertex3f
    - glVertex2i
    - glVertex3dv
- Underlying storage mode is the same easy to create overloaded functions in C++ but issue is efficiency.
- **OpenGL Interface**
    - GL (OpenGL in Windows)
    - GLU (graphics utility library) uses only GL functions, creates common objects (such as spheres)
    - GLUT (GL Utility Toolkit) interfaces with the window system
    - GLX: glue between OpenGL and Xwindow, used by GLUT

Window Management

Five routines perform tasks necessary to initialize a window.

- **glutInit**(int **argc*, char ***argv*) initializes GLUT and processes any command line arguments (for X, this would be options like -display and -geometry). **glutInit()** should be called before any other GLUT routine.
- **glutInitDisplayMode**(unsigned int *mode*) specifies whether to use an *RGBA* or color-index color model. You can also specify whether you want a single- or double-buffered window. (If you're working in color-index mode, you'll want to load certain colors into the color map; use **glutSetColor()** to do this.) Finally, you can use this routine to indicate that you want the window to have an associated depth, stencil, and/or accumulation buffer. For example, if you want a window with double buffering, the RGBA color model, and a depth

buffer, you might call **glutInitDisplayMode**(*GLUT_DOUBLE* | *GLUT_RGB* | *GLUT_DEPTH*).

- **glutInitWindowPosition**(int *x*, int *y*) specifies the screen location for the upper-left corner of your window.
- **glutInitWindowSize**(int *width*, int *size*) specifies the size, in pixels, of your window.
- int **glutCreateWindow**(char **string*) creates a window with an OpenGL context. It returns a unique identifier for the new window. Be warned: Until **glutMainLoop()** is called (see next section), the window is not yet displayed.

The Display Callback

glutDisplayFunc(void (**func*)(void)) is the first and most important event callback function you will see. Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by **glutDisplayFunc()** is executed. Therefore, you should put all the routines you need to redraw the scene in the display callback function.

If your program changes the contents of the window, sometimes you will have to call **glutPostRedisplay**(void), which gives **glutMainLoop()** a nudge to call the registered display callback at its next opportunity.

Running the Program

The very last thing you must do is call **glutMainLoop**(void). All windows that have been created are now shown, and rendering to those windows is now effective. Event processing begins, and the registered display callback is triggered. Once this loop is entered, it is never exited!

Graphics Functions

- Primitive functions:
  points, line segments, polygons, pixels, text, curves, surfaces
- Attributes functions:
  color, pattern, typeface

Some Primitive Attributes

glClearColor (red, green, blue, alpha); - Default = (0.0, 0.0, 0.0, 0.0)

glColor3f (red, green, blue); - Default = (1.0, 1.0, 1.0)

glLineWidth (width); - Default = (1.0)

glLineStipple (factor, pattern) - Default = (1, 0xffff)

glEnable (GL_LINE_STIPPLE);

glPolygonMode (face, mode) - Default = (GL_FRONT_AND_BACK, GL_FILL)

glPointSize (size); - Default = (1.0)

- Viewing functions:
  position, orientation, clipping
- Transformation functions:
  rotation, translation, scaling
- Input functions:
  keyboards, mice, data tablets
- Control functions:
  communicate with windows, initialization, error handling
- Inquiry functions: number of colors, camera parameters/values

Matrix Mode

There are two matrices in OpenGL:

- Model-view: defines COP and orientation
- Projection: defines the projection matrix

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 500.0, 0.0, 500.0);
glMatrixMode(GL_MODELVIEW);
```

Control Functions

- OpenGL assumes origin is bottom left
- glutInit(int *argcp, char **argv);
- glutCreateWindow(char *title);
- glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
- glutInitWindowSize(480,640);

- glutInitWindowPosition(0,0);
- OpenGL default: RGB color, no hidden-surface removal, single buffering

Obtaining Values of OpenGL State Variables

glGetBooleanv (paramname, *paramlist);

glGetDoublev (paramname, *paramlist);

glGetFloatv (paramname, *paramlist);

glGetIntegerv (paramname, *paramlist);

Saving and Restoring Attributes

glPushAttrib (group);

glPopAttrib ( );

where group = GL_CURRENT_BIT, GL_ENABLE_BIT, GL_LINE_BIT, GL_POLYGON_BIT, etc.

Projection Transformations

glMatrixMode (GL_PROJECTION);

glLoadIdentity ( );

glFrustum (left, right, bottom, top, near, far);

gluPerspective (fov, aspect, near, far);

glOrtho (left, right, bottom, top, near, far);

- Default = (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0)

gluOrtho2D (left, right, bottom, top);

Modelview Transformations

glMatrixMode (GL_MODELVIEW);

glLoadIdentity ( );

gluLookAt (eye_x, eye_y, eye_z, at_x, at_y, at_z, up_x, up_y, up_z);

glTranslatef (dx, dy, dz);

```
glScalef (sx, sy, sz);
glRotatef (angle, axisx, axisy, axisz);
```

Writing Bitmapped Text

```
glPixelStorei (GL_UNPACK_ALIGNMENT, 1);
glColor3f (red, green, blue);
glRasterPos2f (x, y);
glutBitmapCharacter (font, character);
```

where font = GLUT_BITMAP_8_BY_13, GLUT_BITMAP_HELVETICA_10, etc.

Managing the Frame Buffer

```
glutInit (&argc, argv);
glutInitDisplayMode (GLUT_RGB | mode);
glutInitWindowSize (width, height);
glutInitWindowPosition (x, y);
glutCreateWindow (label);
glClear (GL_COLOR_BUFFER_BIT);
glutSwapBuffers ( );
```

where mode = GLUT_SINGLE or GLUT_DOUBLE.

Registering Callbacks

```
glutDisplayFunc (callback);
glutReshapeFunc (callback);
glutDisplayFunc (callback);
glutMotionFunc (callback);
glutPassiveMotionFunc (callback);
glutMouseFunc (callback);
glutKeyboardFunc (callback);
```

```
id = glutCreateMenu (callback);
glutMainLoop ( );
```

Display Lists

```
glNewList (number, GL_COMPILE);
glEndList ( );
glCallList (number);
glDeleteLists (number, 1);
```

Managing Menus

```
id = glutCreateMenu (callback);
glutDestroyMenu (id);
glutAddMenuEntry (label, number);
glutAttachMenu (button);
glutDetachMenu (button);
```

where button = GLUT_RIGHT_BUTTON or GLUT_LEFT_BUTTON.

Chapter-2
Design

Design

Tower of Hanoi:

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

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

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

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

Recursive Solution:

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

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

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

1. move $n-1$ discs from A to B. This leaves disc *n* alone on peg A
2. move disc *n* from A to C

3. move $n-1$ discs from B to C so they sit on disc n

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

Explanation about the package:

This is a mini project on Tower of Hanoi based on OPENGL API for Computer graphics and visualization laboratory (10CSL67). The project simulates the optimal solution of the Tower of Hanoi problem given a number of disk. The project implements various geometric transformations like Translation, Rotation, and Scaling. The basic model consists of three poles, source, auxiliary and destination. The disks initially resting on the source pole reaches the destination. The poles are drawn using the GLUT library function glutSolidCone(), and the disks are drawn using glutSolidTorus().Initially the user is presented with an introduction scene which has a menu to choose the number of disks. On pressing the Enter key the actual simulation scene is loaded. The user can use the mouse wheel to advance through the simulation. An optional animation has been implemented, which shows the movement of the disks from pole to pole. The viewing model used is an Orthogonal Projection. The moves to be performed as per the optimal solution are displayed on the top left of the screen is a raster text using the function glutBitmapCharacter(). Lighting has been implemented by the inbuilt OPENGL lighting functions. Menus have been provided to modify various features such as Lighting, Move camera, animation, change background color, to automatically simulate the complete solution, restart the simulation and to exit the program. The movement of the camera is only along the Y axis, and is implemented using the gluLookAt() function.

Chapter-3
Implementation

Complete Source Code of the package :

```
#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
int pos[16] = {10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85};
int peg[3] = {50,150,250};
int moves[10000][3];
int max_moves;
int POLES[3][10];
int top[3]={-1,-1,-1};
int NUM_DISKS=3;
int cnt,counter,speed=20;
int line1=90,line2=85;
float ycoordinate;
int lightflag=1,animationFlag=1,randomColorFlag=0;

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

void pop(int p)
{
```

```
                top[p]--;
}

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

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

```
                }

}
void drawSolved()
{
                glColor3f(1,1,0);
                glRasterPos3f(-60,87,0);
                printString("Solved !!");
                glColor3f(0.6,0.3,0.5);
                glBegin(GL_POLYGON);
                  glVertex3f(-75,93,-5);
                  glVertex3f(-75,83,-5);
                  glVertex3f(10,83,-5);
                  glVertex3f(10,93,-5);
                glEnd();
                glColor3f(1,0,0);
                glRasterPos3f(peg[0],70,0);
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'A');
                glRasterPos3f(peg[1],70,0);
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'B');
                glRasterPos3f(peg[2],70,0);
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'C');
}

void display()
{
                int i,j,k;
                if(randomColorFlag)
```

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

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

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

void mouse(int btn,int mode,int x,int y)
{
                if(btn == 4 && mode == GLUT_DOWN)
                {
                  if(counter<max_moves)
                  {
                        pop(moves[counter][1]);
                        if(animationFlag)

                animate(moves[counter][0],moves[counter][1],moves[counter][2]);
                        push(moves[counter][2],moves[counter][0]);
                        counter++;
                  }
                }
                if(btn == 3 && mode == GLUT_DOWN)
```

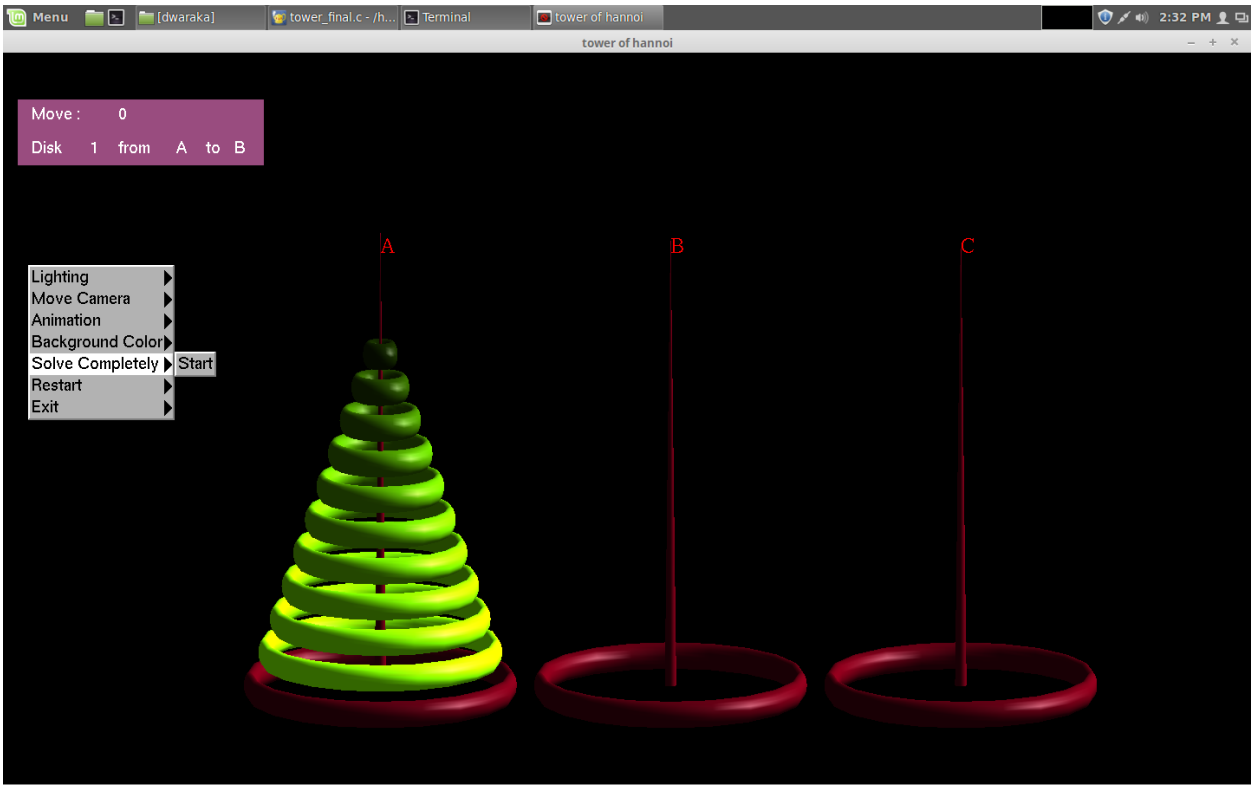
```
                {
                  if(counter>0)
                  {
                        counter--;
                        pop(moves[counter][2]);
                        if(animationFlag)

                animate(moves[counter][0],moves[counter][2],moves[counter][1]);
                        push(moves[counter][1],moves[counter][0]);
                  }
                }
                glutPostRedisplay();
}

void restart()
{
                int i;
                memset(POLES,0,sizeof(POLES));
                memset(moves,0,sizeof(POLES));
                memset(top,-1,sizeof(top));
                cnt=0,counter=0;
                ycoordinate=0.1;
                max_moves = pow(2,NUM_DISKS)-1;
                for(i=NUM_DISKS;i>0;i--)
                {
                  push(0,i);
                }
                tower(NUM_DISKS,0,1,2);
}
```

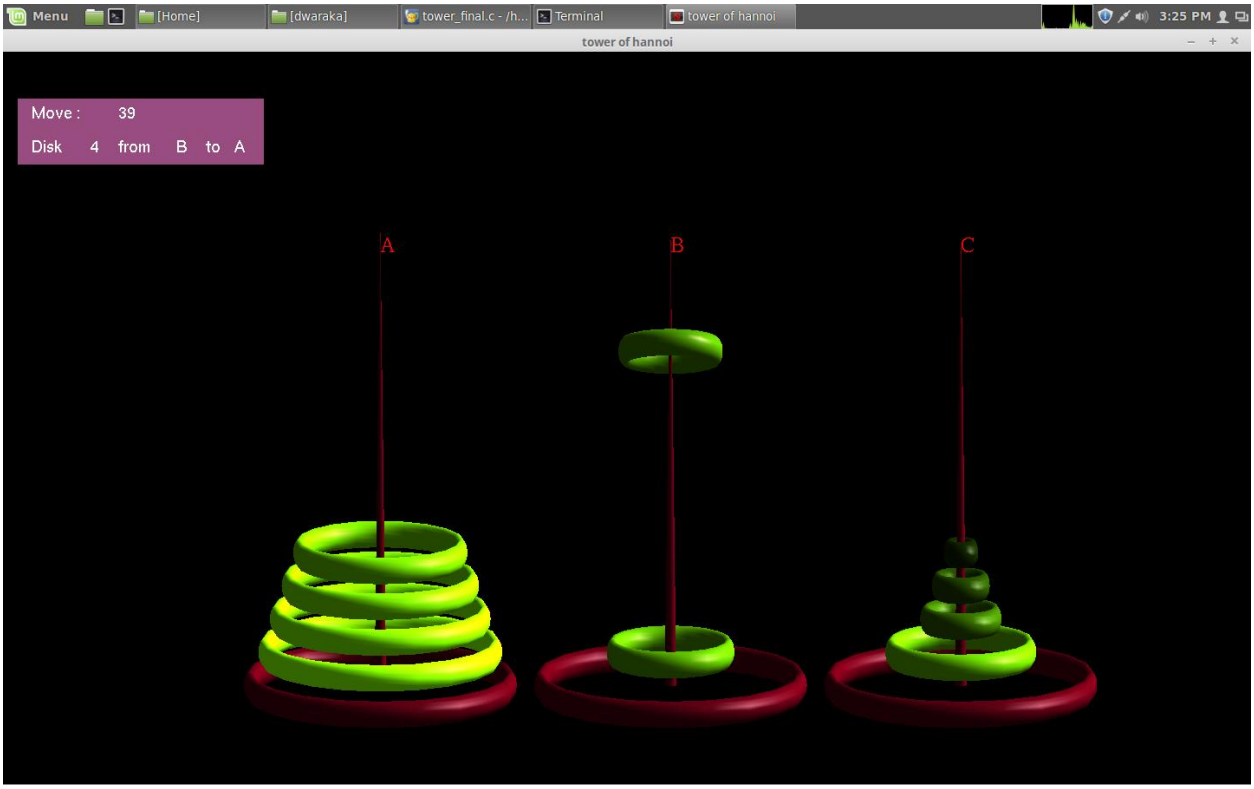
```
void processMenuNumDisks(int option)
{
                        NUM_DISKS=option;
                        restart();
                        glutPostRedisplay();
}

void strokeString(float x,float y,float sx,float sy,char *string,int width)
{
                        char *c;
                        glLineWidth(width);
                        glPushMatrix();
                        glTranslatef(x,y,0);
                        glScalef(sx,sy,0);
                        for (c=string; *c != '\0'; c++) {
                          glutStrokeCharacter(GLUT_STROKE_ROMAN, *c);
                        }
                        glPopMatrix();
}
void keyboard(unsigned char c, int x, int y)
{
                        switch(c)
                        {
                          case 13:
                                  restart();
                                  init();
                                  glutDisplayFunc(display);
                                  createGLUTMenus2();
```

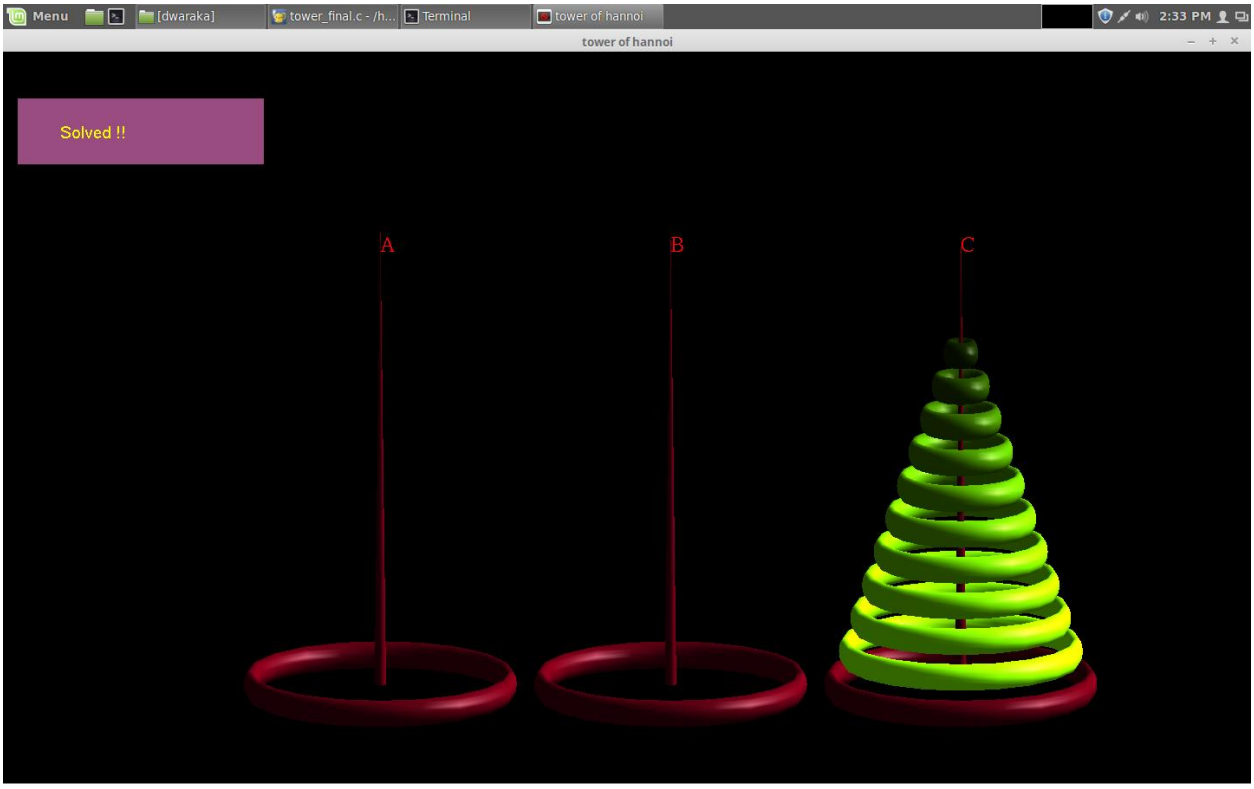
```
                    glutKeyboardFunc(keyboard2);
                    glutMouseFunc(mouse);
                  break;
                }
                glutPostRedisplay();
}

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

Chapter-4
Results


Menu
[dwaraka]
tower_final.c - /h...
Terminal
tower of hannoi
2:32 PM
tower of hannoi
BANGALORE INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Number of Disks
Exit
3
4
5
6
7
8
9
10
AN OPENGL MINI PROJECT ON
TOWER OF HANOI
NUMBER OF DISKS: 3
By:
Dwarakanandan.B.M
1BI13CS052
Under the guidance of.
Girija.J(Associate Professor)
1 . Set the number of disks using the menu
2 . Press (Enter) to start the simulation


Menu
[dwaraka]
tower_final.c - /h...
Terminal
tower of hannoi
2:32 PM
tower of hannoi
Move : 0
Disk 1 from A to B
A
B
C
Lighting
Move Camera
Animation
Background Color
Solve Completely
Start
Restart
Exit


Menu
[Home]
[dwaraka]
tower_final.c - /h...
Terminal
tower of hannoi
3:25 PM
tower of hannoi
Move : 39
Disk 4 from B to A
A
B
C


Menu
[dwaraka]
tower_final.c - /h...
Terminal
tower of hannoi
2:33 PM
tower of hannoi
Solved !!
A
B
C

Chapter-5
Conclusion

Conclusion

It was a wonderful learning experience for me working on this project. This project took me through various phases of project development and gave me real insight into the world of software engineering. The joy of working and the thrill involved while tackling the various problems and challenges gave me a feel of developers industry.

It was due to project that I came to know how professional software's are designed.

I enjoyed each and every bit of work I had to put into this project.

Chapter-6
Bibliography

Bibliography

- Interactive Computer Graphics – Edward Angel
- Official OPENGL Documentation at https://www.**opengl**.org/**documentation**/
- OpenGL Programming Guide: The Official Guide to Learning OpenGL- Dave Shreiner.