**CHAPTER 1**

**INTRODUCTION TO PROJECT**

**N - QUEEN:**

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

**TOWER OF HANOI:**

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1) Only one disk can be moved at a time.

2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

3) No disk may be placed on top of a smaller disk..

It allows user to choose number of disc. It show the time taken to finish the simulation. The speed of simulation can be increased as well as decrement with keyword interaction. There are many views which can be zoom in and zoom out with help of some keys.

**CHAPTER 2**

**OpenGL**

**2.1** **INTRODUCTION TO OPENGL**

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing application that produces 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 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. It is developed by many companies and it is free to use. You can develop openGl application 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.

**2.2 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 library functions are:

1. GL library (OpenGL in Windows) – Main function for Windows
2. GLU(OpenGL utility library) – Creating and viewing objects.
3. GLUT (OpenGL utility toolkit) - Function that help in creating interface of windows

OpenGL draws primitives- points, lines, or polygons , which are 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 from of function calls. These libraries are included in the application program using preprocessor directives.

#include<GL/glut.h>

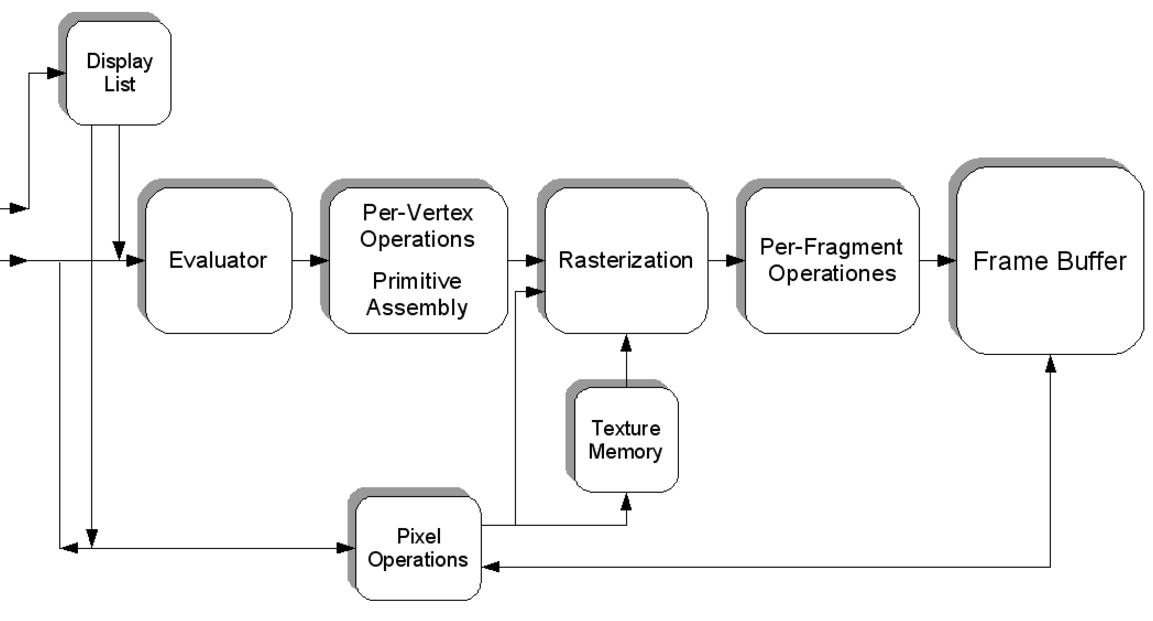
OpenGL user Interface Library (GLUI) is a C++ user interface library based on the OpenGL utility Toolkit (GLUT) which provides controls such as buttons, checkboxes, radio buttons to handle all system dependent OpenGL applications. It is window and operating system independent, relying on GLUT to handle 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 routine that OpenGL provides. It is usually distributed with the base OpenGL package.

**2.3 GRAPHICS PIPELINE ARCHITECTURE**

The figure shown gives an abstract, high level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

The concept of graphics pipeline is what really sets it apart from general CPUs, although the idea of a pipeline is the same as that used by general purpose CPUs. The graphics pipeline is built in stages. Every stage is specialized in precisely one element of the rendering process. Once we are familiar with these tasks, we will be able to recognize them in the designs of the GPU. Lets now take a look at the stages



As shown in the figure , rather than having all commands the flow proceeds immediately through the pipeline. User can choose to accumulate some of them in a display list for processing at a later time. The evaluator stage of processing provides an efficient means for approximating curve and surfaces geometry by evaluating polynomial commands of input values. During the next stage, per vertex operations and primitive assembly, OpenGL processes all of which are described by vertices are defined. Vertices are transformed and lit, primitives are clipped to the viewport in preparation for the next stage. Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each fragment so produced id fed into the last stage, per-fragment operations, which perform the final operations on the data before its stored as pixels in the frame buffer. These operations include conditional updates to the frame buffer based on incoming and previously stored Z-values and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values. Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the pixel operations stage. The result of this stage is either stored as textual memory for use in the rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data. All the Elements of openGL state including the contents of textual memory and even the frame buffer, can be obtained by an OpenGl application.

**2.4 VIEWING**

**2.4.1 The viewing transformation**

void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz);

Defines a viewing matrix and multiplies it to the right of the current matrix. The desired viewpoint is specified by eyex, eyey, and eyez. The centerx, centery, and centerz arguments specify any point along the desired line of sight, but typically they're some point in the center of the scene being looked at. The upx, upy, and upz arguments indicate which direction is up (that is, the direction from the bottom to the top of the viewing volume).

**2.4.2 Modeling transformation**

**Translate**

void glTranslate{fd}(TYPEx, TYPE y, TYPEz);

Multiplies the current matrix by a matrix that moves (translates) an object by the given x, y, and z values (or moves the local coordinate system by the same amounts).

**Rotate**

void glRotate{fd}(TYPE angle, TYPE x, TYPE y, TYPE z);

Multiplies the current matrix by a matrix that rotates an object (or the local coordinate system) in a counterclockwise direction about the ray from the origin through the point (x, y, z). The angle parameter specifies the angle of rotation in degrees.

**Scale**

void glScale{fd}(TYPEx, TYPE y, TYPEz);

Multiplies the current matrix by a matrix that stretches, shrinks, or reflects an object along the axes. Each x, y, and z coordinate of every point in the object is multiplied by the corresponding argument x, y, or z. With the local coordinate system approach, the local coordinate axes are stretched, shrunk, or reflected by the x, y, and z factors, and the associated object is transformed with them.

**2.4.3 Projection transformation**

**Perspective projection**

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

Creates a matrix for a perspective-view frustum and multiplies the current matrix by it. The frustum's viewing volume is defined by the parameters: (left, bottom, -near) and (right, top, -near) specify the (x, y, z) coordinates of the lower-left and upper-right corners of the near clipping plane; near and far give the distances from the viewpoint to the ner and far clipping planes. They should always be positive.

void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);

Creates a matrix for a symmetric perspective-view frustum and multiplies the current matrix by it. fovy is the angle of the field of view in the x-z plane; its value must be in the range [0.0,180.0]. aspect is the aspect ratio of the frustum, its width divided by its height. near and far values the distances between the viewpoint and the clipping planes, along the negative z-axis. They should always be positive.

**Orthographic projection**

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

Creates a matrix for an orthographic parallel viewing volume and multiplies the current matrix by it. (left, bottom, -near) and (right, top, -near) are points on the near clipping plane that are mapped to the lower-left and upper-right corners of the viewport window, respectively. (left, bottom, -far) and (right, top, -far) are points on the far clipping plane that are mapped to the same respective corners of the viewport. Both near and far can be positive or negative.

**2.4.4 Viewing volume clipping**

**Viewport transformation**

void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);

Defines a pixel rectangle in the window into which the final image is mapped. The (x, y) parameter specifies the lower-left corner of the viewport, and width and height are the size of the viewport rectangle. By default, the initial viewport values are (0, 0, winWidth, winHeight), where winWidth and winHeight are the size of the window.

**2.4.5 Manipulating the matrix stacks**

void glPushMatrix(void);

Pushes all matrices in the current stack down one level. The current stack is determined by glMatrixMode(). The topmost matrix is copied, so its contents are duplicated in both the top and second-from-the-top matrix. If too many matrices are pushed, an error is generated.

void glPopMatrix(void);

Pops the top matrix off the stack, destroying the contents of the popped matrix. What was the second-from-the-top matrix becomes the top matrix. The current stack is determined by glMatrixMode(). If the stack contains a single matrix, calling glPopMatrix() generates an error.

**2.5 OPENGL LIGHTING FUNCTION**

**2.5.1 A Hidden-Surface Removal Survival Kit**

A depth buffer works by associating a depth, or distance, from the view plane (usually the near clipping plane), with each pixel on the window. Initially, the depth values for all pixels are set to the largest possible distance (usually the far clipping plane) using the glClear() command with GL\_DEPTH\_BUFFER\_BIT. Then the objects in the scene are drawn in any order.

To use depth buffering, you need to enable depth buffering. This has to be done only once. Before drawing, each time you draw the scene, you need to clear the depth buffer and then draw the objects in the scene in any order.

**2.5.2 Real world and OpenGL Lighting**

The OpenGL lighting model considers the lighting to be divided into four independent components: emissive, ambient, diffuse, and specular.

***ambient*** illumination is light that's been scattered so much by the environment that itsdirection is impossible to determine - it seems to come from all directions.

The ***diffuse*** component is the light that comes from one direction, so it's brighter if it comes squarely down on a surface than if it barely glances off the surface. Once it hits a surface, however, it's scattered equally in all directions, so it appears equally bright, no matter where the eye is located.

***Specular*** light comes from a particular direction, and it tends to bounce off thesurface in a preferred direction. A well-collimated laser beam bouncing off a high-quality mirror produces almost 100 percent specular reflection. Shiny metal or plastic has a high specular component, and chalk or carpet has almost none. You can think of specularity as shininess.

**Creating light source**

The command used to specify all properties of lights is **glLight\*().** void glLight{if}(GLenum light, GLenum pname, TYPEparam); void glLight{if}v(GLenum light, GLenum pname, TYPE \*param);

Creates the light specified by light, which can be GL\_LIGHT0, GL\_LIGHT1, ... , or GL\_LIGHT7. The characteristic of the light being set is defined by pname, which specifies a named parameter param indicates the values to which the pname characteristic is set; it's a pointer to a group of values if the vector version is used, or the value itself if the nonvector version is used. The nonvector version can be used to set only single-valued light characteristics.

examples: -

GLfloat light\_ambient[] = { 0.0, 0.0, 0.0, 1.0 }; GLfloat light\_diffuse[] = { 1.0, 1.0, 1.0, 1.0 }; GLfloat light\_specular[] = { 1.0, 1.0, 1.0, 1.0 }; GLfloat light\_position[] = { 1.0, 1.0, 1.0, 0.0 };

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient); glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse); glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular); glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

**Enabling lighting**

With OpenGL, you need to explicitly enable (or disable) lighting. glEnable(GL\_LIGHTING);

To disable lighting, call **glDisable()** with GL\_LIGHTING as the argument.

**2.6 BLENDING, ANTIALIASING AND FOG**

"Blending" tells you how to specify a blending function that combines color values from a source

and a destination. The final effect is that parts of your scene appear translucent.

"Antialiasing" explains this relatively subtle technique that alters colors so that the edges of points,

lines, and polygons appear smooth rather than angular and jagged.

"Fog" describes how to create the illusion of depth by computing the color values of an object

based on its distance from the viewpoint. Thus, objects that are far away appear to fade into the background, just as they do in real life.

**2.7 BITMAPS AND FONTS**

OpenGL provides only the lowest level of support for drawing strings of characters and manipulating fonts. The commands glRasterPos\*() and glBitmap() position and draw a single bitmap on the screen.

**Current raster position**

void glRasterPos{234}{sifd}(TYPE x, TYPE y, TYPE z, TYPE w); void glRasterPos{234}{sifd}v(TYPE \*coords);

Sets the current raster position. The x, y, z, and w arguments specify the coordinates of the raster position. If the vector form of the function is used, the coords array contains the coordinates of the raster position. If glRasterPos2\*() is used, z is implicitly set to zero and w is implicitly set to one; similarly, with glRasterPos3\*(), w is set to one.

Examples: - glRasterPos2i(20, 20);

**2.8 OPENGL CONTRIBUTIONS**

It is very popular in the video games development industry where it competes with direct 3D window. 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 platform 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.

**2.9 LIMITATIONS**

* OpenGL is case sensitive
* Line Color, filled faces and fill color not supported.
* Bump mapping is not supported
* Navigation rendering is not supported
* Shadow plane is not supported.
* 3D measurement is not supported
* Streaming of individual 3D objectives is not supported

**CHAPTER 3**

**SYSTEM REQUIREMENTS**

**3.1 HARDWARE REQUIREMENTS**

The Hardware requirements are very minimal and the program can be run on most of the machines.

Processor : Intel Core i2

Processor Speed : 2.20 GHZ

RAM : 4 GB

Storage Space : 500 GB

Monitor Resolution : 1024\*768 or 1336\*768 or 1280\*1024

**3.2 SOFTWARE REQUIREMENTS**

Operating System : Windows 7

IDE : Microsoft Visual Studio 2008/2010

* OpenGL libraries
* Header Files which includes GL/glut.h
* object File Libraries
* glu32.lib, Opengl132.lib
* glut32.lib,DLL files
* glu32.dll,glut32.dll
* opengl132.dll

**CHAPTER 4**

**FUNCTION DESCRIPTIONS**

The description of all the function 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 width,int height)**

This specifies the initial position of top-left corner of the windows 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 function that is executed when the window needs to be redrawn.

* **Void glClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a)**

This sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating point numbers between 0 and 1.

* **Void glClear(GLbitfield mask)**

It clears buffers to present values. The value of mask is determined by the bitwise OR of options GL\_COLOR\_BUFER\_BIT or GL\_DEPTH\_BUFFER\_BIT.

* **Void glutPostRedisplay()**

This function requests that the callback be executed after the current callback returns.

* **Void glutReshapeFunc(int width,int height)**

This function registers the reshape callback function. The call back function returns the height and width of the new window. The reshape callback invokes a display callback.

* **Void glviewport (int x,int y, GLsizei width, Glsizei height)**

This function specifies a width\* height viewport in pixels whose lower left corner is at(x,y) measured from the origin of the window.

* **Void glMatrixMode(GLenum mode)**

This function specifies which matrix will be affected by subsequent transformations.

Mode can be GL\_MODEL\_VIEW, GL\_PROJECTION, GL\_TEXTURE

* **Void glLoadIdentity()**

This function sets the current transformation matrix to an identity matrix.

* **Void gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)**

This function defines two dimensional viewing rectangle in the plane z=0.

* **Void glutMouseFunc ( int button,int state,int x,int y)**

This function registers the mouse call back function.The function returns the state of the button , the button pressed and the position of the mouse relative to the top left corner of the window.

* **Void glVertex3f(TYPE xcoordinate, TYPE ycoordinate, TYPE zcoordinate)**

This specifies the position of a vertex in 3 dimensions. If v is present, the argument is a pointer to an array containing the coordinates.

* **Void glBegin(glEnum mode)**

This function initiates a new primitive of type mode and starts the collection of vertices.

* **Void glEnd()**

This function termiates a list of vertices.

* **Void glutMainLoop()**

This function cause the program to enter an infinite loop. It should be the last statement in the main.

**CHAPTER 5**

**SOURCE CODE**

**TOWER OF HANOI SOURCE CODE:**

#include<GL/glew.h>

#include<GL/freeglut.h>

#include<stdio.h>

#include<stdlib.h>

#define BREITE 0.1f

#define STANGENBREITE 0.025f

#define SLICES 32

#define INNERSLICES 16

#define LOOPS 1

#define FPS 64 /\* more looks nicer, uses more cpu power \*/

#define FEM 1000.0/FPS

#define FSEM 0.001f /\* speed (bigger is faster)\*/

struct config

{

GLfloat gap;

GLfloat pinradius;

GLfloat pinheight;

};

struct action

{

char fromstack;

char tostack;

struct action\* next;

};

typedef struct action action;

struct actions

{

action\* head;

action\* tail;

};

typedef struct actions actions;

struct disk

{

char color;

GLfloat radius;

struct disk\* next;

struct disk\* prev;

};

typedef struct disk disk;

struct stack

{

disk\* bottom;

disk\* top;

};

typedef struct stack stack;

int disks = 3; //initial number of disks

GLfloat rotX, rotY, zoom, offsetY = 1.5, speed;

GLUquadricObj\* quadric;

GLfloat pos;

GLboolean fullscreen;

stack pin[3];

float pinheight[3];

struct config config;

actions actqueue;

action\* curaction;

disk\* curdisk;

int duration;

char seconds[24] = "Time: 0s";

int draw, maxdraws;

//function prototypes

void moveDisk(int param);

void hanoiinit(void);

void reset();

void Display(void);

void hanoi(actions\* queue, const int n, const char pin1, const char pin2, const char pin3);

void push(stack\* pin, disk\* item);

disk\* pop(stack\* pin);

void drawDisk(GLUquadricObj\*\* quadric, const GLfloat outer, const GLfloat inner);

void drawPin(GLUquadricObj\*\* quadric, const GLfloat radius, const GLfloat height);

void drawAllPins(GLUquadricObj\*\* quadric, const GLfloat radius, const GLfloat height, const GLfloat gap);

void drawBitmapString(const GLfloat x, const GLfloat y, const GLfloat z, void\* font, char\* string);

void drawBitmapInt(const GLfloat x, const GLfloat y, const GLfloat z, void\* font, const int number);

void hanoi(actions\* queue, const int n, const char pin1, const char pin2, const char pin3)

{

action\* curaction;

if (n > 0)

{

hanoi(queue, n - 1, pin1, pin3, pin2);

/\* push action into action queue \*/

curaction = (action\*)malloc(sizeof(action));

curaction->next = NULL;

curaction->fromstack = pin1;

curaction->tostack = pin3;

if (queue->head == NULL)

queue->head = curaction;

if (queue->tail != NULL)

queue->tail->next = curaction;

queue->tail = curaction;

hanoi(queue, n - 1, pin2, pin1, pin3);

}

}

/\*\* push item to pin \*/

void push(stack\* pin, disk\* item) {

item->next = NULL;

if (pin->bottom == NULL) {

pin->bottom = item;

pin->top = item;

item->prev = NULL;

}

else {

pin->top->next = item;

item->prev = pin->top;

pin->top = item;

}

}

/\*\* pop item from pin \*/

disk\* pop(stack\* pin) {

disk\* tmp;

if (pin->top != NULL) {

tmp = pin->top;

if (pin->top->prev != NULL) {

pin->top->prev->next = NULL;

pin->top = tmp->prev;

}

else {

pin->bottom = NULL;

pin->top = NULL;

}

return tmp;

}

return NULL;

}

void drawDisk(GLUquadricObj\*\* quadric, const GLfloat outer, const GLfloat inner)

{

glPushMatrix();

glRotatef(-90.0, 1.0, 0.0, 0.0);

gluCylinder(\*quadric, outer, outer, BREITE, SLICES, LOOPS);

gluQuadricOrientation(\*quadric, GLU\_INSIDE);

if (inner > 0)

gluCylinder(\*quadric, inner, inner, BREITE, INNERSLICES, LOOPS);

gluDisk(\*quadric, inner, outer, SLICES, LOOPS);

gluQuadricOrientation(\*quadric, GLU\_OUTSIDE);

glTranslatef(0.0, 0.0, BREITE);

gluDisk(\*quadric, inner, outer, SLICES, LOOPS);

gluQuadricOrientation(\*quadric, GLU\_OUTSIDE);

glPopMatrix();

}

void drawPin(GLUquadricObj\*\* quadric, const GLfloat radius, const GLfloat height)

{

glPushMatrix();

glRotatef(-90.0, 1.0, 0.0, 0.0);

gluCylinder(\*quadric, radius, radius, BREITE / 2, SLICES, LOOPS);

gluQuadricOrientation(\*quadric, GLU\_INSIDE);

gluDisk(\*quadric, 0.0, radius, SLICES, LOOPS);

gluQuadricOrientation(\*quadric, GLU\_OUTSIDE);

glTranslatef(0.0, 0.0, BREITE / 2);

gluDisk(\*quadric, 0.0, radius, SLICES, LOOPS);

gluCylinder(\*quadric, STANGENBREITE, STANGENBREITE, height, INNERSLICES, LOOPS);

glTranslatef(0.0, 0.0, height);

gluDisk(\*quadric, 0.0, STANGENBREITE, INNERSLICES, LOOPS);

glPopMatrix();

}

void drawAllPins(GLUquadricObj\*\* quadric, const GLfloat radius, const GLfloat height, const GLfloat gap)

{

glPushMatrix();

drawPin(quadric, radius, height);

glTranslatef(-gap, 0.0, 0.0);

drawPin(quadric, radius, height);

glTranslatef(gap \* 2, 0.0, 0.0);

drawPin(quadric, radius, height);

glPopMatrix();

}

void drawBitmapString(const GLfloat x, const GLfloat y, const GLfloat z, void\* font, const char\* string)

{

const char\* c;

glRasterPos3f(x, y, z);

for (c = string; \*c != '\0'; c++)

glutBitmapCharacter(font, \*c);

}

void drawBitmapInt(const GLfloat x, const GLfloat y, const GLfloat z, void\* font, const int number)

{

char string[17];

// sprintf\_s(string, "%d", number);

drawBitmapString(x, y, z, font, string);

}

void populatePin(void)

{

int i;

disk\* cur;

GLfloat radius = 0.1f \* disks;

for (i = 0; i < disks; i++)

{

cur = (disk\*)malloc(sizeof(disk));

cur->color = (char)i % 6;

cur->radius = radius;

push(&pin[0], cur);

radius -= 0.1;

}

duration = 0;

draw = 0;

}

void clearPins(void)

{

int i;

disk\* cur, \* tmp;

free(curdisk);

curdisk = NULL;

for (i = 0; i < 3; i++)

{

cur = pin[i].top;

while (cur != NULL)

{

tmp = cur->prev;

free(cur);

cur = tmp;

}

pin[i].top = NULL;

pin[i].bottom = NULL;

}

}

void hanoiinit(void)

{

GLfloat radius;

speed = FSEM \* FEM;

radius = 0.1f \* disks;

config.pinradius = radius + 0.1f;

config.gap = radius \* 2 + 0.5f;

config.pinheight = disks \* BREITE + 0.2f;

maxdraws = (2 << (disks - 1)) - 1; //calculate minimum number of moves

populatePin();

/\* calculate actions; initialize action list \*/

actqueue.head = NULL;

hanoi(&actqueue, disks, 0, 1, 2);

curaction = actqueue.head;

curdisk = pop(&pin[(int)curaction->fromstack]);

pos = 0.001;

}

void reset(void)

{

clearPins();

populatePin();

/\* reset actions \*/

curaction = actqueue.head;

curdisk = pop(&pin[(int)curaction->fromstack]);

pos = 0.001;

}

void hanoicleanup(void)

{

action\* acur, \* atmp;

clearPins();

acur = actqueue.head;

do {

atmp = acur->next;

free(acur);

acur = atmp;

} while (acur != NULL);

gluDeleteQuadric(quadric);

}

void setColor(const int color)

{

switch (color) {

case 0:

glColor3f(1.0, 0.0, 0.0);

break;

case 1:

glColor3f(0.0, 1.0, 0.0);

break;

case 2:

glColor3f(1.0, 1.0, 0.0);

break;

case 3:

glColor3f(0.0, 1.0, 1.0);

break;

case 4:

glColor3f(1.0, 0.0, 1.0);

break;

case 5:

glColor3f(0.0, 0.0, 0.0);

break;

}

}

void Init(void)

{

const GLfloat mat\_specular[] = { 1.0, 1.0, 1.0, 1.0 };

const GLfloat mat\_shininess[] = { 50.0 };

const GLfloat light\_position[] = { 0.0, 1.0, 1.0, 0.0 };

glShadeModel(GL\_SMOOTH);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL); /\* draw polygons filled \*/

glClearColor(1.0, 1.0, 1.0, 1.0); /\* set screen clear color \*/

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA); /\* blending settings \*/

glCullFace(GL\_BACK); /\* remove backsides \*/

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);

glLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, GL\_TRUE);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_DEPTH\_TEST);

quadric = gluNewQuadric();

gluQuadricNormals(quadric, GLU\_SMOOTH);

}

/\*Is called if the window size is changed \*/

void Reshape(int width, int height)

{

glViewport(0, 0, (GLint)width, (GLint)height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60.0, 1.0, 1.0, 75.0);

glMatrixMode(GL\_MODELVIEW);

}

/\* react to key presses \*/

void Key(unsigned char key, int x, int y)

{

switch (key)

{

case '1':

disks = 1;

hanoiinit();

reset();

offsetY = 0.9;

zoom = -1.3;

gluLookAt(0.0, 0.9, 3.6 + zoom, 0.0, offsetY, 0.0, 0.0, 1.0, 0.0);

reset();

break;

case '2':

disks = 2;

hanoiinit();

reset();

offsetY = 1.1;

zoom = -0.8;

gluLookAt(0.0, 0.9, 3.6 + zoom, 0.0, offsetY, 0.0, 0.0, 1.0, 0.0);

reset();

break;

case '3':

disks = 3;

hanoiinit();

reset();

offsetY = 1.3;

zoom = -0.3;

gluLookAt(0.0, 0.9, 3.6 + zoom, 0.0, offsetY, 0.0, 0.0, 1.0, 0.0);

break;

case '4':

disks = 4;

hanoiinit();

reset();

offsetY = 1.5;

zoom = 0.8;

gluLookAt(0.0, 0.9, 3.6 + zoom, 0.0, offsetY, 0.0, 0.0, 1.0, 0.0);

break;

case '5':

disks = 5;

hanoiinit();

reset();

offsetY = 1.7;

zoom = 1.3;

gluLookAt(0.0, 0.9, 3.6 + zoom, 0.0, offsetY, 0.0, 0.0, 1.0, 0.0);

break;

case '6':

disks = 6;

hanoiinit();

reset();

offsetY = 1.9;

zoom = 1.8;

gluLookAt(0.0, 0.9, 3.6 + zoom, 0.0, offsetY, 0.0, 0.0, 1.0, 0.0);

break;

case '7':

disks = 7;

hanoiinit();

reset();

offsetY = 2.1;

zoom = 2.3;

gluLookAt(0.0, 0.9, 3.6 + zoom, 0.0, offsetY, 0.0, 0.0, 1.0, 0.0);

break;

case '8':

disks = 8;

hanoiinit();

reset();

offsetY = 2.3;

zoom = 2.8;

gluLookAt(0.0, 0.9, 3.6 + zoom, 0.0, offsetY, 0.0, 0.0, 1.0, 0.0);

break;

case '9':

disks = 9;

hanoiinit();

reset();

offsetY = 2.5;

zoom = 3.3;

gluLookAt(0.0, 0.9, 3.6 + zoom, 0.0, offsetY, 0.0, 0.0, 1.0, 0.0);

break;

case 27:

case 'q':

exit(EXIT\_SUCCESS);

break;

case ' ':

rotX = 0.0;

rotY = 0.0;

zoom = 0.0;

offsetY = 1.5;

speed = FSEM \* FEM;

break;

case '+':

zoom -= 0.1;

break;

case '-':

zoom += 0.1;

break;

case 'r':

reset();

break;

case 'f':

if (fullscreen == 0)

{

glutFullScreen();

fullscreen = 1;

}

else

{

glutReshapeWindow(800, 600);

glutPositionWindow(50, 50);

fullscreen = 0;

}

break;

case 's':

speed += 0.005;

break;

case 'x':

speed -= 0.005;

if (speed < 0.0)

speed = 0.0;

break;

}

glutPostRedisplay();

}

void SpecialKey(int key, int x, int y)

{

switch (key)

{

case GLUT\_KEY\_UP:

rotX -= 5;

break;

case GLUT\_KEY\_DOWN:

rotX += 5;

break;

case GLUT\_KEY\_LEFT:

rotY -= 5;

break;

case GLUT\_KEY\_RIGHT:

rotY += 5;

break;

case GLUT\_KEY\_PAGE\_UP:

offsetY -= 0.1;

break;

case GLUT\_KEY\_PAGE\_DOWN:

offsetY += 0.1;

break;

}

glutPostRedisplay();

}

void

mouse(int button, int state, int x, int y)

{

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)

{

zoom += 0.1;

}

if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)

zoom -= 0.1;

if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN)

reset();

}

void Display(void)

{

disk\* cur;

int i;

GLfloat movY;

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); /\* clear scren \*/

glLoadIdentity();

glColor3f(0, 0, 0);

drawBitmapString(25.0, 32.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, seconds);

drawBitmapInt(25.0, 30.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, draw);

drawBitmapInt(28.0, 30.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, maxdraws);

drawBitmapString(-32.0, 32.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "Controls:");

drawBitmapString(-32.0, 30.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "s : Increase Speed");

drawBitmapString(-32.0, 28.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "x : Decrease Speed");

drawBitmapString(-32.0, 26.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "- : Zoom Out");

drawBitmapString(-32.0, 24.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "+ : Zoom In");

drawBitmapString(-32.0, 22.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "f : Fullscreen");

drawBitmapString(-32.0, 20.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "r : Restart Animation");

drawBitmapString(-32.0, 18.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "f : Fullscreen");

drawBitmapString(-32.0, 16.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "q/Esc : Quit");

drawBitmapString(-32.0, 14.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "Use the arrow keys to rotate the view");

drawBitmapString(-32.0, 12.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "Use PageUp/PageDown to move the view up/down");

drawBitmapString(-32.0, 10.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "Press Spacebar to reset the view");

drawBitmapString(-32.0, 8.0, -60.0, GLUT\_BITMAP\_9\_BY\_15, "Press 1 - 9 to select the number of disks");

gluLookAt(0.0, 0.9, 3.6 + zoom, 0.0, offsetY, 0.0, 0.0, 1.0, 0.0); // calculate view point

glRotatef(rotY, 0.0, 1.0, 0.0); /\* rotate Y axis \*/

glRotatef(rotX, 1.0, 0.0, 0.0); /\* rotate X axis \*/

glColor3f(0.0, 0.0, 0.5);

drawAllPins(&quadric, config.pinradius, config.pinheight, config.gap); // draw pins

glTranslatef(-config.gap, BREITE / 2, 0.0);

glPushMatrix();

for (i = 0; i < 3; i++)

{ /\* fill pins with disks \*/

glPushMatrix();

pinheight[i] = 0;

if ((cur = pin[i].bottom) != NULL)

{

do {

setColor(cur->color);

drawDisk(&quadric, cur->radius, STANGENBREITE);

glTranslatef(0.0, BREITE, 0.0);

pinheight[i] += BREITE;

cur = cur->next;

} while (cur != NULL);

}

glPopMatrix();

glTranslatef(config.gap, 0.0, 0.0);

}

glPopMatrix();

if (curaction != NULL && curaction->fromstack != -1 && curdisk != NULL) {

if (pos <= 1.0)

{

movY = pos \* (config.pinheight - pinheight[(int)curaction->fromstack]);

glTranslatef(config.gap \* curaction->fromstack, pinheight[(int)curaction->fromstack] + movY, 0.0);

}

else

{

if (pos < 2.0 && curaction->fromstack != curaction->tostack)

{

if (curaction->fromstack != 1 && curaction->tostack != 1)

{ /\* jump 2 pins \*/

glTranslatef(config.gap, config.pinheight + 0.05f, 0.0);

if (curaction->fromstack == 0)

glRotatef(-(pos - 2.0f) \* 180 - 90, 0.0, 0.0, 1.0);

else

glRotatef((pos - 2.0f) \* 180 + 90, 0.0, 0.0, 1.0);

glTranslatef(0.0, config.gap, 0.0);

}

else

{

if (curaction->fromstack == 0 && curaction->tostack == 1)

{

glTranslatef(config.gap / 2, config.pinheight + 0.05f, 0.0);

glRotatef(-(pos - 2.0f) \* 180 - 90, 0.0, 0.0, 1.0);

}

else

{

if (curaction->fromstack == 2 && curaction->tostack == 1)

{

glTranslatef(config.gap / 2 \* 3, config.pinheight + 0.05f, 0.0);

glRotatef((pos - 2.0f) \* 180 + 90, 0.0, 0.0, 1.0);

}

else

{

if (curaction->fromstack == 1 && curaction->tostack == 2)

{

glTranslatef(config.gap / 2 \* 3, config.pinheight + 0.05f, 0.0);

glRotatef(-(pos - 2.0f) \* 180 - 90, 0.0, 0.0, 1.0);

}

else

{

glTranslatef(config.gap / 2, config.pinheight + 0.05f, 0.0);

glRotatef((pos - 2.0f) \* 180 + 90, 0.0, 0.0, 1.0);

}

}

}

glTranslatef(0.0, config.gap / 2, 0.0);

}

glRotatef(-90, 0.0, 0.0, 1.0);

}

else

if (pos >= 2.0)

{ /\* drop disk down \*/

movY = config.pinheight - (pos - 2.0f + speed) \* (config.pinheight - pinheight[(int)curaction->tostack]);

glTranslatef(config.gap \* curaction->tostack, movY, 0.0);

}

}

setColor(curdisk->color);

drawDisk(&quadric, curdisk->radius, STANGENBREITE);

}

glutSwapBuffers(); /\* swap buffers (double buffering) \*/

}

void moveDisk(int param)

{

if (param == 1)

reset();

if (curaction != NULL)

{

if (pos == 0.0 || pos >= 3.0 - speed)

{ /\* 0--1 -> disk goes upwards, 1--2 "disk in air", 2--3 disk goes downwards\*/

pos = 0.0;

draw++;

push(&pin[(int)curaction->tostack], curdisk);

curaction = curaction->next;

if (curaction != NULL)

curdisk = pop(&pin[(int)curaction->fromstack]);

}

pos += speed;

if (pos > 3.0 - FSEM)

pos = 3.0 - FSEM;

glutTimerFunc((unsigned)FEM, moveDisk, 0);

}

else

{

curdisk = NULL;

glutTimerFunc(5000, moveDisk, 1);

}

glutPostRedisplay();

}

void timer(int param)

{

if (curaction != NULL)

{

printf(seconds, "Time: %ds", ++duration);

}

glutTimerFunc(1000, timer, 0);

}

int main(int argc, char\* argv[])

{

hanoiinit();

atexit(hanoicleanup);

glutInit(&argc, argv);

//command line arguments for setting the number of disks

if (argc > 1)

disks = (int)argv[1];

glutInitWindowPosition(0, 0);

glutInitWindowSize(800, 600);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE);

if (glutCreateWindow("TOWER OF HANOI PUZZLE SIMULATION") == GL\_FALSE)

exit(EXIT\_FAILURE);

Init();

glutReshapeFunc(Reshape);

glutKeyboardFunc(Key);

glutSpecialFunc(SpecialKey);

glutMouseFunc(mouse);

glutDisplayFunc(Display);

glutTimerFunc((unsigned)FEM, moveDisk, 0);

glutTimerFunc(1000, timer, 0);

glutMainLoop();

return EXIT\_SUCCESS;

}

**N-QUEENS SOURCE CODE:**

#include <GL/freeglut.h>

#include <GL/glu.h>

#include<stdio.h>

char chessboard[100][100] = {};

int queens;

float W; float H;

int savingposition[100][2] = { };

void myinit(void)

{

glClearColor(0, 0, 0, 0);

}

bool checking(int row, int column)

{

for (int i = 0; i < queens; i++)

{

for (int j = 0; j < queens; j++)

{

if ((j + i == column + row || column - row == j - i || i == row || j == column) && chessboard[i][j] == 'Q')

return false;

}

}

return true;

}

void removeflag(int row)

{

for (int i = 0; i < queens; i++)

{

chessboard[row][i] = '0';

}

}

void backtracking(void)

{

for (int row = 0; row < queens; row++)

{

for (int column = 0; column < queens; column++)

{

if (checking(row, column) && chessboard[row][column] != '-')

{

chessboard[row][column] = 'Q'; removeflag(row + 1);

savingposition[row][0] = row;

savingposition[row][1] = column;

row++; column = -1;

if (row == queens) break;

}

else if (column == queens - 1 && chessboard[row][column] != 'Q')

{

row--;

column = savingposition[row][1];

printf("%d%d",&row,&column);

chessboard[row][column] = '-';

column = 0;

}

}

}

}

void displaytable(void)

{

for (int i = 0; i < queens; i++)

{

for (int j = 0; j < queens; j++)

{

if (chessboard[i][j] != 'Q')

printf(".");

else

printf("%c",&chessboard[i][j]);

}

}

}

void displayboard()

{

glTranslatef(25, H - 25, 0.0);

glPushMatrix();

for (int i = 0; i < queens; i++)

{

glPushMatrix();

for (int j = 0; j < queens; j++)

{

if ((i + j) % 2 == 0)

glColor3f(1, 1, 1);

else

glColor3f(0, 0,0);

glPushMatrix();

glScalef(50.0, 50.0, 0.0);

glutSolidCube(1);

if (chessboard[i][j] == 'Q')

{

glPushMatrix();

glColor3f(0, 0, 1);

glutSolidSphere(0.4, 50, 50);

glPopMatrix();

}

glPopMatrix();

glTranslatef(50.0, 0.0, 0.0);

}

glPopMatrix();

glTranslatef(0.0, -50.0, 0.0);

}

glPopMatrix();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glPushMatrix();

displayboard();

glPopMatrix();

displaytable();

glutSwapBuffers();

}

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, W, 0, H);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();//important: last line must be glLoadIdentity() for coloring with openGL

//glTranslatef (.0, .0, -5.); //do not translate at all if glLoadIdentity() is used

}

void mouse(int button, int state, int x, int y) //mouse function...

{

int row, column;

switch (button)

{

case GLUT\_LEFT\_BUTTON:

if (state == GLUT\_DOWN)

{

row = (y / 50);

column = (x / 50);

if (checking(row, column))

chessboard[row][column] = 'Q';

}

printf("X ", x);

printf("Y ", y);

glutPostRedisplay();

break;

case GLUT\_RIGHT\_BUTTON:

if (state == GLUT\_DOWN)

{

row = (y / 50);

column = (x / 50);

chessboard[row][column] = ' ';

}

printf("X ",x);

printf("Y ", y);

glutPostRedisplay();

break;

}

}

void main(int argc, char\*\* argv)

{

int input;

printf("WELCOME TO N-QUEEN PROBLEM !!!!\n");

printf("1. Show The Solution by Computer!!\n");

printf("2. Try to solve the puzzle\n");

printf("3. Exit!!!\n");

printf("Select the NUMBER from 1 - 3 :\n");

scanf\_s("%d", &input);

switch (input)

{

case 1:

printf("Please enter the number of the queens 1 to 100:\n");

scanf\_s("%d", &queens);

backtracking();

break;

case 2:

printf("What size of N queens do you want to play ?\n");

scanf\_s("%d", &queens);

printf("Left Cick on the box to select!!\n Right click to deselect \n");

break;

case 3:

printf("THANK YOU FOR PLAYING\n");

break;

default:

printf("WRONG INPUT\n");

break;

}

W = 50 \* queens;

H = 50 \* queens;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(W, H);

glutInitWindowPosition(100,100);

glutCreateWindow("N-QUEEN PROBLEM");

myinit();

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutReshapeFunc(reshape);

glutMainLoop();

}

**CHAPTER 6**

**PROJECT SNAPSHOTS**

**TOWER OF HANOI SNAPSHOTS:**

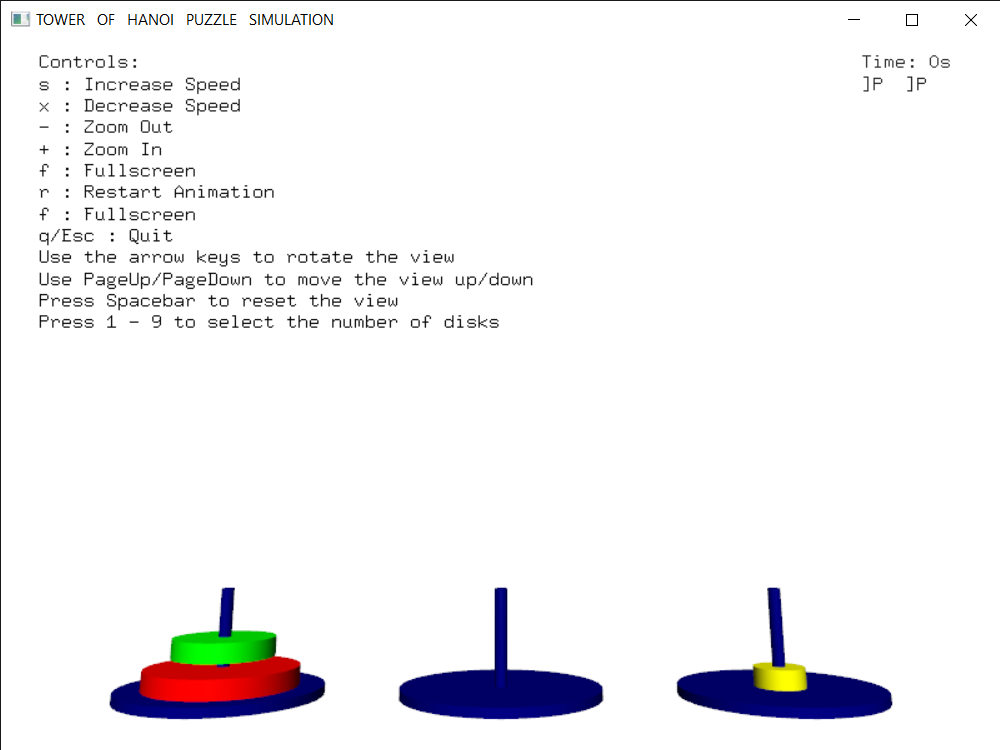


Fig 1.1 Transformation of First Disk from A rod to the C rod.

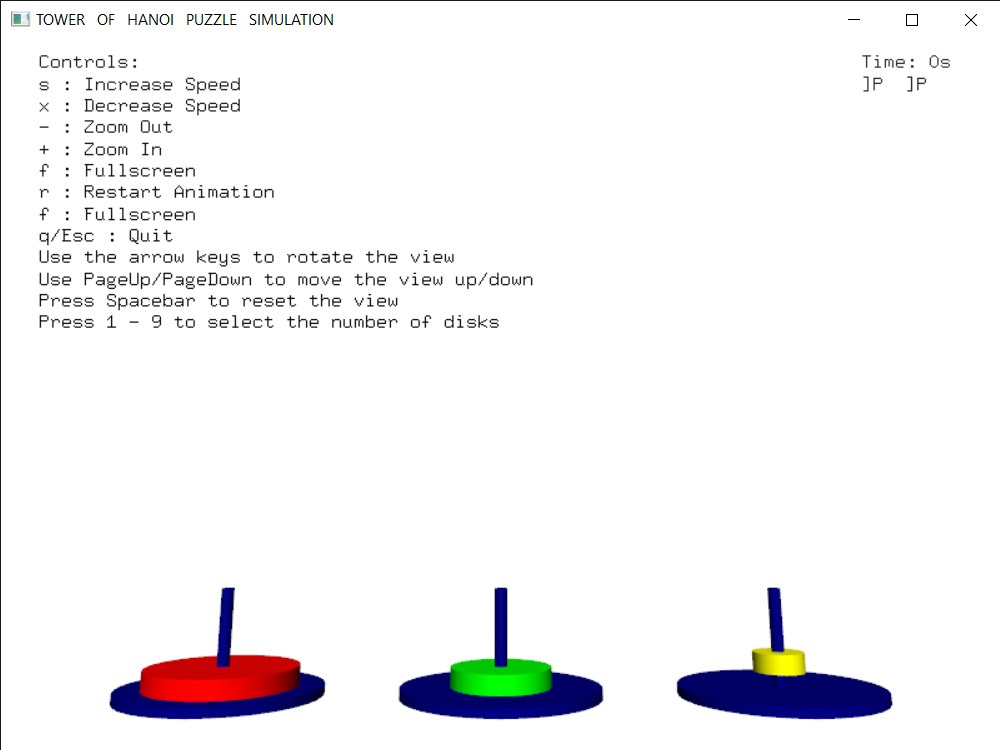


Fig 1.2 Transformation of Second Disk from A rod to the B rod.

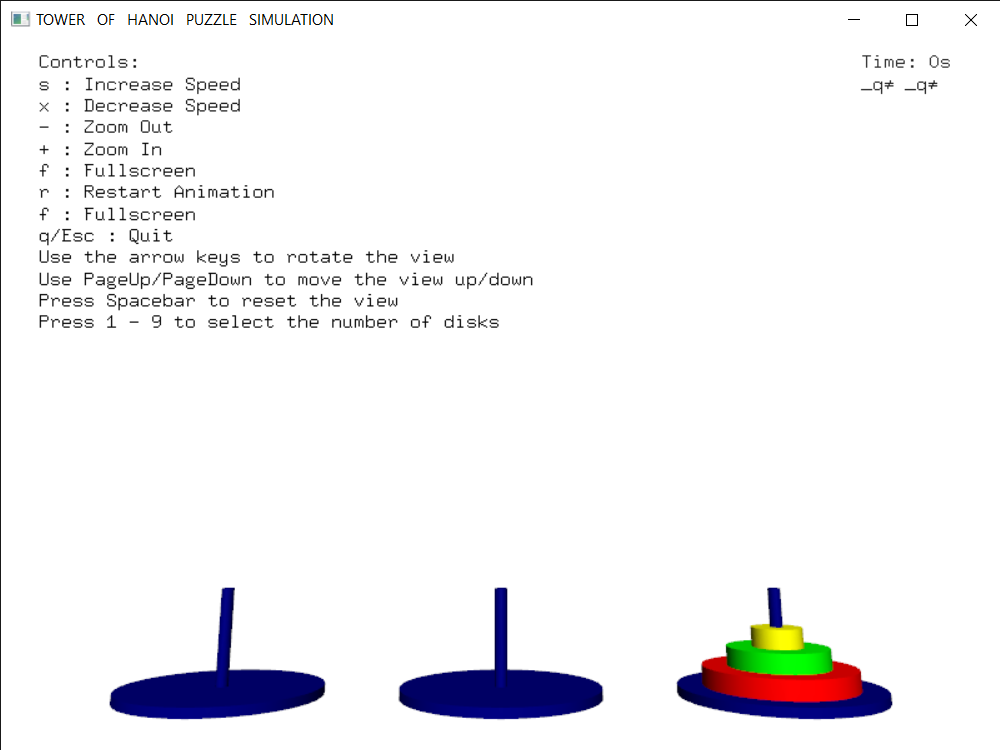


Fig 1.3 Transformation of Third Disk from A rod to the C rod.

**N-QUEENS PROBLEM SNAPSHOTS:**

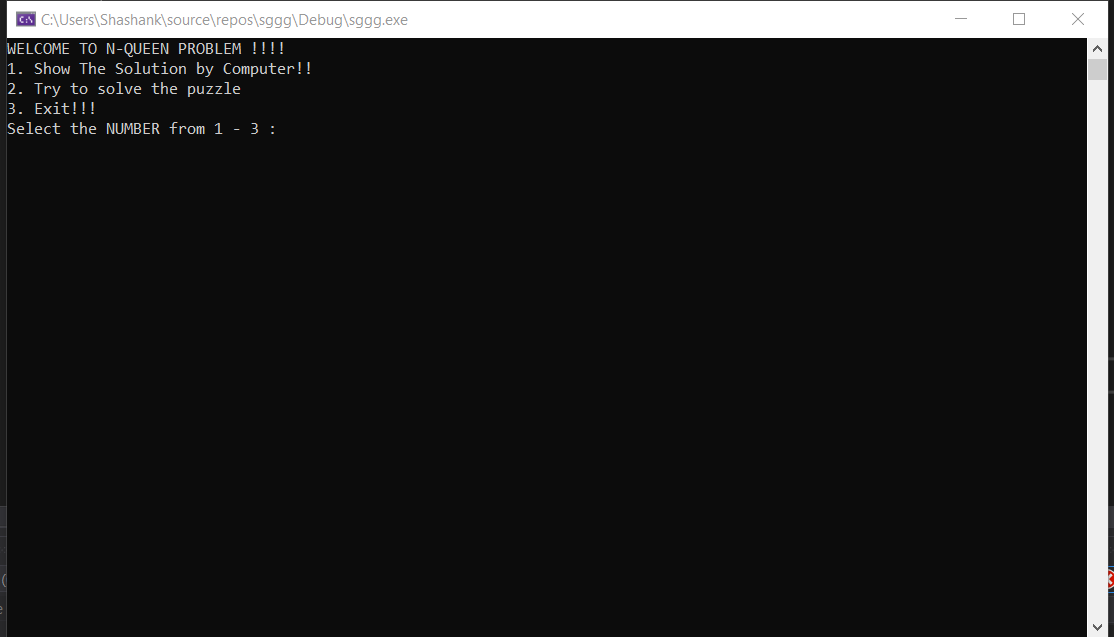


Fig 2.1: Menu to choose how many number of Queens do you want to play .

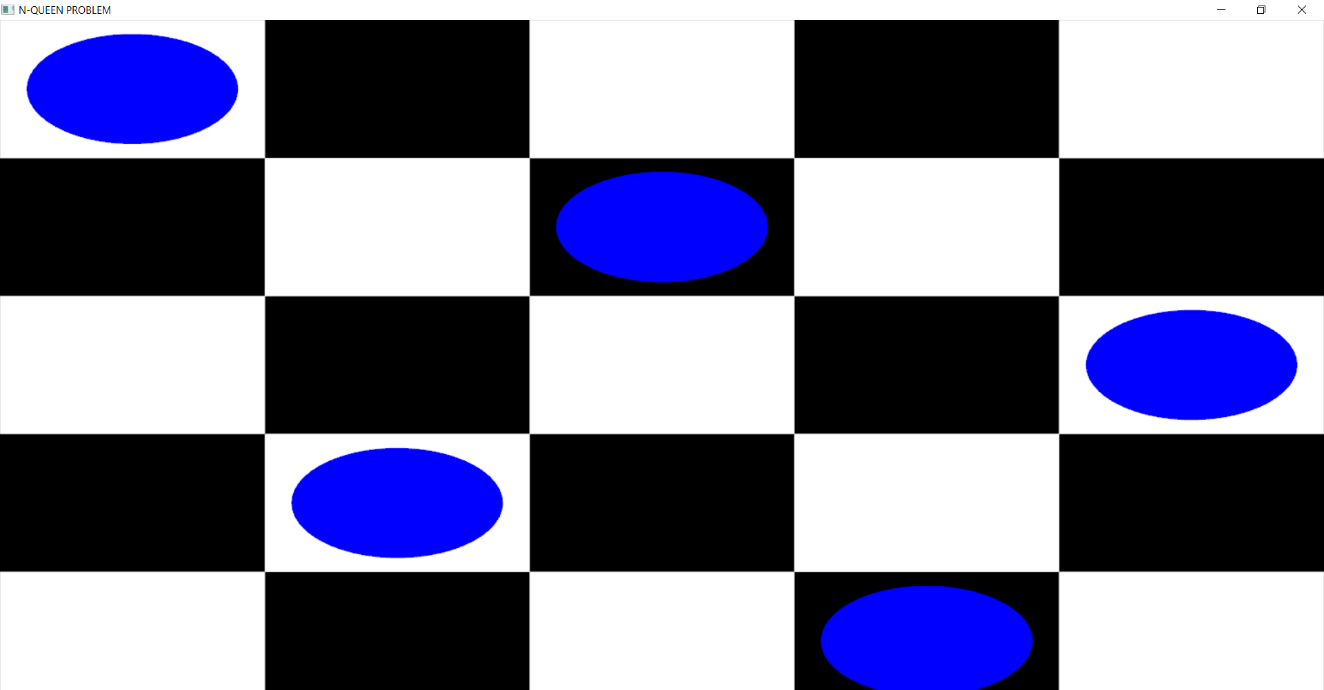


Fig2.2: The final result of N-Queen fully solved.

**CHAPTER 7**

**CONCLUSION**

Perhaps the dominant characteristics of this millennium is how computer and communication technologies have become dominant forces in our lives. Activities as wide ranging as filmmaking , publishing ,banking and education continue to undergo revolutionary changes as these technologies alter the ways in which we conduct out daily activities. The combination of computers, networks and the complex human visual system through computer graphics has led to new ways of displaying information, seeing virtual worlds and communicating with people and machines since the project developed is mainly academics based, the project consists of only a few functions and implementations of few concepts.

This mini project is developed using openGL software ad so far has proved to be having good performance. The user can easily understand the mini project. Over all, the mini project was good experience and provides knowledge of working on OpenGL.

**REFERENCES**

[1] Edward Angel, Interactive Computer Graphics, A Top-Down Approach Using OpenGL, 5th edition

[2] OpenGL Architecture Review Board, Dave Shreiner, Opengl® Reference Manual,4th edition

[3] OpenGL ® SuperBible: Comprehensive Tutorial and Reference, by [Richard S. Wright](http://www.amazon.com/exec/obidos/search-handle-url/ref=ntt_athr_dp_sr_1?%5Fencoding=UTF8&search-type=ss&index=books&field-author=Richard%20S.%20Wright) (Author), [Benjamin Lipchak](http://www.amazon.com/exec/obidos/search-handle-url/ref=ntt_athr_dp_sr_2?%5Fencoding=UTF8&search-type=ss&index=books&field-author=Benjamin%20Lipchak) (Author), [Nicholas Haemel](http://www.amazon.com/exec/obidos/search-handle-url/ref=ntt_athr_dp_sr_3?%5Fencoding=UTF8&search-type=ss&index=books&field-author=Nicholas%20Haemel) (Author), 4th edition

[4] www.wikipedia.org

[5] www.stackoverflow.com