**CHAPTER 1**

**INTRODUCTION**

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

* The representation and manipulation of image data by a computer
* The various technologies used to create and manipulate images
* The images so produced appears to be real photographic.
* The subfield of computer science which studies methods for digitally synthesizing and manipulating visual content.

Today,computers and computer generated images touch many aspects of daillife.Computer imagry is found on television,in newspapers,for example in weather reports,or for example in all kinds of medical investigation and surgical procedures.A well-constructed graph can present complex statistics in a form that is easier to understand and interpret.In the media “such graphs are used to illustrate papers,reports” ,and other presentation material.

Many powerful tools have been developed to visualize data.Computer generated imagery can be categorized into several different types:2D,3D and animated graphics.As technology has improved 3D computer graphics have become more common,but 2D computer graphics are still widely used.Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.Over the past decade,other specialized fields have been developed like information visualization and scientific visualization more concerned with “the visualization of dimensional phenomen, where the emphasis is on realistic renderings of volumes,surfaces illumination sources and so forth perhaps with a dynamic component”.

**1.1 Concepts and Principles**

* **Image**

An image or picture is an artifact that resembles a physical object or person.The term includes two-dimensional objects like photographs and sometimes includes three-dimensional representations.Images are captured by optical devices—such as cameras mirrors lenses telescopes microscopes etc and natural objects and phenomena such as the human eye or water surfaces.

* **Pixel**

In the enlarged portion of the image individual pixels are rendered as squares and can be easily seen.In digital imaging a pixel(or a picture element) is a single point in a raster image.Pixels are normally arranged in a regular 2-dimansional grid and are often represented using dots or squares.

* **Graphics**

Graphics are visual presentations on some surface,such as a wall,canvas,computer screen paper or stone to brand inform illustrate or entertain.Examples are photographs drawings lineart graphs diagrams geometric designs,maps,engineering drawings or other images.Graphics often combine text,illustration and color.

* **Rendering**

Rendering is the process of generating an image from a model by means of computer programs.It would contain geometry viewpoint texture lighting and shading information.The image is a digital image or raster graphics image.

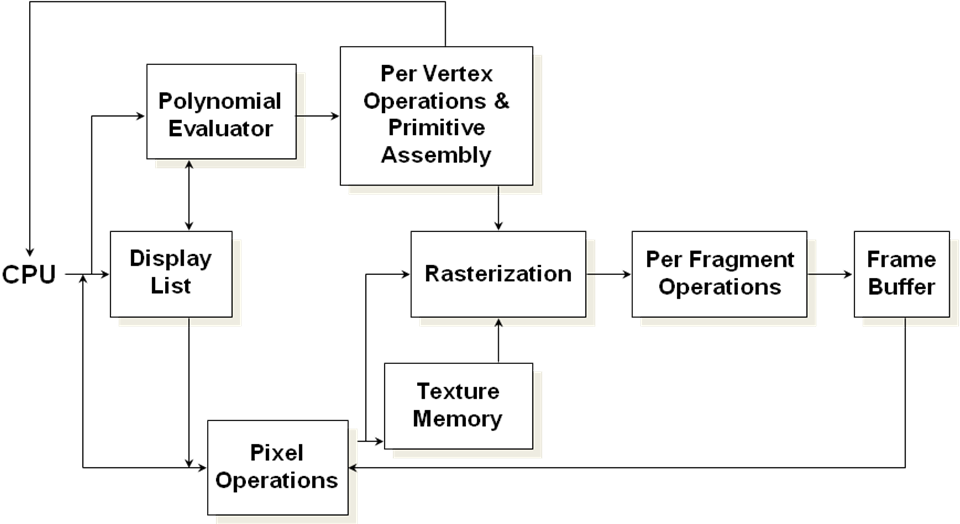
* **3D modeling**

3D modeling is the mathematical wireframe representation of any three dimensional object called a “3D model” via specialized software.Models may be created automatically or manually.

**1.2 Application of Computer Graphics**

* Display of information
* Simulations
* User Interface
* Design

**1.3 OpenGL Architecture**



**Fig 1.1 OpenGL Architecture**

This is the most important diagram you will see today, representing the flow of graphical information, as it is processed from CPU to the frame buffer.

There are two pipelines of data flow. The upper pipeline is for geometric, vertex-based primitives. The lower pipeline is for pixel-based, image primitives. Texturing combines the two types of primitives together.

**CHAPTER 2**

**SYSTEM REQUIREMENTS**

## Software:

* Operating system: Windows XP professional
* Language used: OpenGL/C++ programming language
* Compiler: Microsoft visual C++ 6.0

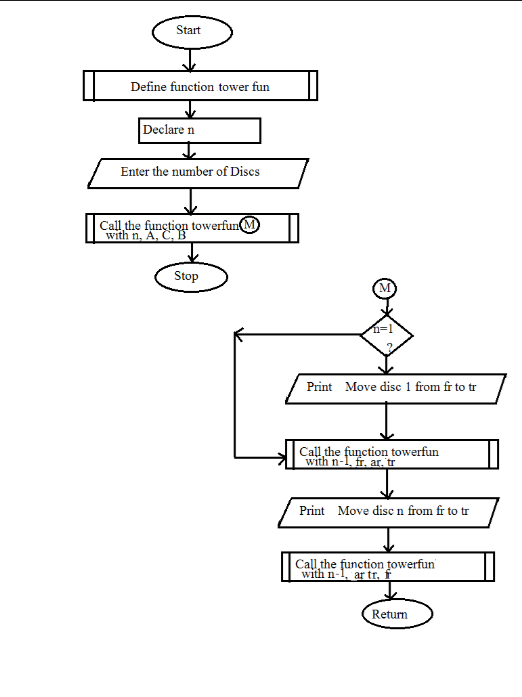
## Hardware:

* Monitor: Colour monitor (LCD/CRT)
* Keyboard: standard keyboard (104 keys)
* Mouse: optical mouse
* Processor: Intel pentium4 and above
* Hard disk: 5GB
* RAM: 128MB

**CHAPTER 3**

**SYSTEM DESIGN**

**3.1 Flow Diagram**

****

**Fig 1.2 Flow Diagram of Tower of Hanoi**

**CHAPTER 4**

**IMPLEMENTATION**

Implementation is the process of converting a new system design into an operationalSystem. It is the key stage in achieving a successful new system.It must therefore be carefully planned and controlled.The implementation of a system is done after theDevelopment effort is completed. The system is implemented using Microsoft visual

C++ 6.0 language. Steps for implementation:

* Installation of hardware and software utilities.
* Testing the developed system with sampled data.
* Detection and correction of errors. Data updating.

**4.1Functions Description**

**1. glLight():** Specifies a light. The number of lights depends on the implementation, but at least eight lights are supported.They are identified by symbolic names of theform GL\_LIGHT i, where i ranges from 0 to the value of GL\_MAX\_LIGHTS - 1.

**2. glMaterial()**: assigns values to material parameters.There are two matched sets ofmaterial parameters. One, the front-facingset, is used to shade points, lines, bitmaps, and all polygons (when two-sided lighting is disabled), or just front-facing polygons(when two-sided lighting is enabled). The other set back-facing is used to shade back-facing polygons only when two-sided lighting is enabled. Refer to the glLightModel reference page for details concerning one- and two-sided lightingcalculations.

**3. glShadeModel():** GL primitives can have either flat or smooth shading. Smooth shading, the default causes the computed colors of vertices to be interpolated as theprimitive is rasterized typically assigning different colors to each resulting pixelfragment. Flat shading selects the computed Color of just one vertex and assigns it toall the pixel fragments generated by rasterizing a single primitive.

**4. glPushMatrix():** There is a stack of matrices for each of the matrix modes. In GL\_MODELVIEW mode, the stack depth is at least 32. In the othermodes, GL\_COLOR, GL\_PROJECTION and GL\_TEXTURE, the depth is at least 2.The current matrix in any mode is the matrix on the top of the stack for that mode.

**5. glTranslate():** produces a translation by x y z . The current matrix(see glMatrixMode) is multiplied by this translation matrix with the product replacingthe current matrix.

**6. glSolidSphere():** Renders a sphere centered at the modeling coordinates origin ofthe specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

**7. glPopMatrix():** There is a stack of matrices for each of the matrix modes. In GL\_MODELVIEW mode, the stack depth is at least 32. The current matrix in anymode is the matrix on the top of the stack for that mode.

**8.Void glClearColor(int red, int green, int blue, int alpha);**

This function is used to clear the color of the screen. The 4 values that are passed as arguments for this function are (RED, GREEN, BLUE, ALPHA) where the red green and blue components are taken to set the background color and alpha is a value that specifies depth of the window. It is used for 3D images.

**9.gluCylinder Function**

The gluCylinder function draws a cylinder.

Syntax

void WINAPI gluCylinder(GLUquadric \*qobj,GLdouble baseRadius,GLdouble topRadius, GLdouble height, GLintslices, GLint stacks);

Parameters

* qobj The quadric object (created with gluNewQuadric).
* baseRadius The radius of the cylinder at z = 0.
* topRadius The radius of the cylinder at z = height.
* height The height of the cylinder.
* slices The number of subdivisions around the z-axis.
* stacks The number of subdivisions along the z-axis.

Return value

**10.gluQuadricDrawStyle Function**

The gluQuadricDrawStyle function specifies the draw style desired for quadrics.

Syntax

void WINAPI gluQuadricDrawStyle(GLUquadric \*quadObject, GLenum drawStyle);

Parameters

* quadObject
  + - The quadric object (created with gluNewQuadric).
* drawStyle
  + - The desired draw style. The following values are valid.
* Value Meaning
* GLU\_FILL: Quadrics are rendered with polygon primitives. The polygons are drawn in a counterclockwise fashion with respect to their normals (as defined with gluQuadricOrientation).
* GLU\_LINE: Quadrics are rendered as a set of lines.
* GLU\_SILHOUETTE: Quadrics are rendered as a set of lines, except that edges separating coplanar faces will not be drawn.
* GLU\_POINT : quadrics are rendered as a set of points.

**11.gluNewQuadric Function**

The **gluNewQuadric** function creates a quadric object.

**Syntax**

GLUquadric\* WINAPI gluNewQuadric(void);

**Parameters**

This function has no parameters.

**Remarks**

The **gluNewQuadric** function creates and returns a pointer to a new quadric object. Refer to this object when calling quadric rendering and control functions. A return value of zero means there is not enough memory to allocate to the object

**DESCRIPTION**

The primary requirements for this program are to declare constants for the viewport such as screen width (*WIDTH*) and height (*HEIGHT*). The number of disks/rings for which the program is to be solved is declared in a global variable *NUM\_DISKS*. By default this is set to 20. Other global constants defined are for a disk’s height (*DISK\_HEIGHT*), and other run-time options such as Solve (*HANOI\_SOLVE*), Exit (*HANOI\_QUIT*), Lighting settings (*HANOI\_LIGHTING*), Back-Wall settings (*HANOI\_WALL*) and Fog (*HANOI\_FOG*). Besides these, several OpenGL integral constants for rotation (*xangle, yangle, xlangle, ylangle*) and Boolean constants (*motion, back\_wall*) are defined. For the purpose of lighting, we require direction vectors and position vectors for every light source. Also we need to set the ambient lighting.

After these constants, structures *stack* and *stack\_node* are defined. The *stack* structure stores the disks in the three towers, whereas the *stack\_node* structure links the topmost disk of every tower to other towers (*stack*s). As we use a stack data structure, we define the *push* and *pop* functions for the movement of disks.

The next function we define is *init*. In this function, we create the stacks and set the tail pointers to null. Then we display the three cones for the towers (from the *DrawPosts* function), before which the rings are made by the solid geometric structure call a Torus (from the function *DrawDisks)*. After this, the draw function is called. As the program depends on the user’s choices at runtime, we also declare some exit conditions. A user can exit the program at any time by pressing any of the following keys: ‘Q’, ‘q’, ‘Esc’. These are passed to the function *keyboard*. The user can choose to implement other features such as switching on/off the lighting, fogging the display, displaying the backwall and exiting by right-clicking on the screen and selecting the desired option. All this is handled by the function *hanoi\_menu*, which uses a switch statement to perform the selected option.

For solving, we use the recursive procedure for the Towers of Hanoi problem. This is implemented in the recursive function *mov*. After every movement the entire screen is redrawn in the method *update*, which calls the glutPostRedisplay() function. This translates the topmost disk from one tower to another.

The rotation can be performed by the user during run-time by using the mouse. For this, the user is required to left-click on the screen. The program then finds the location where the left mouse button was pressed down and passes the coordinates to the function *hanoi\_mouse*. Then it continuously tracks the motion of the mouse’s movements and using these values, rotates the entire set of towers and disks about the origin, which is the screen’s center.

The *main* function calls the *glutInit* function to create the window and all the elements such as display-mode (RGBA), enabling the lighting settings, creating the menu, etc. Then the initial call to the function *mov* is called, which starts the execution of the program.

**SOURCE CODE**

#include <stdlib.h>

#include <GL/glut.h>

#include <stdio.h>

#include <string.h>

double WIDTH = 1360;

double HEIGHT = 768;

int NUM\_DISKS = 20;

GLboolean motion = GL\_FALSE;

GLboolean back\_wall = GL\_FALSE;

GLint xangle = 0, yangle = 0;

GLint xlangle = 0, ylangle = 0;

#define other(i,j) (6-(i+j))

#define wallz -(WIDTH/2)

#define DISK\_HEIGHT 20

#define CONE NUM\_DISKS+1

#define WALL CONE + 1

#define HANOI\_SOLVE 0

#define HANOI\_QUIT 1

#define HANOI\_LIGHTING 2

#define HANOI\_WALL 3

#define HANOI\_FOG 4

GLfloat lightOneDirection[] = {-0.1, 0, -1};

GLfloat lightOnePosition[] = {450, 100, 1500, 1};

GLfloat lightOneColor[] = {1.0, 0.5, 0, 1.0};

GLfloat lightTwoDirection[] = {0, 0.05, -1};

GLfloat lightTwoPosition[] = {900, 100, 1500, 1};

GLfloat lightTwoColor[] = {1.0, 0.0, 1, 1.0};

GLfloat lightZeroPosition[] = {400, 200, 300, 1};

GLfloat lightZeroColor[] = {.3, .3, .3, .3};

GLfloat diskColor[] = {1.0, 0.5, 0.5, .1}, poleColor[] = {0.5, 0.5, 0.5, 1.0};

//Structures for stack

typedef struct stack\_node

{

int size;

struct stack\_node \*next;

} stack\_node;

typedef struct stack

{

struct stack\_node \*head;

int depth;

} stack;

stack poles[4];

void push(int which, int size)

{

stack\_node \*new = malloc(sizeof(stack\_node));

if (!new)

{

fprintf(stderr, "out of memory!\n");

exit(-1);

}

new->size = size;

new->next = poles[which].head;

poles[which].head = new;

poles[which].***depth***++;

}

int pop(int which)

{

int retval = poles[which].head->size;

stack\_node \*temp = poles[which].head;

poles[which].head = poles[which].head->next;

poles[which].depth--;

free(temp);

return retval;

}

typedef struct move\_node

{

int t, f;

struct move\_node \*next;

struct move\_node \*prev;

} move\_node;

typedef struct move\_stack

{

int depth;

struct move\_node \*head, \*tail;

} move\_stack;

move\_stack moves;

//Initialize towers and rings

void init(void)

{

int i;

for (i = 0; i < 4; i++)

{

poles[i].head = NULL;

poles[i].depth = 0;

}

moves.head = NULL;

moves.tail = NULL;

moves.depth = 0;

for (i = 1; i <= NUM\_DISKS; i++)

{

glNewList(i, GL\_COMPILE);

{

glutSolidTorus(DISK\_HEIGHT / 2, 5 \* i, 15, 15);

}

glEndList();

}

glNewList(CONE, GL\_COMPILE);

{

glutSolidCone(10, (NUM\_DISKS + 1) \* DISK\_HEIGHT, 20, 20);

}

glEndList();

}

void mpop(void)

{

move\_node \*temp = moves.head;

moves.head = moves.head->next;

free(temp);

moves.depth--;

}

void mpush(int t, int f)

{

move\_node \*new = malloc(sizeof(move\_node));

if (!new)

{

fprintf(stderr, "Out of memory!\n");

exit(-1);

}

new->t = t;

new->f = f;

new->next = NULL;

new->prev = moves.tail;

if (moves.tail)

moves.tail->next = new;

moves.tail = new;

if (!moves.head)

moves.head = moves.tail;

moves.depth++;

}

//Exit conditions

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

{

switch (key)

{

case 27: /\* ESC \*/

case 'q':

case 'Q':

exit(0);

}

}

void update(void)

{

glutPostRedisplay();

}

void DrawPost(int xcenter)

{

glPushMatrix();

{

glTranslatef(xcenter, 0, 0);

glRotatef(90, -1, 0, 0);

glCallList(CONE);

}

glPopMatrix();

}

void DrawPosts(void)

{

glColor3fv(poleColor);

glLineWidth(10);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, poleColor);

DrawPost((int)(WIDTH / 4));

DrawPost((int)(2 \* WIDTH / 4));

DrawPost((int)(3 \* WIDTH / 4));

}

void DrawDisk(int xcenter, int ycenter, int size)

{

glPushMatrix();

{

glTranslatef(xcenter, ycenter, 0);

glRotatef(90, 1, 0, 0);

glCallList(size);

} glPopMatrix();

}

void DrawDooDads(void)

{

int i;

stack\_node \*temp;

int xcenter, ycenter;

glColor3fv(diskColor);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diskColor);

for (i = 1; i <= 3; i++)

{

xcenter = i \* WIDTH / 4;

for (temp = poles[i].head, ycenter = DISK\_HEIGHT \* poles[i].depth - DISK\_HEIGHT / 2; temp; temp = temp->next, ycenter -= DISK\_HEIGHT) {

DrawDisk(xcenter, ycenter, temp->size);

}

}

}

#define MOVE(t,f) mpush((t),(f))

static void mov(int n, int f, int t)

{

int o;

if (n == 1)

{

MOVE(t, f);

return;

}

o = other(f, t);

mov(n - 1, f, o);

mov(1, f, t);

mov(n - 1, o, t);

}

GLfloat wallcolor[] = {0, 0.5, 0.5, .1};

void DrawWall(void)

{

int i, j;

glColor3fv(wallcolor);

for (i = 0; i < WIDTH; i += 10)

{

for (j = 0; j < HEIGHT; j += 10)

{

glBegin(GL\_POLYGON);

{

glNormal3f(0, 0, 1);

glVertex3f(i + 10, j, wallz);

glVertex3f(i + 10, j + 10, wallz);

glVertex3f(i, j + 10, wallz);

glVertex3f(i, j, wallz);

}

glEnd();

}

}

}

void draw(void)

{

int t, f;

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

if (back\_wall)

{

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, wallcolor);

DrawWall();

}

glPushMatrix();

{

glTranslatef(WIDTH / 2, HEIGHT / 2, 0);

glRotatef(xlangle, 0, 1, 0);

glRotatef(ylangle, 1, 0, 0);

glTranslatef(-WIDTH / 2, -HEIGHT / 2, 0);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightZeroPosition);

}

glPopMatrix();

glPushMatrix();

{

glTranslatef(WIDTH / 2, HEIGHT / 2, 0);

glRotatef(xangle, 0, 1, 0);

glRotatef(yangle, 1, 0, 0);

glTranslatef(-WIDTH / 2, -HEIGHT / 2, 0);

DrawPosts();

DrawDooDads();

}

glPopMatrix();

if (motion && moves.depth) {

t = moves.head->t;

f = moves.head->f;

push(t, pop(f));

mpop();

}

glutSwapBuffers();

}

void hanoi\_menu(int value)

{

switch (value)

{

case HANOI\_SOLVE:

motion = !motion;

if(motion)

{

glutIdleFunc(update);

}

else

{

glutIdleFunc(NULL);

}

break;

case HANOI\_LIGHTING:

if (glIsEnabled(GL\_LIGHTING))

glDisable(GL\_LIGHTING);

else

glEnable(GL\_LIGHTING);

break;

case HANOI\_WALL:

back\_wall = !back\_wall;

break;

case HANOI\_FOG:

if (glIsEnabled(GL\_FOG))

glDisable(GL\_FOG);

else {

glEnable(GL\_FOG);

glFogi(GL\_FOG\_MODE, GL\_EXP);

glFogf(GL\_FOG\_DENSITY, .01);

}

break;

case HANOI\_QUIT:

exit(0);

break;

}

glutPostRedisplay();

}

int oldx, oldy;

GLboolean leftb = GL\_FALSE, middleb = GL\_FALSE;

void hanoi\_mouse(int button, int state, int x, int y)

{

if (button == GLUT\_LEFT\_BUTTON)

{

oldx = x;

oldy = y;

if (state == GLUT\_DOWN)

leftb = GL\_TRUE;

else

leftb = GL\_FALSE;

}

if (button == GLUT\_MIDDLE\_BUTTON)

{

oldx = x;

oldy = y;

if (state == GLUT\_DOWN)

middleb = GL\_TRUE;

else

middleb = GL\_FALSE;

}

}

void hanoi\_visibility(int state)

{

if (state == GLUT\_VISIBLE && motion)

{

glutIdleFunc(update);

}else

{

glutIdleFunc(NULL);

}

}

void hanoi\_motion(int x, int y)

{

if (leftb) {

xangle -= (x - oldx);

yangle -= (y - oldy);

}

if (middleb) {

xlangle -= (x - oldx);

ylangle -= (y - oldy);

}

oldx = x;

oldy = y;

glutPostRedisplay();

}

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

{

int i;

glutInit(&argc, argv);

for(i=1; i<argc; i++)

{

if(!strcmp("-n", argv[i]))

{

i++;

if(i >= argc)

{

printf("hanoi: number after -n is required\n");

exit(1);

}

NUM\_DISKS = atoi(argv[i]);

}

}

glutInitWindowSize((int)WIDTH, (int)HEIGHT);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE | GLUT\_DEPTH);

glutCreateWindow("Hanoi");

glutDisplayFunc(draw);

glutKeyboardFunc(keyboard);

glViewport(0, 0, (int)WIDTH, (int)HEIGHT);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0, WIDTH, 0, HEIGHT, -10000, 10000);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glClearColor(0, 0, 0, 0);

glClearDepth(1.0);

glEnable(GL\_CULL\_FACE);

glEnable(GL\_DEPTH\_TEST);

glLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, 1);

glLightfv(GL\_LIGHT1, GL\_POSITION, lightOnePosition);

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, lightOneColor);

glLightf(GL\_LIGHT1, GL\_SPOT\_CUTOFF, 10);

glLightfv(GL\_LIGHT1, GL\_SPOT\_DIRECTION, lightOneDirection);

glEnable(GL\_LIGHT1);

glLightfv(GL\_LIGHT2, GL\_POSITION, lightTwoPosition);

glLightfv(GL\_LIGHT2, GL\_DIFFUSE, lightTwoColor);

//glLightf(GL\_LIGHT2,GL\_LINEAR\_ATTENUATION,.005);

glLightf(GL\_LIGHT2, GL\_SPOT\_CUTOFF, 10);

glLightfv(GL\_LIGHT2, GL\_SPOT\_DIRECTION, lightTwoDirection);

glEnable(GL\_LIGHT2);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightZeroColor);

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHTING);

glutMouseFunc(hanoi\_mouse);

glutMotionFunc(hanoi\_motion);

glutVisibilityFunc(hanoi\_visibility);

glutCreateMenu(hanoi\_menu);

glutAddMenuEntry("Solve", HANOI\_SOLVE);

glutAddMenuEntry("Lighting", HANOI\_LIGHTING);

glutAddMenuEntry("Back Wall", HANOI\_WALL);

glutAddMenuEntry("Fog", HANOI\_FOG);

glutAddMenuEntry("Quit", HANOI\_QUIT);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

init();

for (i = 0; i < NUM\_DISKS; i++)

push(1, NUM\_DISKS - i);

mov(NUM\_DISKS, 1, 3);

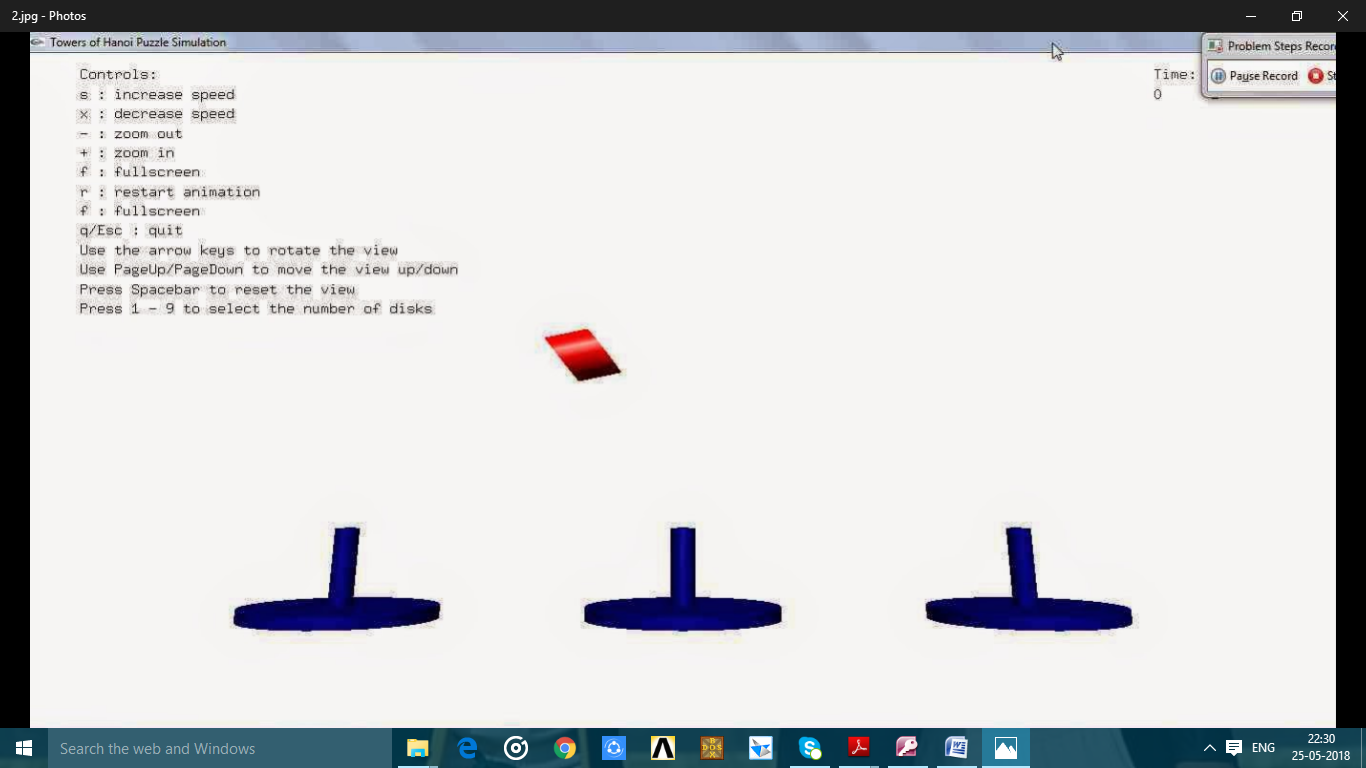
glutMainLoop();

return 0;

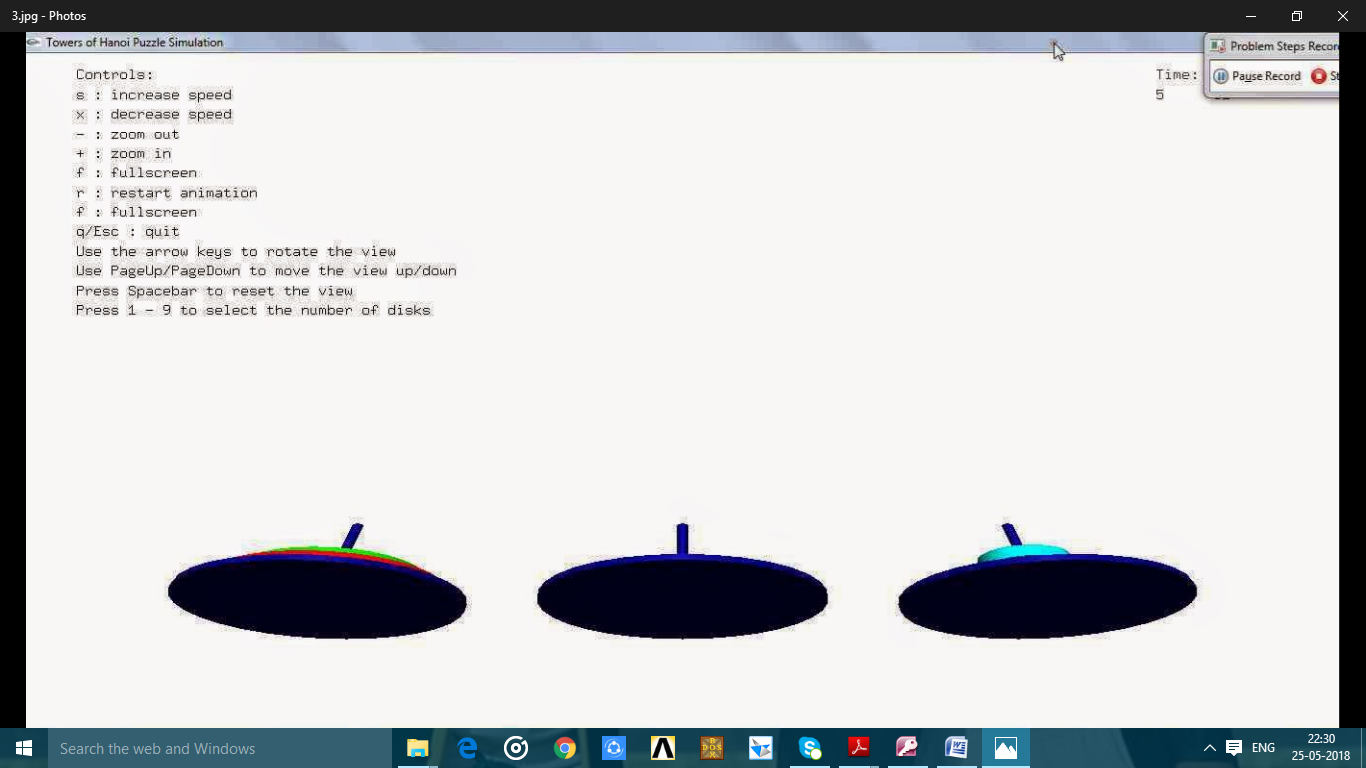
}

**CHAPTER 5**

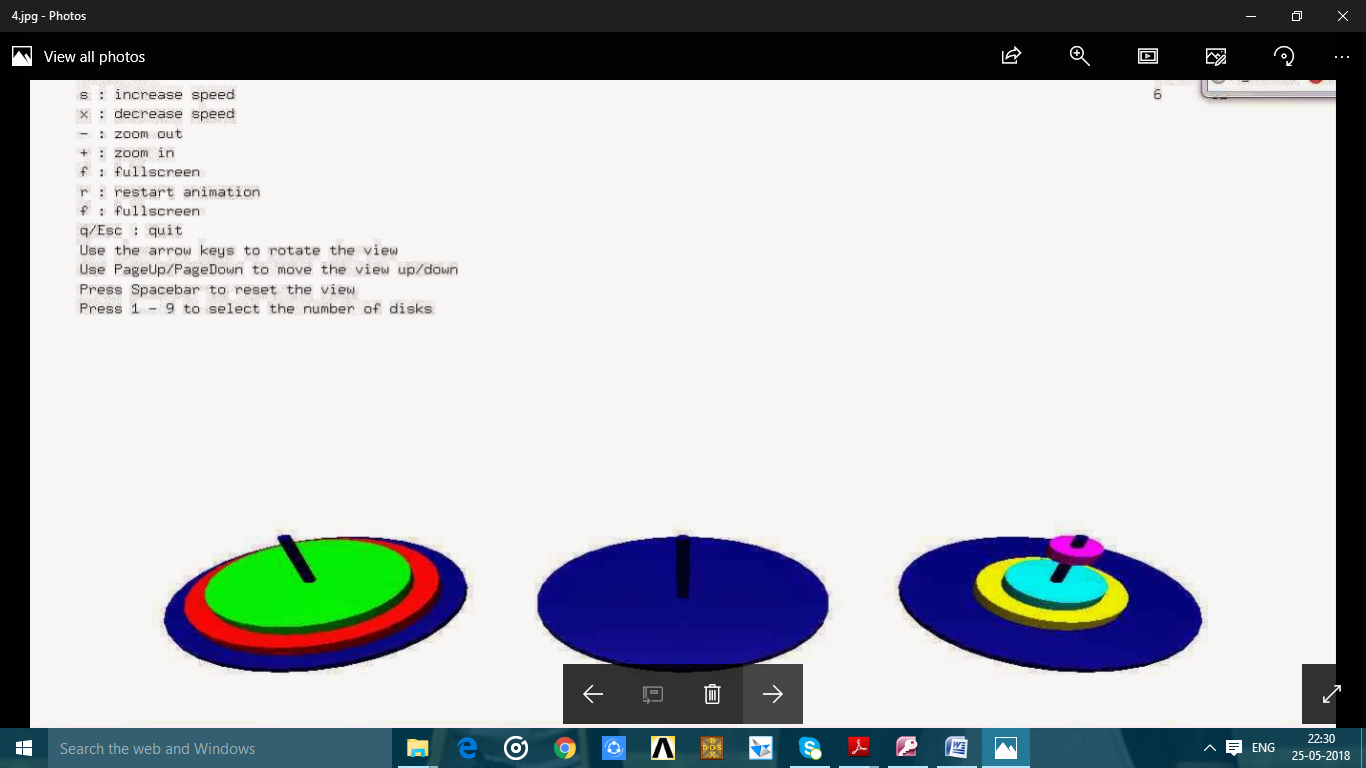
**SNAPSHOTS**

-

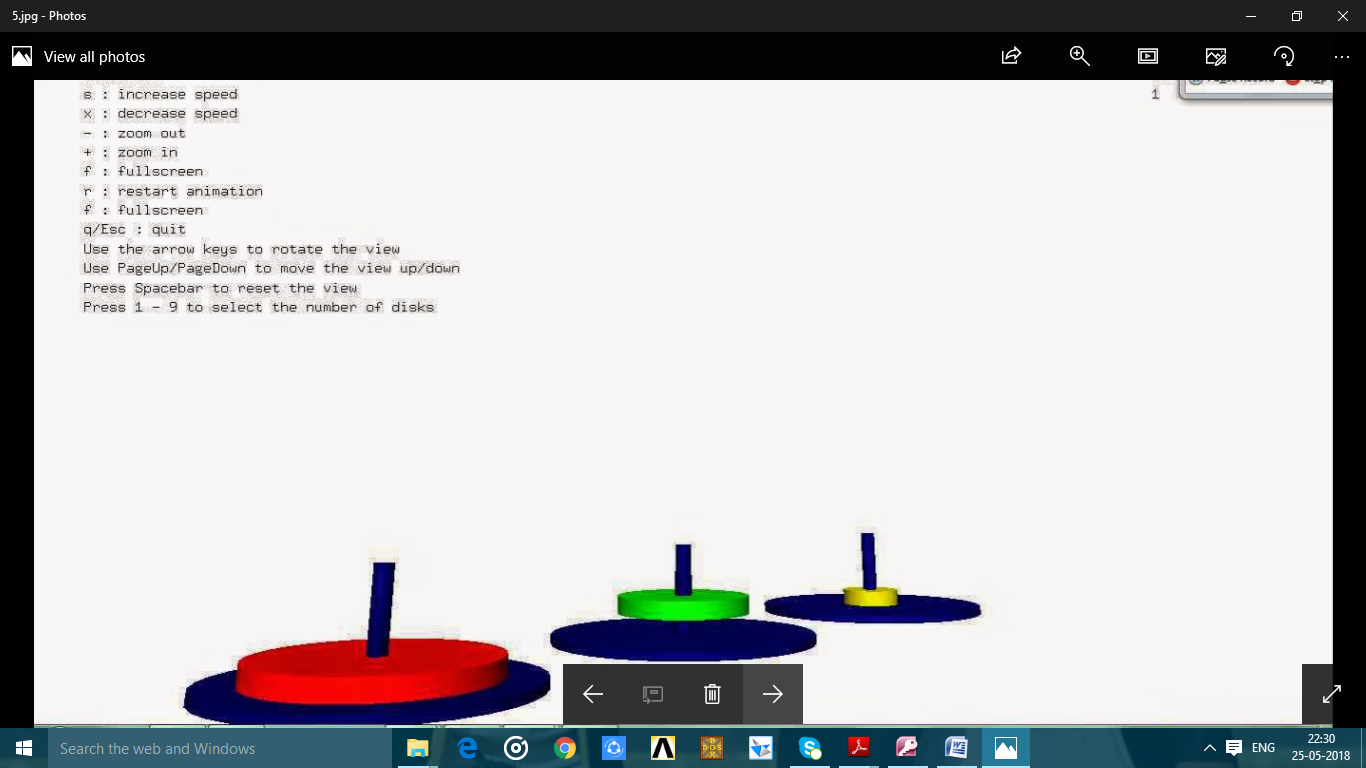
**Fig 1.3 Disk Shifting from one tower to other tower.**



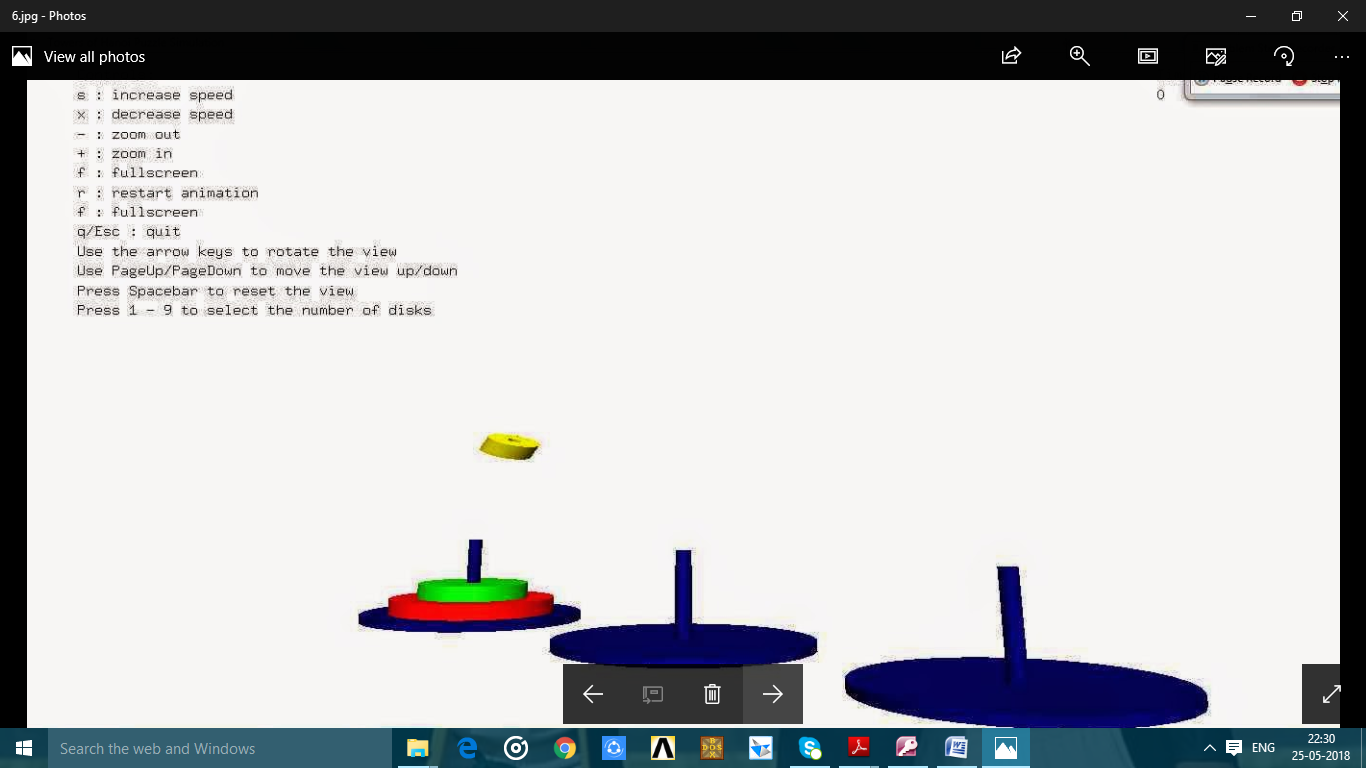
**Fig 1.4 Towers rotated upwards using ‘up’ navigation key.**



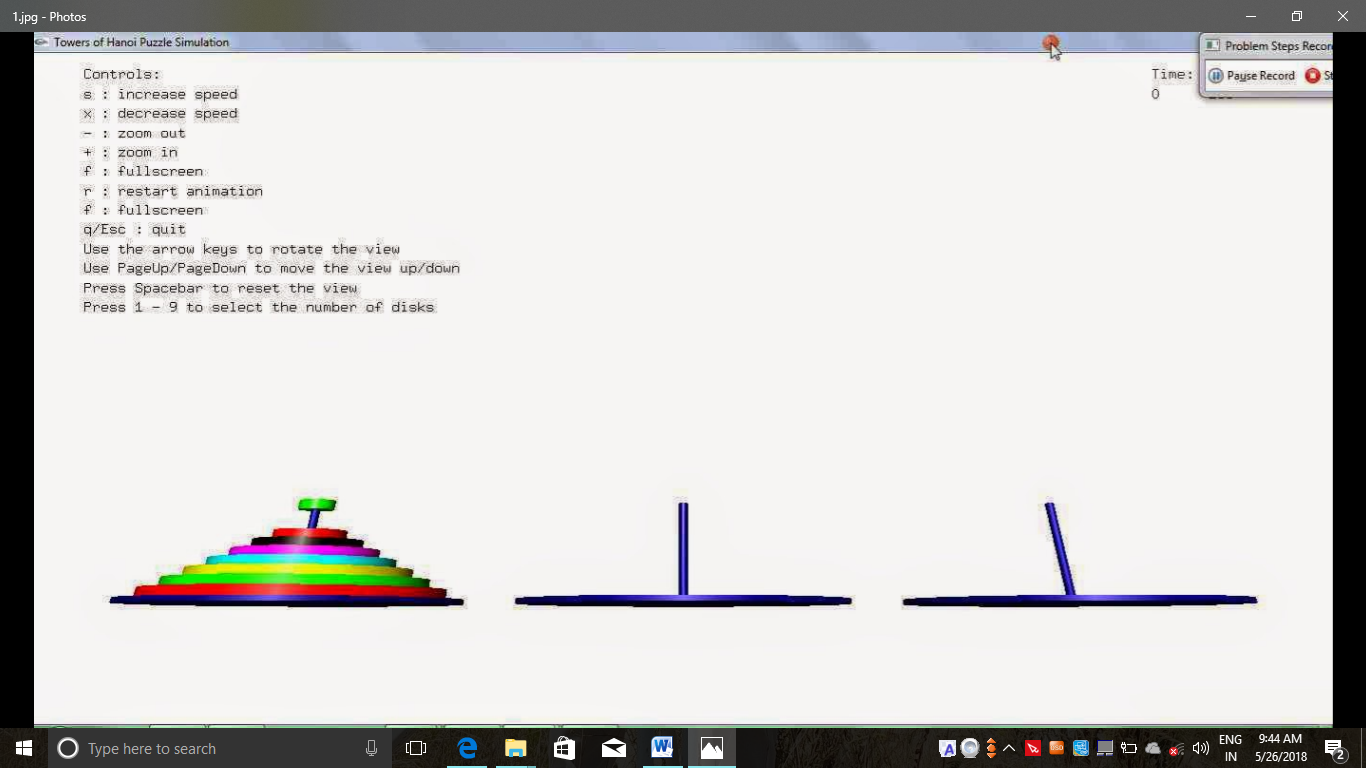
**Fig 1.5 Towers rotated downwards navigation key.using ‘down’ navigation key.**



**Fig 1.6 Towers rotated anticlock wise using ‘right’ navigation key.**

****

**Fig 1.7 Towers rotated clock wise using ‘left’ navigation key.**

****

**Fig 1.8 Reset position of disks on 1st tower.**

**CHAPTER 6**

**CONCLUSION**

This project is mainly aimed to demonstrate the detachment of 3d rocket in space. The objective of this program is to implement simple and basic functions of OpenGL. This project is developed highly on academic based, the project consists a very few functions and implementations of very few concepts. Implementations of certain other functions other than the ones in code can enhance the project to a very high level.

Certain concepts such as implementing texture, viewing from other angles, the display of the object in its original form after any transformations etc. can be enhanced in future.We can use these types and other animated objects in the field of animation industry.

We can add lighting effect by using the instructions available in the openGL. Texture also added to make more attractive.

**BIBLIOGRAPHY**

**BOOKS:**

[1]• Edward Angel: Interactive Computer Graphics A Top –Down

Approach with OpenGL, 2nd Edition, Addison-Wisley, 2000

[2]• F.S.Hill,Jr:Computer Graphics Using OpenGL ,2nd Edition,

Pearson Education, 2001

[3]• James D Foley,Andries van Dam,Steven K Feiner,John

Huges, Computer Graphics, Addison-wesley 1997

[4]• Computer Graphics with Open GL by Hearn and Baker,3rd edition

**WEBSITES:**

[5]• Google search

[6]• www.opengl.org