CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO COMPUTER GRAPHICS

Computer Graphics is concerned with all aspects of producing pictures or images using a computer. Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention
of photography and television.

Applications of Computer Graphics

1. Display of information
2. Design
3. Simulation and animation
4. User interfaces

The Graphics Architecture

Graphics Architecture can be made up of seven components:

1. Display processors
2. Pipeline architectures
3. The graphics pipeline
4. Vertex processing
5. Clipping and primitive assembly
6. Rasterization
7. Fragment processing

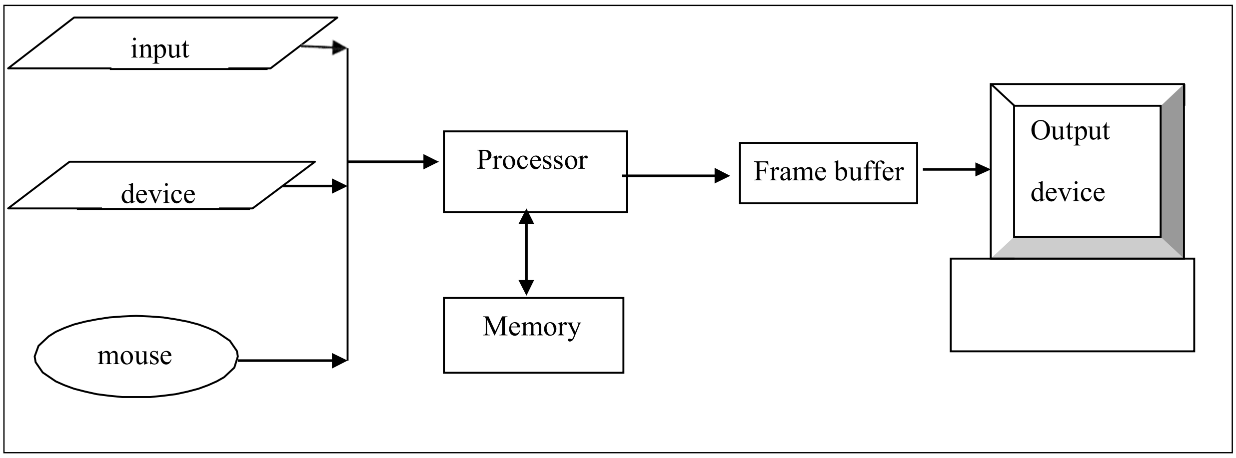

Figure 1.1: Components of Graphics Architecture and their working

1.2 INTRODUCTION TO OPENGL

OpenGL is software used to implement computer graphics. The structure of OpenGL is similar to that of most modern APIs including Java 3D and DirectX. OpenGL is easy to learn, compared with other.

APIs are nevertheless powerful. It supports the simple 2D and 3D programs. It also supports the advanced rendering techniques. OpenGL API explains following 3 components

1. Graphics functions
2. Graphics pipeline and state machines
3. The OpenGL interfaces

There are so many polygon types in OpenGL like triangles, quadrilaterals, strips and fans. There are 2 control functions, which will explain OpenGL through,

1. Interaction with window system
2. Aspect ratio and view ports

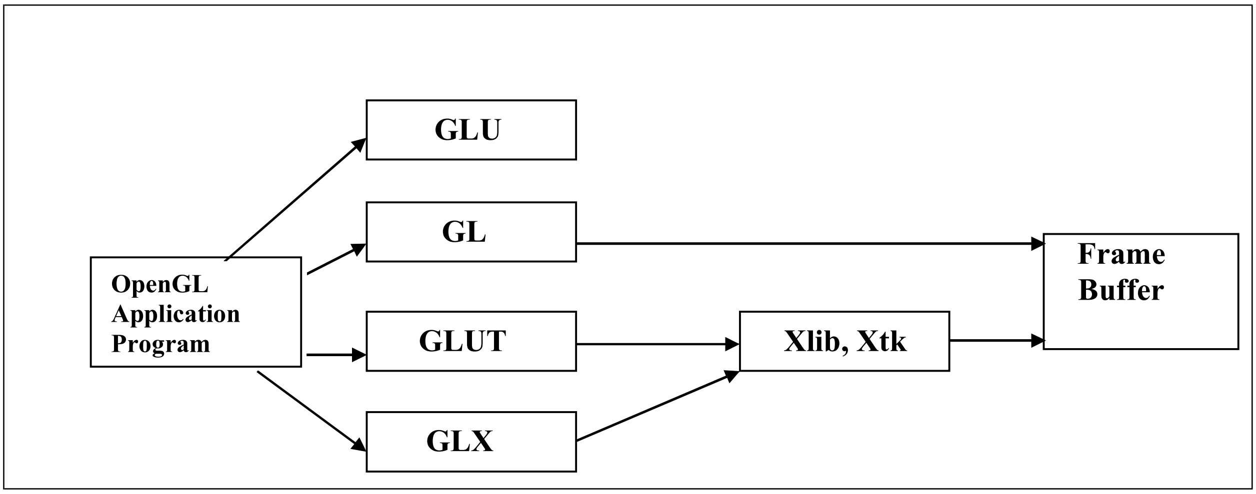

Figure 1.2: OpenGL Library organization

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. This ordering, as shown in Figure 1.2, is not a strict rule of how OpenGL is implemented but provides a reliable guide for predicting what OpenGL will do. The following diagram shows the assembly line approach, which OpenGL takes to process data. Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo the same final steps before the final pixel data is written into the frame buffer.

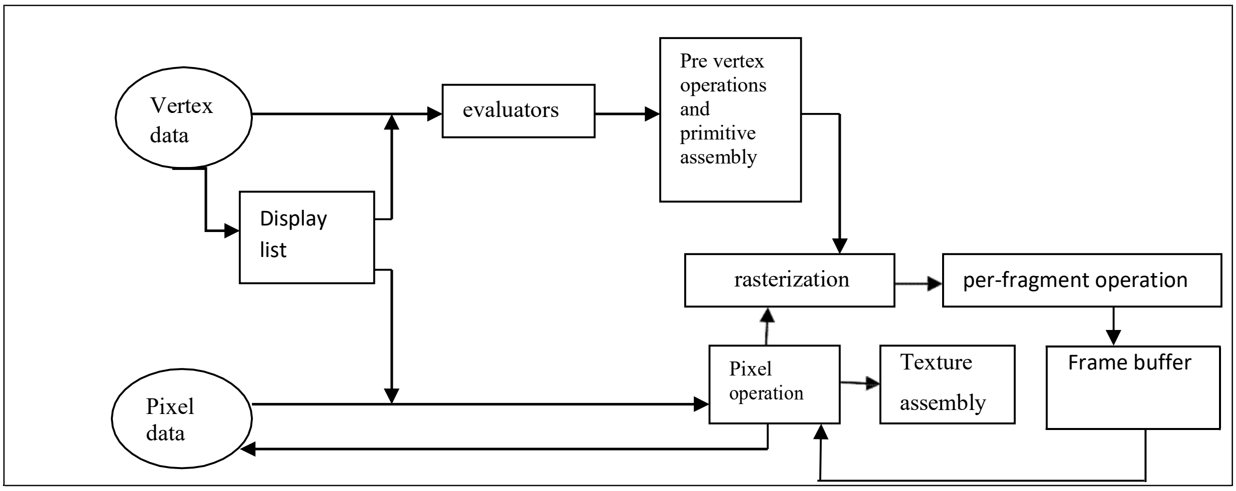

Figure 1.3: OpenGL Order of Operations

CHAPTER 2

REQUIREMENTS SPECIFICATION

2.1 SOFTWARE REQUIREMENTS

- Operating system – Windows 10
- Code::Blocks 17.12
- OPENGL library files – GL, GLU, GLUT
- Language used is C/C++

2.2 HARDWARE REQUIREMENTS

- Processor – Intel i5 7^{th} Gen
- Memory – 8GB RAM
- 1TB Hard Disk Drive
- Mouse or other pointing device
- Keyboard
- Display device

CHAPTER 3

SYSTEM DEFINITION

3.1 PROJECT DESCRIPTION

This project contains Ferris Wheel. When the user presses ‘c/C’ key, wheel will start rotating and vice-versa. The user can control the speed of rotation by pressing ‘v/V’ or ‘x/X’ keys from keyboard where ‘v/V’ increases the speed and ‘x/X’ decreases the speed. When the user press ‘w/W’,’s/S’, ’a/A’ & ’d/D’ key, it will rotate the whole wheel into Top, Bottom, Right and Left direction. These are showed in the System-Architecture.

The user can also increase and decrease the number of cabins by pressing key ‘+/1’ and ‘-/0’ respectively.

3.2 USER DEFINED FUNCTIONS

- **void reshape(int w, int h):** The reshape function is a call-back function which is called to regain the size or shape of the application window.
- **void draw_scene():** This function is used to draw the base structure for the Ferris wheel.
- **void keyboard(unsigned char key, int x, int y):** This function contains the key function that is used to control the movements.
- **float toRad(float deg):** This function is used to change the degree into radian.
- **void init(void):** This function is used to initialize the components of the window created .
- **void rotate():**This function is used to rotates the blades of fan.
- **void idle():** This function is used for animation.

3.3 DATA FLOW DIAGRAM

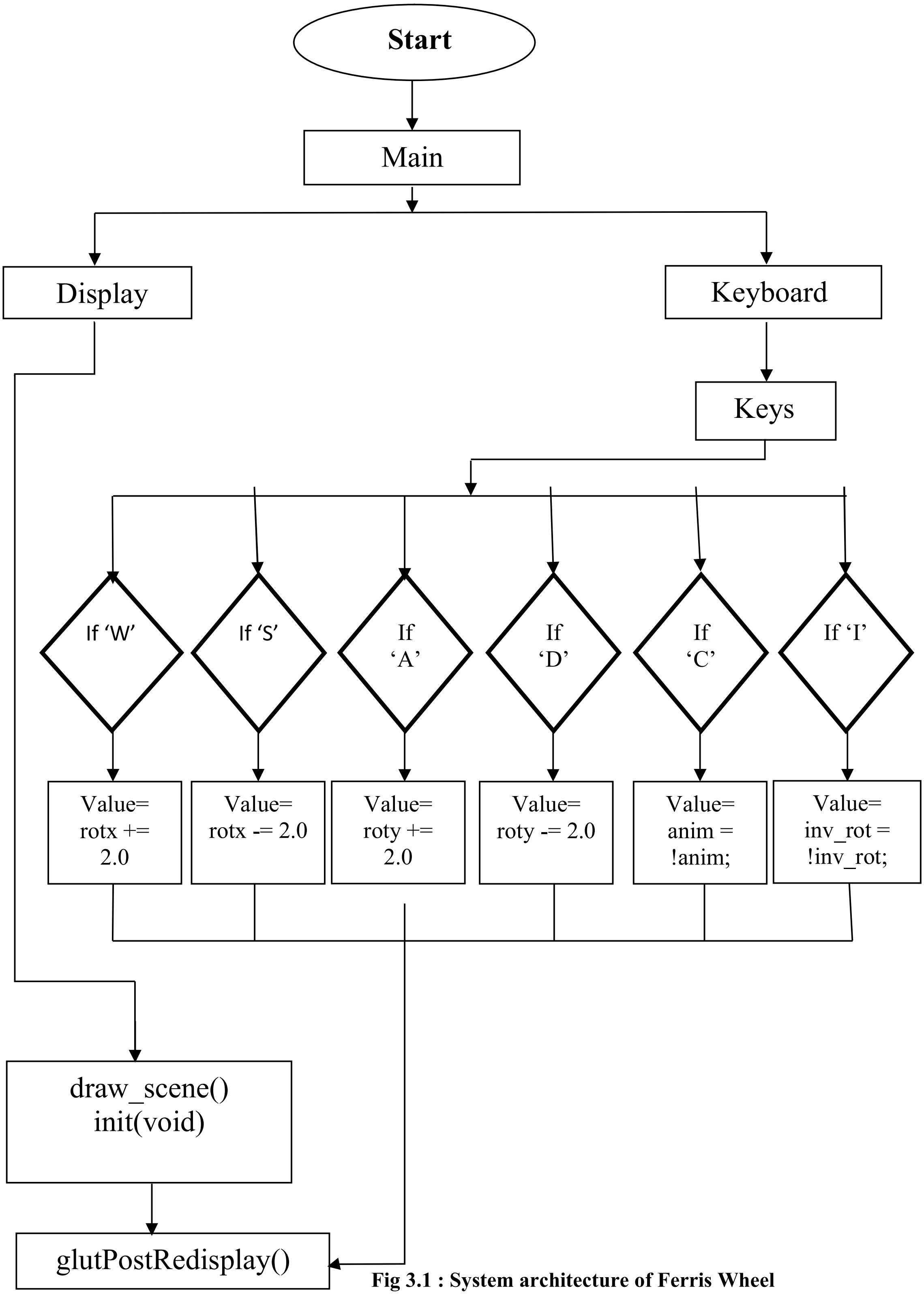

Fig 3.1 : System architecture of Ferris Wheel

CHAPTER 4

IMPLEMENTATION

4.1 SOURCE CODE

```
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/gl.h>
#include <cmath>
#include<stdio.h>
#define M_PI 3.1414
#define WHEEL 1
#define CABIN 2

GLfloat global_ambient[] = { 0.3f,0.3f,0.3f,1.0f };
GLfloat dir_ambient[] = { 0.6f,0.6f,0.6f,1.0f };
GLfloat dir_diffuse[] = { 0.8f,0.8f,0.8f,1.0f };
GLfloat dir_specular[] = { 1.0f,1.0f,1.0f,1.0f };
GLfloat dir_position[] = { 0.0f,1.0f,1.0f,0.0f };
// Color Material - Specular
GLfloat mat_specular[] = { 0.6,0.6,0.6,1.0 };

GLfloat rotx = 0, roty = 0;
GLfloat c_radius, w_rot;
GLint n_bars = 16, w_speed = 0.25;
bool anim = false;
bool inv_rot = false;
float toRad(float deg)
{
        return (deg*M_PI) / 180;
}
void init(void)
{
        glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient);
        glLightfv(GL_LIGHT0, GL_AMBIENT, dir_ambient);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, dir_diffuse);
        glLightfv(GL_LIGHT0, GL_SPECULAR, dir_specular);
        glLightfv(GL_LIGHT0, GL_POSITION, dir_position);
        glEnable(GL_LIGHT0);
        glEnable(GL_COLOR_MATERIAL);
```

```
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);

glMateriali(GL_FRONT, GL_SHININESS, 30);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_DEPTH_TEST);
glEnable(GL_NORMALIZE);
glShadeModel(GL_SMOOTH);
glNewList(WHEEL, GL_COMPILE);
glColor3f(0.5, 0, 0);

// Center front
glBegin(GL_TRIANGLE_FAN);
glVertex3f(0, 0, 15);
for (float angle = 0; angle <= 360; angle += 0.1)
{
        glVertex3f(5 * cos(toRad(angle)), 5 * sin(toRad(angle)), 15);
}
glEnd();
// Center
glBegin(GL_QUAD_STRIP);
for (float angle = 0; angle <= 360; angle += 0.1) {
        glVertex3f(5 * cos(toRad(angle)), 5 * sin(toRad(angle)), 15);
        glVertex3f(5 * cos(toRad(angle)), 5 * sin(toRad(angle)), -15);
}
glEnd();
// Center back
glBegin(GL_TRIANGLE_FAN);
glVertex3f(0, 0, -15);
for (float angle = 0; angle <= 360; angle += 0.1) {
        glVertex3f(5 * cos(toRad(angle)), 5 * sin(toRad(angle)), -15);
}
glEnd();
// Wheel structure front
for (float angle = 0; angle <= 360; angle += 0.1) {
        glBegin(GL_POLYGON);
        glVertex3f(30 * cos(toRad(angle)), 30 * sin(toRad(angle)), 4);
        glVertex3f(28 * cos(toRad(angle)), 28 * sin(toRad(angle)), 4);
        glVertex3f(28 * cos(toRad(angle + 0.2)), 28 * sin(toRad(angle + 0.2)), 4);
        glVertex3f(30 * cos(toRad(angle + 0.2)), 30 * sin(toRad(angle + 0.2)), 4);
```

```
        glEnd();
    }
    glBegin(GL_QUAD_STRIP);
    for (float angle = 0; angle <= 360; angle += 0.1) {
        glVertex3f(30 * cos(toRad(angle)), 30 * sin(toRad(angle)), 4);
        glVertex3f(30 * cos(toRad(angle)), 30 * sin(toRad(angle)), 3.5);
    }
    glEnd();
    glBegin(GL_QUAD_STRIP);
    for (float angle = 0; angle <= 360; angle += 0.1) {
        glVertex3f(28 * cos(toRad(angle)), 28 * sin(toRad(angle)), 4);
        glVertex3f(28 * cos(toRad(angle)), 28 * sin(toRad(angle)), 3.5);
    }
    glEnd();
    // Wheel structure front
    for (float angle = 0; angle <= 360; angle += 0.1) {
        glBegin(GL_POLYGON);
        glVertex3f(30 * cos(toRad(angle)), 30 * sin(toRad(angle)), 3.5);
        glVertex3f(28 * cos(toRad(angle)), 28 * sin(toRad(angle)), 3.5);
        glVertex3f(28 * cos(toRad(angle + 0.2)), 28 * sin(toRad(angle + 0.2)), 3.5);
        glVertex3f(30 * cos(toRad(angle + 0.2)), 30 * sin(toRad(angle + 0.2)), 3.5);
        glEnd();
    }
    // Wheel structure back
    for (float angle = 0; angle <= 360; angle += 0.1) {
        glBegin(GL_POLYGON);
        glVertex3f(30 * cos(toRad(angle)), 30 * sin(toRad(angle)), -4);
        glVertex3f(28 * cos(toRad(angle)), 28 * sin(toRad(angle)), -4);
        glVertex3f(28 * cos(toRad(angle + 0.2)), 28 * sin(toRad(angle + 0.2)), -4);
        glVertex3f(30 * cos(toRad(angle + 0.2)), 30 * sin(toRad(angle + 0.2)), -4);
        glEnd();
    }
    glBegin(GL_QUAD_STRIP);
    for (float angle = 0; angle <= 360; angle += 0.1) {
        glVertex3f(30 * cos(toRad(angle)), 30 * sin(toRad(angle)), -4);
        glVertex3f(30 * cos(toRad(angle)), 30 * sin(toRad(angle)), -3.5);
    }
    glEnd();
    glBegin(GL_QUAD_STRIP);
    for (float angle = 0; angle <= 360; angle += 0.1) {
        glVertex3f(28 * cos(toRad(angle)), 28 * sin(toRad(angle)), -4);
```

```
        glVertex3f(28 * cos(toRad(angle)), 28 * sin(toRad(angle)), -3.5);
    }
    glEnd();
    // Wheel structure back
    for (float angle = 0; angle <= 360; angle += 0.1) {
        glBegin(GL_POLYGON);
        glVertex3f(30 * cos(toRad(angle)), 30 * sin(toRad(angle)), -3.5);
        glVertex3f(28 * cos(toRad(angle)), 28 * sin(toRad(angle)), -3.5);
        glVertex3f(28 * cos(toRad(angle + 0.2)), 28 * sin(toRad(angle + 0.2)), -3.5);
        glVertex3f(30 * cos(toRad(angle + 0.2)), 30 * sin(toRad(angle + 0.2)), -3.5);
        glEnd();
    }
    glEndList();
    // CAbin
    glNewList(CABIN, GL_COMPILE);
    glColor3f(0.5, 0, 0);
    // Bar
    glBegin(GL_TRIANGLE_FAN);
    glVertex3f(0, 0, -4);
    for (float angle = 0; angle <= 360; angle += 0.1) {
        glVertex3f(0.5*cos(toRad(angle)), 0.5*sin(toRad(angle)), -4);
    }
    glEnd();
    glBegin(GL_QUAD_STRIP);
    for (float angle = 0; angle <= 360; angle += 0.1) {
        glVertex3f(0.5*cos(toRad(angle)), 0.5*sin(toRad(angle)), -3.5);
        glVertex3f(0.5*cos(toRad(angle)), 0.5*sin(toRad(angle)), 3.5);
    }
    glEnd();
    glBegin(GL_TRIANGLE_FAN);
    glVertex3f(0, 0, 4);
    for (float angle = 0; angle <= 360; angle += 0.1) {
        glVertex3f(0.5*cos(toRad(angle)), 0.5*sin(toRad(angle)), 4);
    }
    glEnd();
    // Cono
    glPushMatrix();
    glColor3f(0.95, 0.67, 0.06);
    glTranslatef(0, -5, 0);
    glRotatef(-90, 1, 0, 0);
    glutSolidCone(2, 5, 50, 50);
```

```
        glPopMatrix();
        glEndList();
}

void draw_scene() {
        glPushMatrix();
        glColor3f(0.02, 0.22, 0.02);
        glTranslatef(0, -40, 0);
        glScalef(50, 2, 50);
        glutSolidCube(2);
        glPopMatrix();
        glPushMatrix();
        glCallList(WHEEL);
        glPopMatrix();
        glPushMatrix();
        glRotatef(w_rot, 0, 0, 1);
        for (int i = 0; i < n_bars; i++) {
                c_radius = (360.0 / n_bars)*i;
                glPushMatrix();
                // Axis
                glPushMatrix();
                glColor3f(0.92, 0.12, 0.12);
                glRotatef(c_radius, 0, 0, 1);
                glRotatef(-11, 1, 0, 0);
                glTranslatef(0, 14.5, 10);
                glScalef(1, 28, 0.5);
                glutSolidCube(1);
                glPopMatrix();
                glPushMatrix();
                glColor3f(0.92, 0.12, 0.12);
                glRotatef(c_radius, 0, 0, 1);
                glRotatef(11, 1, 0, 0);
                glTranslatef(0, 14.5, -10);
                glScalef(1, 28, 0.5);
                glutSolidCube(1);
                glPopMatrix();
                glTranslatef(28 * cos(toRad(c_radius)), 28 * sin(toRad(c_radius)), 0);
                glRotatef(-w_rot, 0, 0, 1);
                glCallList(CABIN);
                glPopMatrix();
        }
```

```
glPopMatrix();
glPushMatrix();
glColor3f(0.3, 0.3, 0.3);
glRotatef(-30, 0, 0, 1);
glTranslatef(0, -25, 12);
glScalef(3, 40, 2);
glutSolidCube(1);
glPopMatrix();
glPushMatrix();
glColor3f(0.3, 0.3, 0.3);
glRotatef(30, 0, 0, 1);
glTranslatef(0, -25, 12);
glScalef(3, 40, 2);
glutSolidCube(1);
glPopMatrix();
glPushMatrix();
glColor3f(0.3, 0.3, 0.3);
glRotatef(-30, 0, 0, 1);
glTranslatef(0, -25, -12);
glScalef(3, 40, 2);
glutSolidCube(1);
glPopMatrix();
glPushMatrix();
glColor3f(0.3, 0.3, 0.3);
glRotatef(30, 0, 0, 1);
glTranslatef(0, -25, -12);
glScalef(3, 40, 2);
glutSolidCube(1);
glPopMatrix();
glPushMatrix();
glTranslatef(20, -30, 30);
glColor3f(1, 0, 0);
glPushMatrix();
glRotatef(-90, 1, 0, 0);
glutSolidCone(2, 2, 50, 50);
glPopMatrix();
glColor3f(1, 0.5, 0.5);
glutSolidSphere(1, 50, 50);
glPushMatrix();
glTranslatef(0, -2, 0);
glScalef(0.5, 4, 0.5);
```

```
        glutSolidCube(1);
        glPopMatrix();
        glPushMatrix();
        glTranslatef(-0.9, -6, 0);
        glRotatef(-20, 0, 0, 1);
        glScalef(0.5, 5, 0.5);
        glutSolidCube(1);
        glPopMatrix();

        glPushMatrix();
        glTranslatef(0.9, -6, 0);
        glRotatef(20, 0, 0, 1);
        glScalef(0.5, 5, 0.5);
        glutSolidCube(1);
        glPopMatrix();
        glPushMatrix();
        glTranslatef(-0.9, -2, 0);
        glRotatef(-120, 0, 0, 1);
        glScalef(0.5, 2, 0.5);
        glutSolidCube(1);
        glPopMatrix();
        glPushMatrix();
        glTranslatef(0.9, -2, 0);
        glRotatef(120, 0, 0, 1);
        glScalef(0.5, 2, 0.5);
        glutSolidCube(1);
        glPopMatrix();
        glPopMatrix();
}

void display(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glClearColor(0.64, 0.77, 1, 1);
        glPushMatrix();
        glTranslatef(0, 0, -100);
        glRotatef(rotx, 1, 0, 0);
        glRotatef(roty, 0, 1, 0);
        draw_scene();
        glPopMatrix();
        glutSwapBuffers();
}
```

```
void idle() {
	if (anim) {
		if (inv_rot)
			w_rot -= w_speed;
		else
			w_rot += w_speed;
		glutPostRedisplay();
	}
}
 void reshape(int w, int h) {
	// Prevent a divide by zero
	if (h == 0)
		h = 1;
	// Set Viewport to window dimensions
	glViewport(0, 0, w, h);
	// Calculate aspect ratio of the window
	float fAspect = (GLfloat)w / (GLfloat)h;
	// Set the perspective coordinate system
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(65.0, (GLfloat)w / (GLfloat)h, 1.0, 800.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}
void keyboard(unsigned char key, int x, int y) {
	switch (key) {
	case 'w':
	case 'W':
		rotx += 2.0;
		break;
	case 's':
	case 'S':
		rotx -= 2.0;
		break;
	case 'a':
	case 'A':
		roty += 2.0;
		break;
	case 'd':
	case 'D':
```

```
			roty -= 2.0;
			break;
		case '+':
		case '1':
			if (n_bars < 24)
				n_bars++;
			break;
		case '-':
		case '0':
			if (n_bars > 8)
				n_bars--;
			break;
		case 'x':
		case 'X':
			w_speed = w_speed - 0.05;
			break;
		case 'v':
		case 'V':
			w_speed = w_speed + 0.05;
			break;
		case 'c':
		case 'C':
			anim = !anim;
			break;
		case 'i':
		case 'I':
			inv_rot = !inv_rot;
		}
		glutPostRedisplay();
	}

	int main(int argc, char** argv) {
		printf("\n****************************************************\n");
		printf("\n------------------------VTU PROJECT-----------------------\n");
		printf("\n---------------------Mr. ANURAJ DEEP--------------------\n");
		printf("\n----------------------USN 1GA16CS017---------------------\n");
		printf("\n****************************************************\n\n");
		printf("\n--------------FERRIS WHEEL Mini Project---------------\n");
		printf("\n---------------------VIth SEM (17CSL68) ---------------------\n");
		printf("\n---Department of Computer Science and Engineering---\n\n");
		printf("\n--------------Global Academy of Technology--------------\n\n");
```

```
	printf("\n**************************************************\n\n\n");
	printf("\n	Top Movement		:w or W\n");
	printf("\n	Bottom Movement		:s or S\n");
	printf("\n	Rotate Right		:a or A\n");
	printf("\n	Rotate Left		:d or D\n");
	printf("\n	Wheel Rotation		:c or C\n");
	printf("\n	Clockwise & Anti-clockwise :i or I\n");
	printf("\n	Wheel Rotation		:c or C\n");
	printf("\n	Increase Speed		:v or V\n");
	printf("\n	Decrease Speed		:x or X\n");
	printf("\n	Increase No. of Cabin	:+ or 1\n");
	printf("\n	Decrease No. of Cabin	:- or 0\n");
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(1000, 700);
	glutInitWindowPosition(500, 100);
	glutCreateWindow("FERRIS WHEEL");
	init();
	glutReshapeFunc(reshape);
	glutIdleFunc(idle);
	glutDisplayFunc(display);
	glutKeyboardFunc(keyboard);
	glutMainLoop();
	return 0;
}
```

CHAPTER 5

TESTING AND RESULTS

5.1 DIFFERENT TYPES OF TESTING

1. Unit Testing

Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components.

2. Module Testing

A module is a collection of dependent components such as a object class, an abstract Data type or some looser collection of procedures and functions. A module related Components, so can be tested without other system modules.

3. System Testing

This is concerned with finding errors that result from unanticipated interaction between Sub-system interface problems.

4. Acceptance Testing

The system is tested with data supplied by the system customer rather than simulated test data.

5.2 TEST CASES

The test cases provided here test the most important features of the project.

Test cases for the project:

Input	Expected Result	Observed Result	Remark
Press ‘w/W’ from keyboard	Vertically Upward Movement	Wheel moved in upward direction	Pass
Press ‘s/S’ from keyboard	Vertically Downward Movement	Wheel moved in downward direction	Pass
Press ‘a/A’ from keyboard	Right-Hand Side Movement	Wheel moved along right-hand side	Pass
Press ‘d/D’ from keyboard	Left-Hand Side Movement	Wheel moved along left-hand side	Pass
Press ‘c/C’ from keyboard	Rotation of Ferris Wheel and Vice-versa	Wheel started rotating and vice-versa	Pass
Press ‘x/X’ from keyboard	Gradually Increase Speed of Wheel	Speed of Wheel increased gradually	Pass
Press ‘v/V’ from keyboard	Gradually Decrease Speed of Wheel	Speed of Wheel decreased gradually	Pass
Press ‘i/I’ from keyboard	Rotation of Wheel in Clockwise and Anti Clockwise Direction	Wheel rotated in clockwise and anti-clockwise direction	Pass
Press ‘+/1’ from keyboard	It’ll Increase No. of Cabins	No. of Cabins increased by 1 each time	Pass
Press ‘-/0’ from keyboard	It’ll Decrease No. of Cabins	No. of Cabins decreased by 1 each time	Pass

CHAPTER 6

SNAPSHOTS

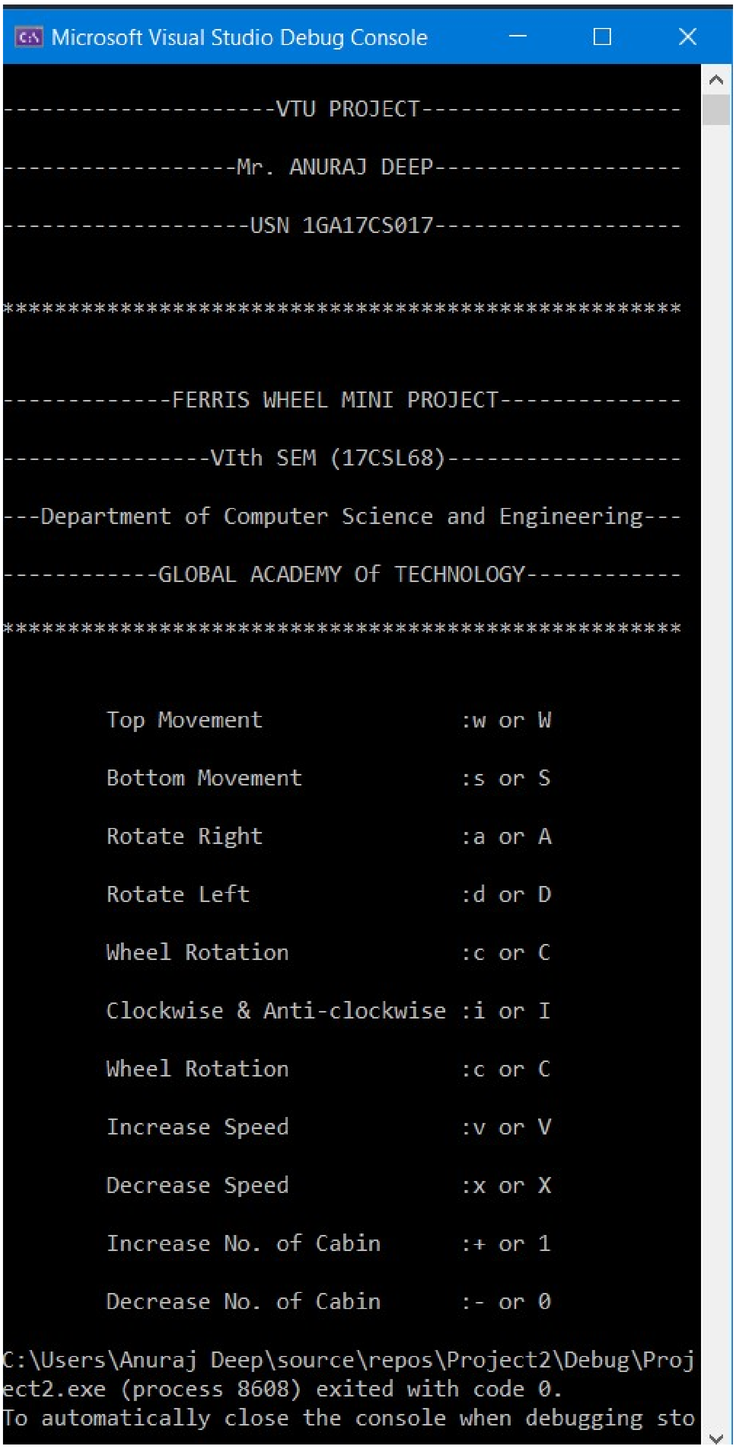

Figure 6.1: Console Window

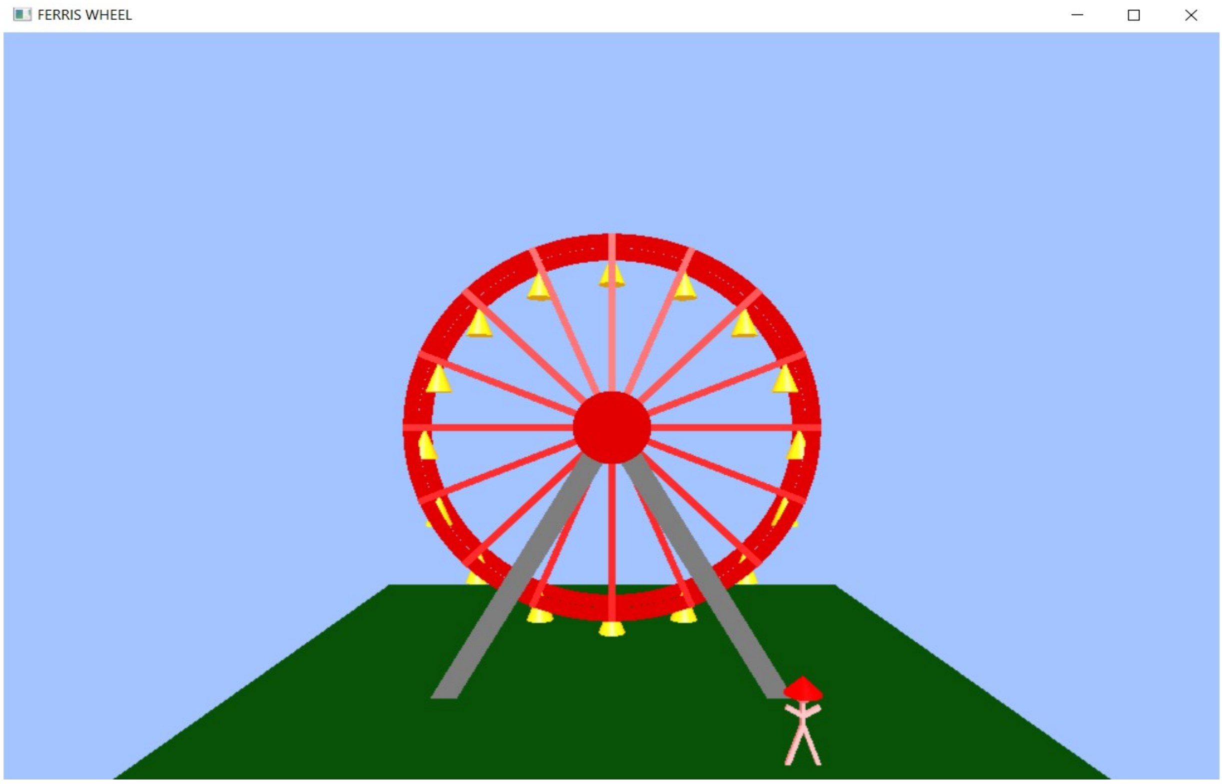

Figure 6.2: Ferris wheel front view

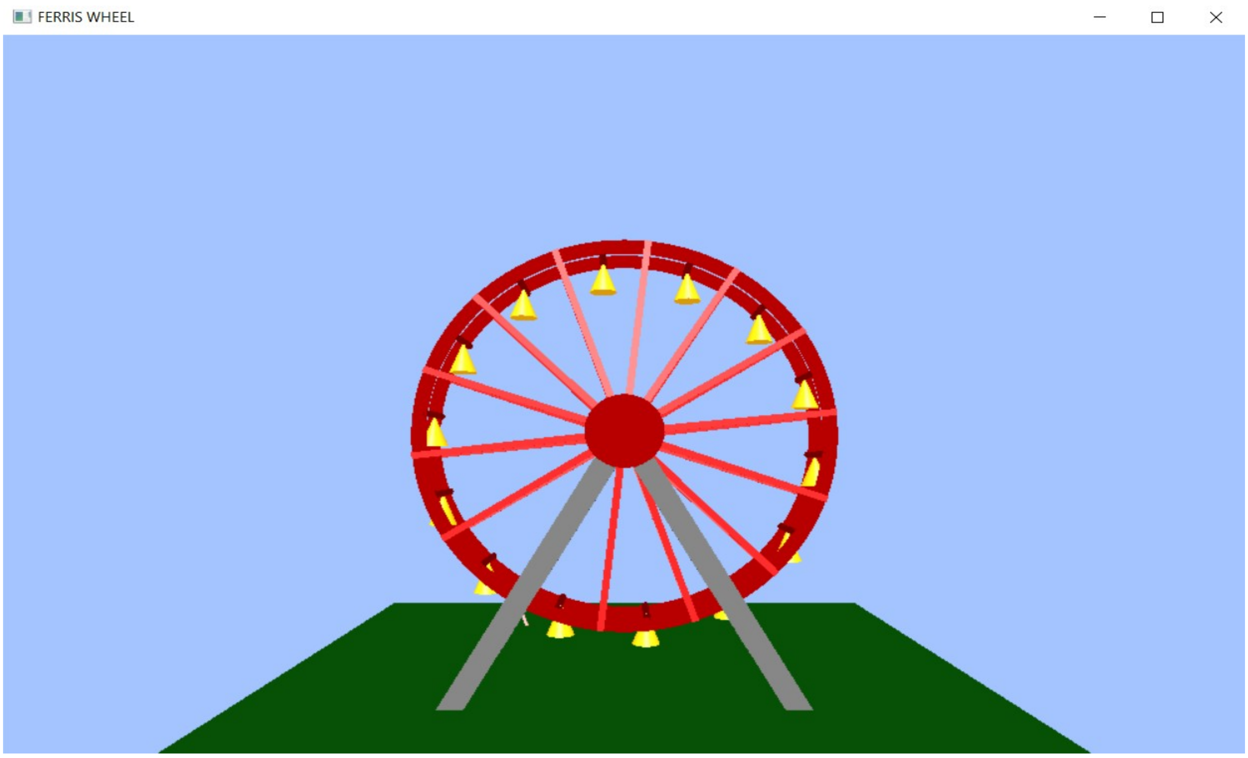

Figure 6.3: Ferris wheel back view

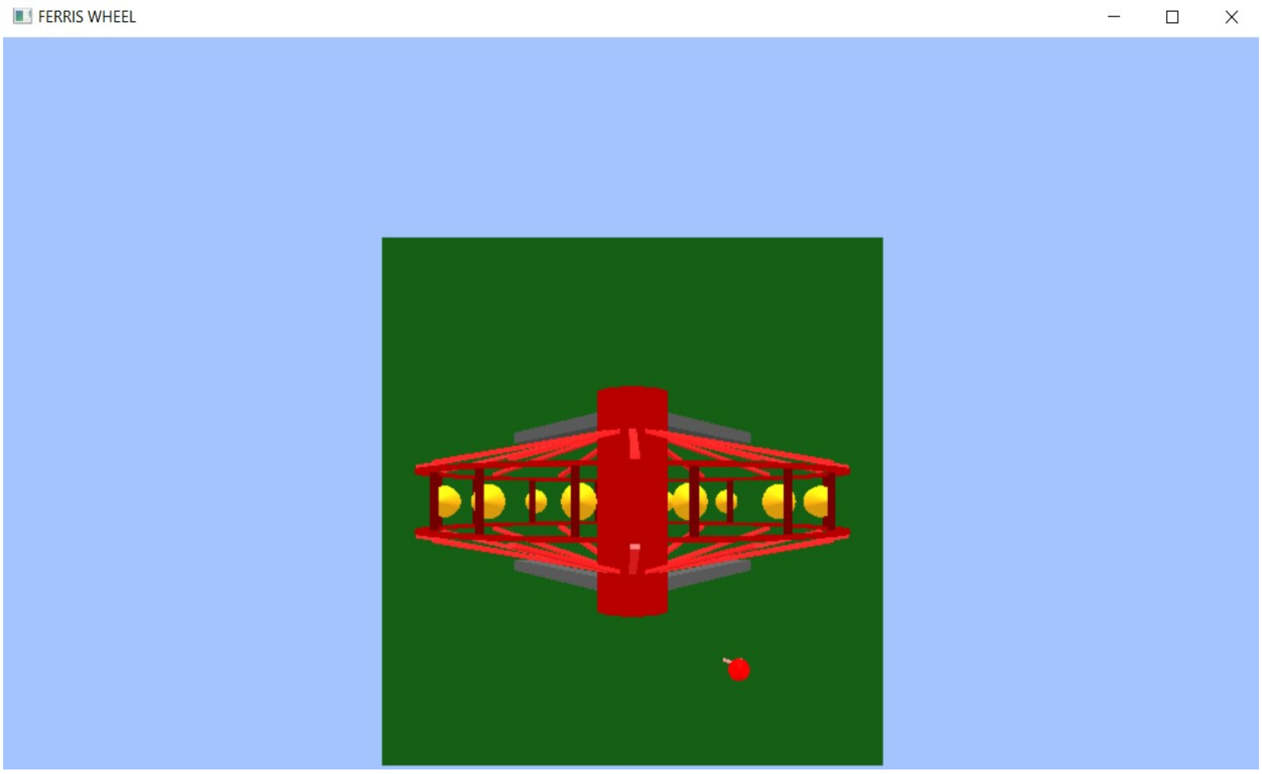

Figure 6.4: Ferris Wheel top view

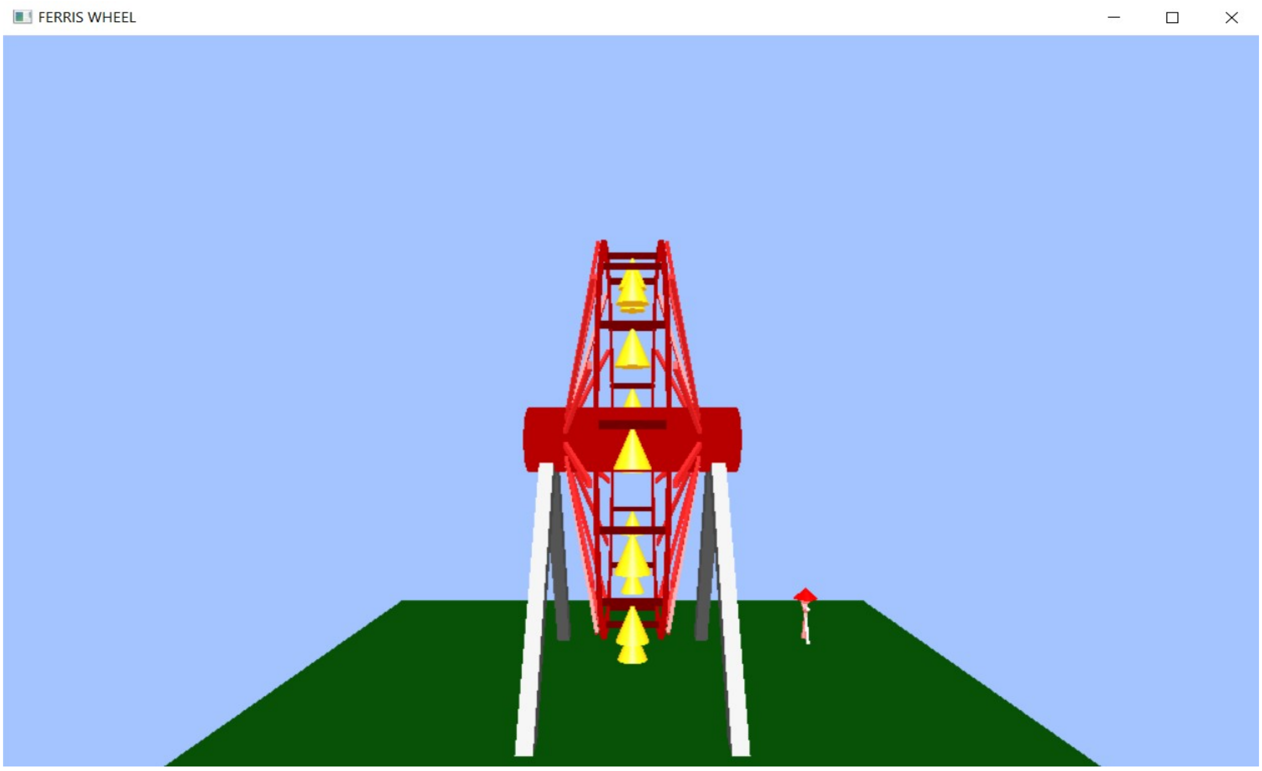

Figure 6.5: Ferris Wheel side view

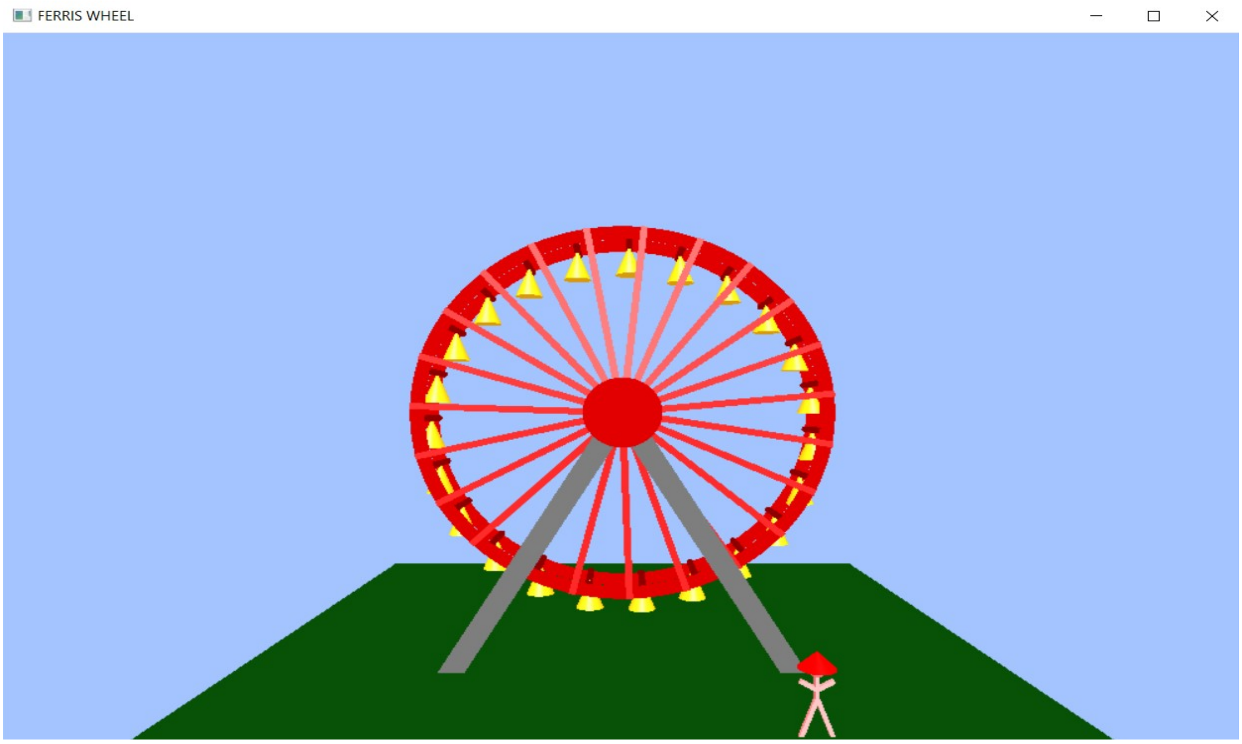

Figure 6.6: Ferris Wheel with maximum number of Cabins

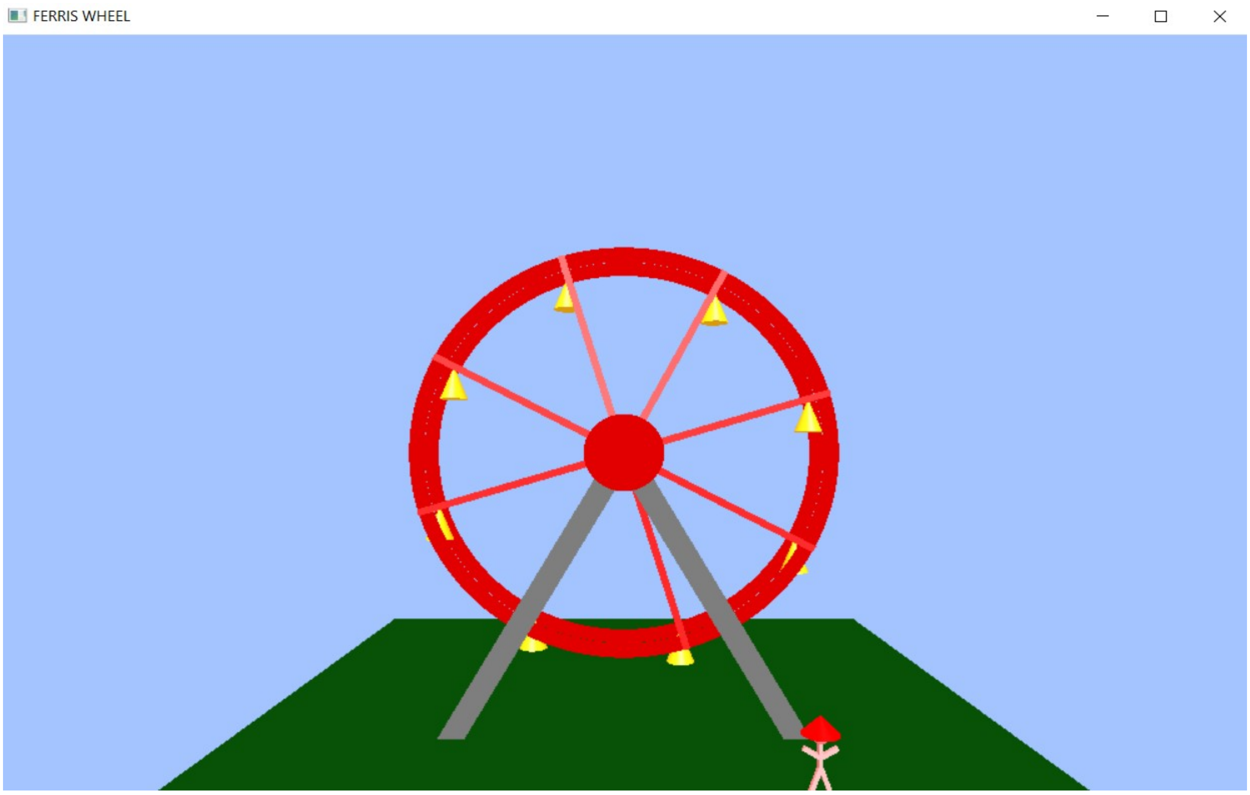

Figure 6.7: Ferris Wheel with lowest number of Cabin

CONCLUSION

Using OpenGL functions and user defined functions a basic Ferris Wheel simulation is being done which is runs at different speed and no. of cabins can also be change. This is basic graphic animation made using only some built-in function and APIs. There are many functions and APIs available in OpenGL that makes animation effective and realistic.

The user-friendly interface allows the user to interact with it very effectively. So, I conclude on note that this project has given me a great exposure to the OpenGL and computer graphics. This is very reliable graphics package supporting various primitive objects like polygon, line loops, ambient light, triangle fan, quad strip etc. Also, this project is designed in such a way that one can view it from any directions using keyboard keys. Transformations like translation, rotation is also provided.

BIBLIOGRAPHY

References

[1]. Donald D. Hearn, Pauline Baker, Warren Carithers: Computer Graphics with OpenGL, 4th Edition, Pearson Education.

[2]. Edward Angel: Interactive Computer Graphics A Top-Down Approach with OpenGL, 5th Edition, Pearson Education, 2008

[3]. www.opengl.org

[4]. www.khronos.org

[5]. www.github.com