**CHAPTER 1**

# INTRODUCTION

## 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

input

device

mouse

Memory

Frame buffer

Processor

Output device

**Figure 1.1: Components of Graphics Architecture and their working**



## 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 are2 control functions, which will explain OpenGL through,

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

**GLX**

**Xlib, Xtk**

**GLUT**

**OpenGL Application Program**

**Frame Buffer**

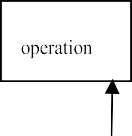
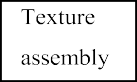
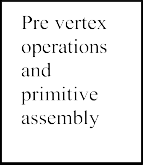
**GL**

**GLU**

**Figure 1.2: OpenGL Library organization**

**OpenGL Application Program**

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.



Vertex

Pixel

Pixel

Display

**Figure 1.3: OpenGL Order of Operations**

**CHAPTER 2**

# REQUIREMENTS SPECIFICATION

## SOFTWARE REQUIREMENTS

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

## HARDWARE REQUIREMENTS

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

**CHAPTER 3**

# IMPLEMENTATION

Implementation is the stage where all planned activities are put into action. Before the implementation of a project, the implementors should identify their strength and weaknesses, opportunities and threats.

Implementation also includes a pseudo code.

## 3.1 Pseudo code:

void display()

{

Clears the window and draws the room as viewed from the current position.

}

void keyboard()

{

This function is used to provide the user interaction.

}

void draw\_scene()

{

This function is used when we want to highlight any object in the input image.

}

int main()

{

Initializes the glut library and creates the display window. It also creates the menu.

}

## 3.2 Flowchart:

**Start**

Main

****

Display

Keyboard



If ‘W’

If ‘S’

If ‘A’

If ‘D’

If ‘C’

If ‘I’

Value= rotx -= 2.0

Value= roty += 2.0

Value=inv\_rot =

!inv\_rot;

Value=anim=

!anim;

Value= roty -= 2.0

Value=

rotx += 2.0

Keys

glutPostRedisplay()

draw\_scene() init(void)

**Fig 4.2.1: Flowchart of the project**

## Function used:

**3.3.1 User defined functions used in the project:**

* **Void draw\_scene(void)**

This method specify is used to draw the giant wheel and person in the scene and here we specify the color of giant wheel and person object.

* **Void init()**

Here we specify the points and gl constants required to draw the gaint wheel and the cabins attached to it . We specify the logic of wheel rotation and lighting settings.

* **Float toRad(float deg)**

This function is used to convert given degree to radians.

Parameters:

float deg – This accepts a floating point value which specifies degree passed from called function**.**

Return type:

float – This function returns a float value after calculating/converting given degrees to radians.

## Standard library functions used in the project:

### gluLookAt()

gluLookAt creates a viewing matrix derived from an eye point, a reference point indicating the center of the scene, and an UP vector. The matrix maps the reference point to the negative z axis and the eye point to the origin.

### glEnable(GL\_LIGHTING)

Enables lighting calculations in OpenGL.

### glEnable(GL\_LIGHT0), glEnable(GL\_LIGHT1)

This function enables specific light source.

### glMaterialfv(GLenum face, GLenum pname, const GLfloat\* params)

Sets material properties for objects in the scene.

face: Specifies whether the material properties apply to the front (GL\_FRONT) or back (GL\_BACK) faces of the objects.

pname: Specifies the material parameter to be set. Common parameters include:

GL\_AMBIENT: Ambient material color.

GL\_DIFFUSE: Diffuse material color.

GL\_SPECULAR: Specular material color.

GL\_SHININESS: Material shininess (specular exponent).

params: An array of values representing the desired parameter.

### glShadeModel()

The function in OpenGL is used to specify the shading model to be used for rendering polygons. The shading model determines how colors are interpolated across the surface of a polygon, affecting the visual appearance of objects in a scene.

The function has a single parameter that specifies the shading model. The shading model can take one of two values:

GL\_FLAT: This option specifies flat shading. In flat shading, the color of the polygon is determined by the color of a single vertex, typically the first vertex of the polygon. The color remains constant across the entire polygon, resulting in a faceted appearance.

GL\_SMOOTH: This option specifies smooth shading. In smooth shading, the color of the polygon is interpolated across its surface using the colors of the vertices. The resulting color transition creates a smoother appearance, giving the illusion of a curved surface.

### glNewList(GLuint list, GLenum mode)

glNewList() function is part of the display list functionality in OpenGL. Display lists are a way to optimize rendering by storing a sequence of OpenGL commands for later execution. The glNewList() function is used to create a new display list and define its content.

The parameters of glNewList() are as follows:

* + list: This parameter specifies the unique identifier for the display list. It is of type GLuint, which is an unsigned integer. You can think of it as an index or handle that allows you to refer to the display list later on. It must be a value greater than zero.
  + mode: This parameter specifies how the display list will be compiled and executed. It is of type GLenum, which is an enumerated type. The possible values are:
* GL\_COMPILE: Specifies that the display list should be compiled only. The OpenGL commands inside the glNewList() and glEndList() block will be recorded into the display list, but not executed immediately.
* GL\_COMPILE\_AND\_EXECUTE: Specifies that the display list should be compiled and immediately executed. The commands inside the glNewList() and glEndList() block will be both recorded and executed.

After calling glNewList(), you can use other OpenGL commands to specify the desired rendering operations within the display list. These commands can include transformations, drawing primitives, texture binding, and other relevant operations. Once you finish defining the display list, you can end it using the glEndList() function.

Display lists can be executed later using the glCallList() function, which takes the list parameter as an argument. This allows you to reuse the compiled commands and improve rendering performance, as the execution of the display list can be faster than issuing individual OpenGL commands.

### gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)

### gluPerspective() is a function in the OpenGL Utility Library (GLU) that is used to set up a perspective projection matrix. The perspective projection is commonly used for rendering 3D scenes, creating a sense of depth and realistic perspective.

### Its parameters are:

### fovy (Field of View):

### Type: GLdouble (double precision floating-point)

### Specifies the field of view angle in the vertical direction, in degrees.

### It defines the extent of the vertical viewing angle, determining how much of the scene is visible. Typical values range from 30 to 60 degrees, depending on the desired perspective effect.

### aspect (Aspect Ratio):

### Type: GLdouble

### Represents the aspect ratio of the viewport (width divided by height).

### It determines the shape of the viewing frustum and ensures that the scene is displayed proportionally regardless of the window dimensions.

### zNear (Near Clipping Plane):

### Type: GLdouble

### Specifies the distance from the viewer (camera) to the near clipping plane. Objects closer to the camera than this distance will be clipped and not rendered. It must be a positive value, typically greater than zero.

### zFar (Far Clipping Plane):

### Type: GLdouble

### Specifies the distance from the viewer (camera) to the far clipping plane.

### Objects farther away from the camera than this distance will be clipped and not rendered.It must be a positive value greater than zNear.

### glLightfv(GLenum light, GLenum pname, const GLfloat\* params)

This function sets various parameters for a specific light source.

light: Specifies the light source to be modified. It can be one of the predefined constants like GL\_LIGHT0, GL\_LIGHT1, etc.

pname: Specifies the parameter to be set. Some commonly used parameters include:

GL\_AMBIENT: Ambient light color.

GL\_DIFFUSE: Diffuse light color.

GL\_SPECULAR: Specular light color.

GL\_POSITION: Light position in homogeneous coordinates.

params: An array of values representing the desired parameter.

**CHAPTER 4**

# TESTING AND RESULTS

## 4.1 DIFFERENT TYPES OF TESTING

### Unit Testing

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

### 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.

### System Testing

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

### Acceptance Testing

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

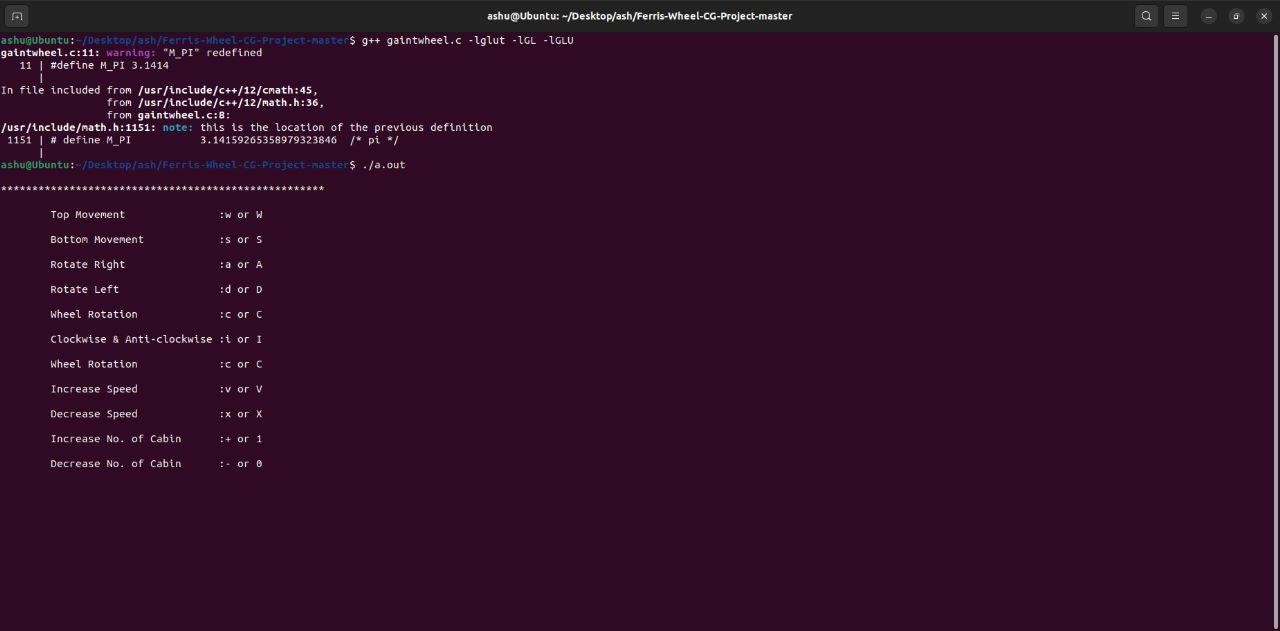
## 4.2 TEST CASES

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

**CHAPTER 5**

# SNAPSHOTS



**Figure 5.1 : Console Window**

**A screenshot of a video game

Description automatically generated**

**Figure 5.2: Giant wheel front view**

A red wheel on a green surface

Description automatically generated

**Figure 5.3 :Backview of Giant wheel**

A screenshot of a video game

Description automatically generated

**Figure 5.4: Giant Wheel top view**

A video game of a red roller coaster

Description automatically generated

**Figure 5.5: Giant Wheel side view**

A screenshot of a video game

Description automatically generated

**Figure 5.6: Giant Wheel with maximum number of Cabins**

A screenshot of a video game

Description automatically generated

**Figure 5.7: Giant Wheel with lowest number of Cabin**

# CONCLUSION

Using OpenGL functions and user defined functions a basic Giant Wheel simulation is being done which is runs at different speed and no. of cabins can also be change. This is basic graphic animationmade 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.

# FUTURESCOPE

In future versions of this project ,addition of some more in-built 2D and 3D models of geometric And also to set light properties according to users requirements is a feasible idea.

Making the users-interface of this program simpler and more user friendly will certainly help beginners in using this program more easily.

In future versions the cones representing cabins can be switched to real life similar cubical cabins.

The ground can be expanded and texture can be added to make it look like grass, mud, sand.

# 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.wikipedia.com](http://www.wikipedia.com) [4].[www.khronos.org](http://www.khronos.org/) [5].[www.github.com](http://www.github.com/)