CHAPTER 1

INTRODUCTION

Computer Graphics is a complex and diversified technology. To understand the technology it is necessary to subdivide it into manageable Parts. This can be accomplished by considering that the end product of computer graphics is a picture. The picture may, of course, be used for a large variety of purposes; e.g., it may be an engineering drawing, an exploded parts illustration for a service manual, a business graph, an architectural rendering for a proposed construction or design project, an advertising illustration, or a single frame from an animated movie. The picture is the fundamental cohesive concept in computer graphics.

Consider how: Pictures are represented in computer graphics.

- Pictures are prepared for presentation.
- Previously prepared pictures are presented.
- Interaction with the picture is accomplished.

Here “picture” is used in its broadest sense to mean any collection of lines, points, text, etc. displayed on a graphics device.

1.1 Computer Graphics

The totality of computer graphics software encompasses the concepts from data structures, from data base design and management, from the psychology, ergonometric of the man-machine interface, from programming languages and operating system.

Numerous computer graphics standards can be grouped following categories.

- First is the graphics application interface, where ideas are translated into a form that is understandable by a computer system. Current representative standards are the GKS, GKS-3D, and the Programmer’s Hierarchical Interactive Graphics Standards (PHIGS).
- The Second is concerned with the storage and transmission of data between graphics systems and between graphics-based computer aided design and computer aided manufacturing systems. The current standard in this area is the Initial Graphics Exchange Specification (IGES).

A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.

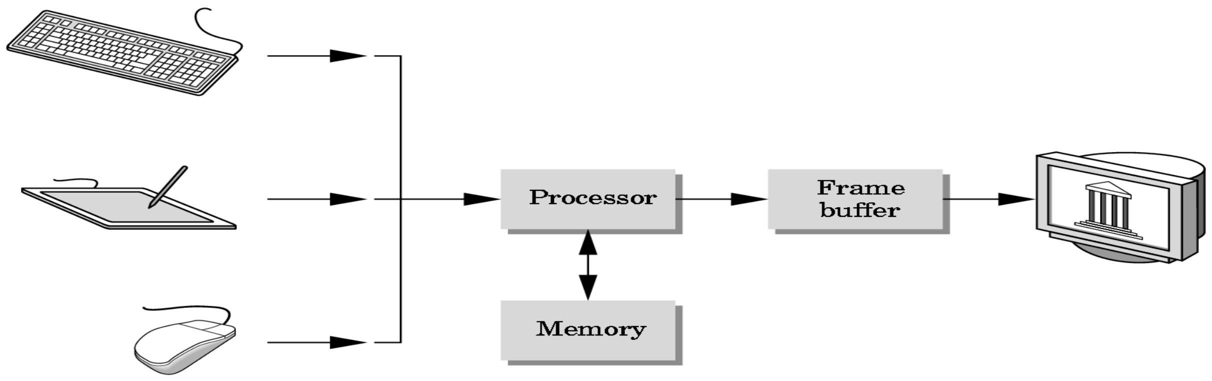

Fig. 1.1: A graphics system

1.2 OpenGL Technology

OpenGL is a graphics application programming interface (API) which was originally developed by Silicon Graphics. OpenGL is not in itself a programming language, like C++, but functions as an API which can be used as a software development tool for graphics applications. The term Open is significant in that OpenGL is operating system independent. GL refers to graphics language. OpenGL also contains a standard library referred to as the OpenGL Utilities (GLU). GLU contains routines for setting up viewing projection matrices and describing complex objects with line and polygon approximations.

OpenGL gives the programmer an interface with the graphics hardware. OpenGL is a low-level, widely supported modelling and rendering software package, available on all platforms. It can be used in a range of graphics applications, such as games, CAD design, modelling.

OpenGL is the core graphics rendering option for many 3D games, such as Quake 3. The providing of only low-level rendering routines is fully intentionalbecause this gives the programmer a great control and flexibility in his applications. These routines can easily be used to build high-level rendering and modelling libraries. The OpenGL Utility Library (GLU) does exactly this, and is included in most OpenGL distributions! OpenGL was originally developed in 1992 by Silicon Graphics, Inc, (SGI) as a multi-purpose, platform independent graphics API. Since 1992 all of the development of OpenGL.

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

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using.

Simplifies Software Development, Speeds Time-to-Market Routines simplify the development of graphics software—from rendering a simple geometric point, line, or filled polygon to the creation of the most complex lighted and texture mapped NURBS curved surface. OpenGL gives software developers access to geometric and image primitives, display lists, modeling transformations,lighting and texturing, anti-aliasing, blending, and many other features. Every conforming OpenGL implementation includes the full complement of OpenGL functions. The well-specified OpenGL standard has language bindings for C, C++, Fortran, Ada, and Java. All licensed OpenGL implementations come from a single specification and language binding document and are required to pass a set of conformance tests. Applications utilizing OpenGL functions are easily portable across a wide array of platforms for maximized programmer productivity and shorter time-to-market.

All elements of the OpenGL state—even the contents of the texture memory and the frame buffer—can be obtained by an OpenGL application. OpenGL also supports visualization applications with 2D images treated as types of primitives that can be manipulated just like 3D geometric objects. As shown in the OpenGL visualization programming pipeline diagram above.

1.3 PROJECT DESCRIPTION:

In this project, we strive to obtain a 3-Dimensional Model from a normal 2D Image. The principle behind the working of the project is that, based on the intensity of the RGB values of a particular pixel, we increase the depth of the object. We achieve this by taking an image of .JPEG or .JPG form and use certain tools to obtain a Standard Template Library (.stl) File. This file format is widely used for 3D Modelling. Resizing is done internally when this process occurs, we specify the width and the depth scaling during the conversion. We specify only the width of the image as the aspect ratio is maintained during conversion.

We use this file and convert the file into an WavefrontObject(.obj) File. Wavefront Object files are mainly used in 3D printing and thus have all the necessary information to render a 3D Model. We use OpenGL for rendering of the object. We achieve this by using the Wavefront Object file as a List. This List contains all the vertices required for rendering the model. We make use of Material and Lighting API of OpenGL to make the model seem better. With all the data on hand we create a 3-Dimensional Quad Mesh of the Image.Wemake use of C with OpenGl for entire coding purpose along with some features of Windows. The OpenGl Utility is a Programming Interface. We use light and material functions to add luster, shade and shininess to graphical objects. The toolkit supports much functionalities like multiple window rendering, callback event driven processing using sophisticated input devices etc.

CHAPTER 2

REQUIREMENTS SPECIFICATION

2.1 Hardware Requirements:

- Intel® Pentium 4 CPU and higher versions
- 128 MB or more RAM.
- A standard keyboard, and Microsoft compatible mouse
- VGA monitor.

2.2 Software requirements:

- The graphics package has been designed for OpenGL; hence the machine must Have Dev C++.
- Software installed preferably 6.0 or later versions with mouse driver installed.
- GLUT libraries, Glut utility toolkit must be available.
- Operating System**:** Windows
- Version of Operating System**:** Windows XP, Windows NT and Higher
- Language**:** C
- Code::Blocks: cross-platform Integrated Development Environment (IDE)

CHAPTER 3

DESIGN

Design of any software depends on the architecture of the machine on which that software runs, for which the designer needs to know the system architecture. Design process involves design of suitable algorithms, modules, subsystems, interfaces etc.

3.1 Architecture:

CONTROL FLOW DIAGRAM

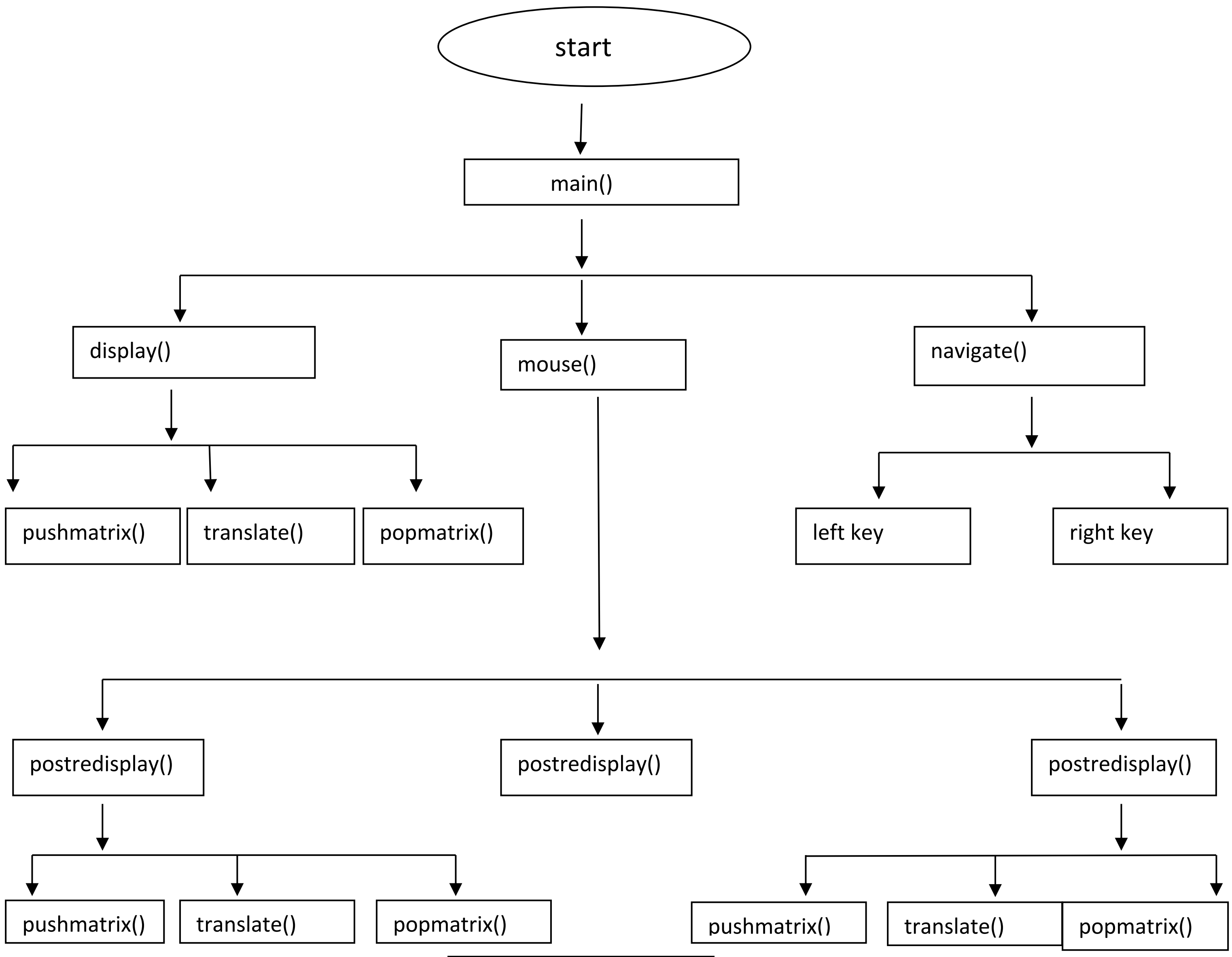

Fig3.1.1-Flowchart

This flowchart explains the functions used in the OpenGL program. We can see that the main function consists of 3 important functions which is further classified. The display() function is further divided into pushmatrix(),translate() and popmatrix(). These functions are used for the appearance of the building. The second function inside the main is mouse(). This mouse function mainly deals with the size of the building, like the zoom in and zoom out feature. The third feature is the feature called navigate(), where in it is used to navigate the building. This is the function where we can move the building and take a look at every view.

3.2 Initialization

This function is the initial stage of the system where the system initializes the various aspects of the graphics system based on the user requirements, which include Command line processing, window system initialization and also the initial window creation state is controlled by these routines.

3.3 Event Processing

This routine enters GLUT's event processing loop. This routine never returns, and it continuously calls GLUT callback as and when necessary. This can be achieved with the help of the callback registration functions. These routines register callbacks to be called by the GLUT event processing loop.

CHAPTER 4

IMPLEMENTATION

Step 1: The given 2d image must first be converted into an STL format image so as to provide the depth for the given image.

Step 2: The STL format image is then converted into an object file and then it is provided to the program to perform the required operations to display the 3D object.

4.1 Pseudo Code

//1.The header files are:

```
#include<windows.h>
 #include <GL/glut.h>
 #include <stdlib.h>
```

//2. The code for processing object files are:

```
        double winHt=4.0;
        float theta[5]={0};
        void  leaf()
        {
        //this function is used to create a leaf structure on the building
        }
        void leafseries()
        {
        // this function is used to create multiple leaf structures
        }
        void drawtop()
        {
        // this function is used to create the top of the building
        }
        void octtop()
        {
```

```
//this function is used to create the top of the building
}
void drawdoor()
{
//this function is used to create a door for the building
}
void drawdoor1()
{
//this function is to draw a door
}
void octagon()
{
// this function is for the shape of the building
}
void baseoctagon()
{
//this is the base of the building
}
void drawedge()
{
// this function is to draw the edges of the building
}
```

1.4 OpenGl Functions Used:

This project is developed using CodeBlocks and this project is implemented by making extensive use of library functions offered by graphics package of OpenGl, a summary of those functions follows:

1. glBegin() :

Specifies the primitives that will be created from vertices presented between glBegin and subsequent glEnd. GL_POLYGON, GL_LINE_LOOP etc.

2. glEnd(void):

It ends the list of vertices.

3. glPushMatrix() :

*‘ void**glPushMatrix**(void)’*

glPushMatrix pushes the current matrix stack down by one level, duplicating the current matrix.

4. glPopMatrix() :

*‘void**glPopMatrix(**void)’*

glPopMatrix pops the top matrix off the stack, destroying the contents of the popped matrix. Initially, each of the stacks contains one matrix, an identity matrix.

5. glTranslate() :

‘*void**glTranslate**(GLdouble x, GLdouble y, GLdouble z)’*

Translation is an operation that displaces points by a fixed distance in a given direction. *Parameters x*, *y*, *z* specify the *x*, *y*, and *z* coordinates of a translation vector. Multiplies current matrix by a matrix that translates an object by the given x, y and z-values.

6. glClear() :

*‘void**glClear**(GLbitfield mask)’*

glClear takes a single argument that is the bitwise *or* of several values indicating which buffer is to be cleared. GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_ACCUM_ BUFFER_BIT, and GL_STENCIL_BUFFER_BIT.Clears the specified buffers to their current clearing values.

7.glClearColor() :

‘*void**glClearColor**(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)’*

Sets the current clearing color for use in clearing color buffers in RGBA mode. The red, green, blue, and alpha values are clamped if necessary to the range [0,1]. The default clearing color is (0, 0, 0, 0), which is black.

8. glMatrixMode() :

*‘void**glMatrixMode**(GLenum mode)’*

It accepts three values GL_MODELVIEW, GL_PROJECTION and GL_TEXTURE. It specifies which matrix is the current matrix. Subsequent transformation commands affect the specified matrix.

9. glutInitWindowPosition() :

*‘void**glutInitWindowPosition**(int x, int y);’*

This API will request the windows created to have an initial position. The arguments x, y indicate the location of a corner of the window, relative to the entire display.

10. glLoadIdentity() :

*‘void**glLoadIdentity**(void);’*

It replaces the current matrix with the identity matrix.

11. glutInitWindowSize() :

‘*void**glutInitWindowSize**(int width, int height);’*

The API requests windows created to have an initial size. The arguments width and height indicate the window’s size (in pixels). The initial window size and position are hints and may be overridden by other requests.

12.glutInitDisplayMode

*'void **glutInitDisplayMode**(unsigned int mode);'*

Specifies the display mode, normally the bitwise OR-ing of GLUT display mode bit *masks.*This API specifies a display mode (such as RGBA or color-index, or single or double-buffered) for windows.

13.glFlush () :

***'**void**glFlush**(void);'*

The glFlush function forces execution of OpenGL functions in finite time.

14. glutCreateWindow() :

'*int**glutCreateWindow**(char *name);'*

The parameter *name* specifies any name for window and is enclosed in double quotes. This opens a window with the set characteristics like display mode, width, height, and so on. The string name will appear in the title bar of the window system. The value returned is a unique integer identifier for the window. This identifier can be used for controlling and rendering to multiple windows from the same application.

15. glutDisplayFunc() :

'*void**glutDisplayFunc**(void (*func)(void))'*

Specifies the new display callback function. The API specifies the function that's called whenever the contents of the window need to be redrawn. All the routines need to be redraw the scene are put in display callback function.

16. glVertex2f

*‘void**glVertex2f**(GLfloatx,GLfloat y);’*

x Specifies the x-coordinate of a vertex.

y Specifies the y-coordinate of a vertex.

The glVertex function commands are used within glBegin/glEnd pairs to specify point, line, and polygon vertices. The current color, normal, and texture coordinates are associated with the vertex when glVertex is called. When only x and y are specified, z defaults to 0.0 and w defaults to 1.0. When x, y, and z are specified, w defaults to 1.0.

17. glColor3f

‘void glColor3f(GLfloat red, GLfloat green, GLfloat blue);’

PARAMETERS:

1. Red: The new red value for the current color.
2. Green: The new green value for the current color.
3. Blue: The new blue value for the current color.

Sets the current color.

18. glRotate():

‘voidglRotate(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);’

PARAMETERS:

angle: The angle of rotation, in degrees.

x: The x coordinate of a vector.

y: The y coordinate of a vector.

z: The z coordinate of a vector.

The glRotated and glRotatef functions multiply the current matrix by a rotation matrix.

19. gluPerspective():

voidgluPerspective(GLdoublefovy, GLdouble aspect, GLdoublezNear, GLdoublezFar);

PARAMETERS:

fovy : Specifies the field of view angle, in degrees, in the y direction.

aspect: Specifies the aspect ratio that determines the field of view in the x direction.
The aspect ratio is the ratio of x (width) to y (height).

zNear: Specifies the distance from the viewer to the near clipping plane (always positive).

zFar : Specifies the distance from the viewer to the far clipping plane (always positive).

Sets up a perspective projection matrix.

20. glMaterialfv():

' voidglMaterialfv (GLenum face, GLenumpname, constGLfloatparams);'

PARAMETERS:

face : The face or faces that are being updated. Must be one of the following: GL_FRONT, GL_BACK, or GL_FRONT and GL_BACK.

Pname: The material parameter of the face or faces being updated.
The parameters that can be specified using glMaterialfv, and their interpretations by the lighting equation, are as follows.
GL_SPECULAR: The parameter contains four integer or floating-point values that specify the seculars RGBA reflectance of the material. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative represent able value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default specular reflectance for both front-facing and back-facing materials is (0.0, 0.0, 0.0, 1.0).
The glMaterialfv function specifies material parameters for the lighting model.

21. glutInit():

PARAMETERS:

*glutInit(int *argcp, char **argv);*

argcp : A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.

argv : The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

glutInit(&argc,argv);

glutInit is used to initialize the GLUT library.

22. glutMainLoop ()

'void*glutMainLoop(void); glutMainLoop();'*
glutMainLoop enters the GLUT event processing loop.

23. glLightfv():

*'voidglLightfv(GLenum light, GLenumpname, GLfloat *params);'*

The glLightfv function returns light source parameter values.

PARAMETERS:

Light: The identifier of a light. The number of possible lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form GL_LIGHTi where i is a value: 0 to GL_MAX_LIGHTS - 1.

Pname: A light source parameter for light. The following symbolic names are accepted:

GL_DIFFUSE: The params parameter contains four integer or floating-point values that specify the diffuse RGBA intensity of the light. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative represent able value maps to 1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default diffuse intensity is (0.0, 0.0, 0.0, 1.0) for all lights other than light zero. The default diffuse intensity of light zero is (1.0, 1.0, 1.0, 1.0).

GL_SPECULAR: The params parameter contains four integer or floating-point values that specify the specular RGBA intensity of the light. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative represent able value maps to 1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default specular intensity is (0.0, 0.0, 0.0, 1.0) for all lights other than light zero. The default specular intensity of light zero is (1.0, 1.0, 1.0, 1.0).

GL_AMBIENT: The params contains four integer or floating-point values that specify the ambient RGBA intensity of the light. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative representable value maps to -1.0 . Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial ambient light intensity is (0,0, 0, 1).

glLightfv(GL_LIGHT1,GL_POSITION,pos);

24. glEnable():

‘voidglEnable(GLenum cap);’

‘glEnable(GL_CULL_FACE);’

PARAMETERS:

cap: A symbolic constant indicating an OpenGL capability.

CHAPTER 5

RESULTS & SNAPSHOTS

The final output of the project:

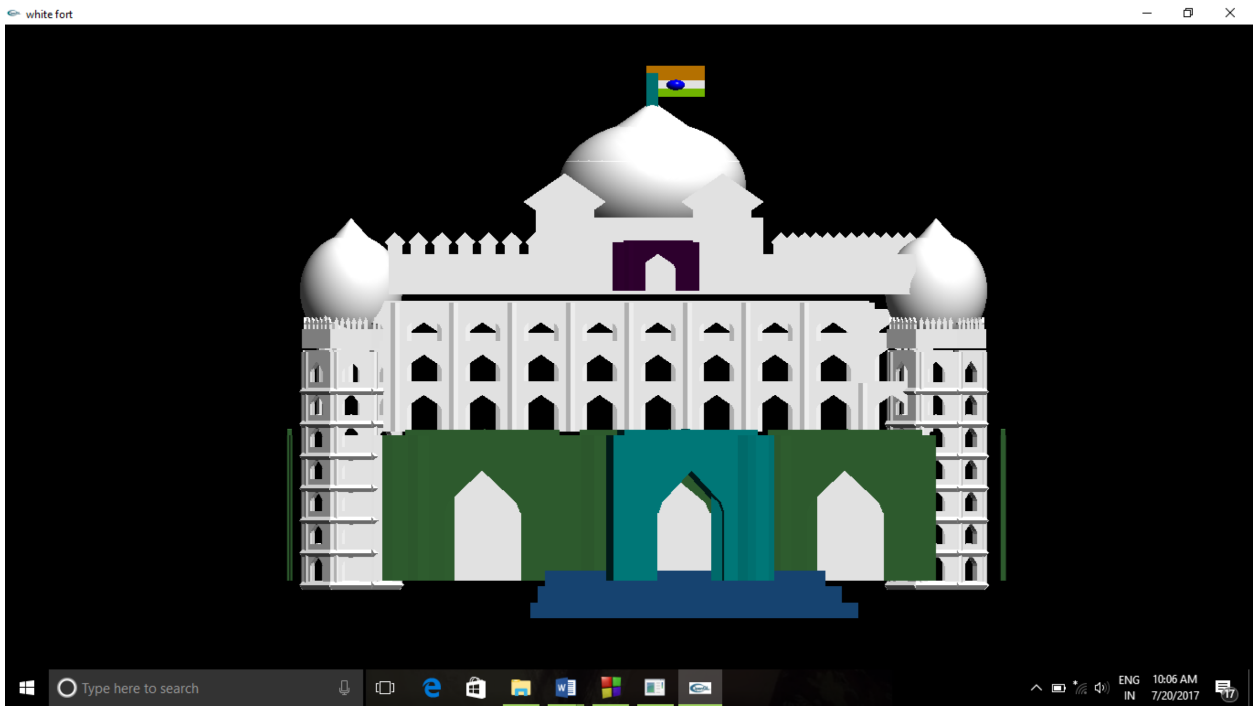

Fig 5.1.Front view of Vidhana Soudha

This snapshot mainly shows us the final output of this project. This is the front view of Vidhana Soudha. Here, we can see the leaf structures on the top and door in the front section of the building.

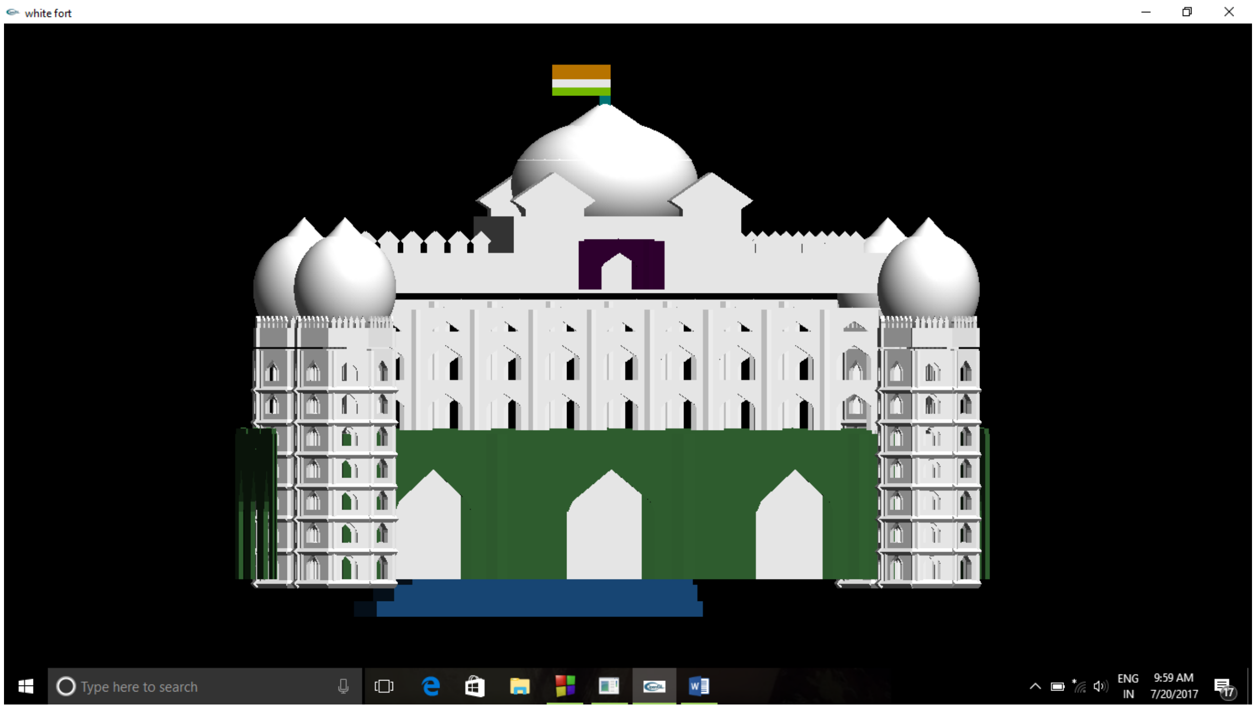

Fig5.2.Back view of Vidhana Soudha

This is the backview of the project. This view mainly tells us about the back portion of the building. This portion also gives us the details like, the back doors, the flag hoisted on the top of the building and the leaf structures in it.

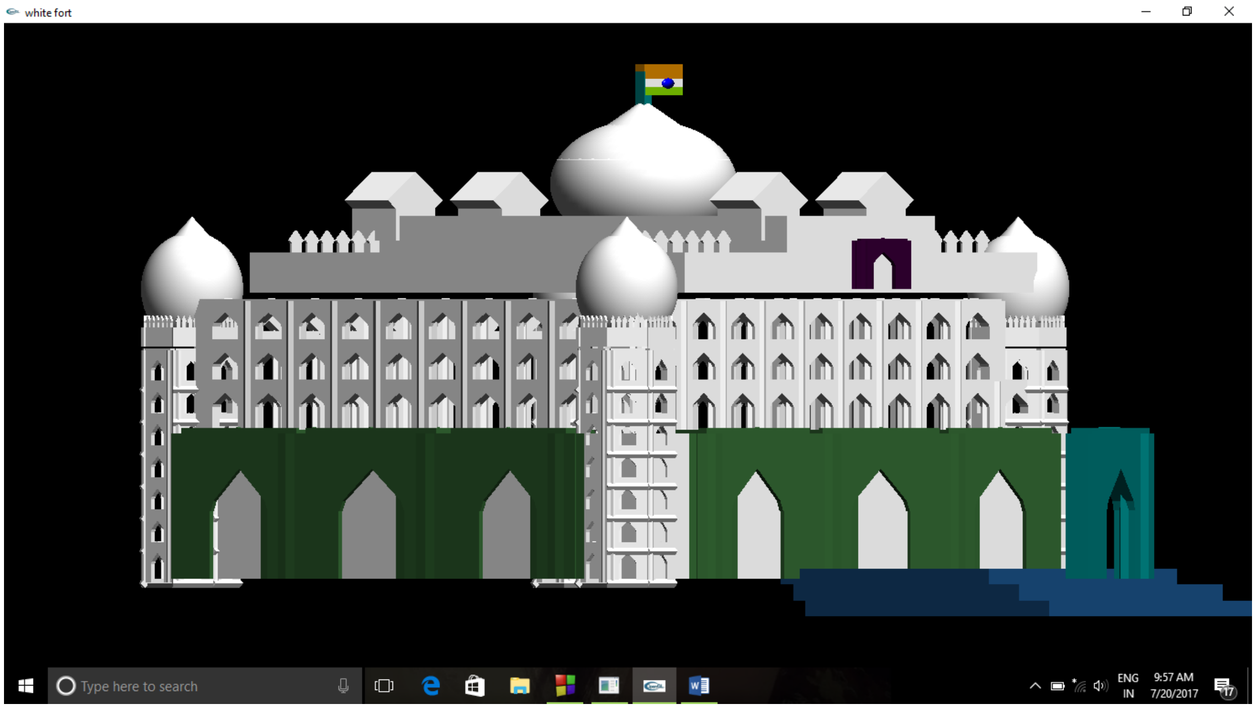

Fig5.3.Side view of Vidhana Soudha

This snapshot gives us the information on the side view of the project. The side view gives us the information on the wall structures, the leaf structures on the top of the building. The side view gives the details about the back and the adjacent side's view.

CONCLUSION & FUTURE ENHANCEMENT

Conclusion

Our project aims at converting a 2D image to a 3D object in OpenGL. Using built in functions provided by graphics library and integrating it with the C implementation of list it was possible to visually represent the vertices and faces of a 3D object.

The described project demonstrates the power of Viewing which is implemented using different modes of viewing. The lighting and material functions of OpenGl library add effect to the objects in animation.

The aim in developing this program was to design a simple program using Open GL application software by applying the skills we learnt in class, and in doing so, to understand the algorithms and the techniques underlying interactive graphics better.

The designed program will incorporate all the basic properties that a simple program must possess.

The program is user friendly as the only skill required in executing this program is the knowledge of graphics.

Future Enhancement

- We can try to add actual color to the object drawn in this program.
- We can try to improve the quality of the object drawn.

REFERENCES

Books:

Interactive Computer Graphics, 5th Edition, Edward Angel

Computer Graphics and Multimedia, Udit Sharma

Websites:

- http://www.amazon.in/Computer-Graphics-Multimedia-Udit-Agarwal/dp/935014316X?tag=googinhydr18418-21
- http://en.wikipedia.org/wiki/Computer_graphics
- http://stackoverflow.com/questionsquickly
- http://www.opengl-tutorial.org/intermediate-tutorials/
- http://www.opengl-tutorial.org/
- https://open.gl/
- http://www.cs.uccs.edu/~ssemwal/indexGLTutorial.html
- http://www.videotutorialsrock.com/
- http://ogldev.atspace.co.uk/
- https://www.opengl.org/sdk/docs/tutorials/
- http://learnopengl.com/
- http://lazyfoo.net/tutorials/OpenGL/
- http://en.wikibooks.org/wiki/OpenGL_Programming