VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi – 590018

COMPUTER GRAPHICS MINI PROJECT REPORT

on

"WHITE FORT"

Submitted in partial fulfilment for the award of degree of

Bachelor of Engineering

in

Computer Science and Engineering

Submitted by

GAGAN R	**(1ST20CS039)**
HARIPRASAD BP	**(1ST20CS043)**

Under the Guidance of

Prof. Mohanapriya M
Assistant professor
Department of CSE
SaIT, Bengaluru

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SAMBHRAM INSTITUTE OF TECHNOLOGY

M. S. Palya, Bengaluru – 560097

2022-2023

SAMBHRAM INSTITUTE OF TECHNOLOGY

M. S. Palya, Bengaluru – 560097

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the Mini Project work entitled "**WHITE FORT**" carried out by Mr. **GAGAN R (1ST20CS039) AND HARIPRASAD BP (1ST20CS043)**, a bonafide student of **SAMBHRAM INSTITUTE OF TECHNOLOGY** in partial fulfilment for the award of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING** of **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**, Belgaum during the year **2022-2023**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The Mini Project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the said Degree.

Prof.Mohanapriya M	**Dr. T. John Peter**
Assistant professor	HOD
Dept. of CSE	Dept. of CSE
SaIT, Bengaluru	SaIT, Bengaluru

Signatures of the Examiners

1

2

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this mini project work. We would like to take this opportunity to thank them all.

We would like to thank **Dr. T. John Peter,** Prof. & Head, Department of Computer Science & Engineering, SAIT, Bangalore, for his valuable suggestions and expert advice.

We deeply express our sincere gratitude to our guide **Mohanapriya M, Assistant professor** Department of CSE, SAIT, Bangalore, for his able guidance , regular source of encouragement and assistance throughout this mini project work.

We would like to thank all the teaching and non-teaching staff of Department of Computer Science & Engineering SAIT, Bangalore for their constant support and encouragement.

Date:
Place: Bengaluru

GAGAN R
1ST20CS039
HARIPRASAD BP
1ST20CS043

ABSTRACT

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.

We make 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

TABLE OF CONTENT

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.

1.1 Computer Graphics

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

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).

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 intentional because 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.

CHAPTER 2

OPEN GL FUNCTIONS

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 dentifier 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():

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

PARAMETERS:

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.

CHAPTER 3

REQUIREMENT ANALYSIS

HARDWARE REQUIREMENTS:

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

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)

MISCELLANEOUS REQUIREMENTS:

All the required library and header files should be available in the include directories. The files associated with this editor should be placed in either the same folder or in a specified folder.

LANGUAGE USED IN CODING:

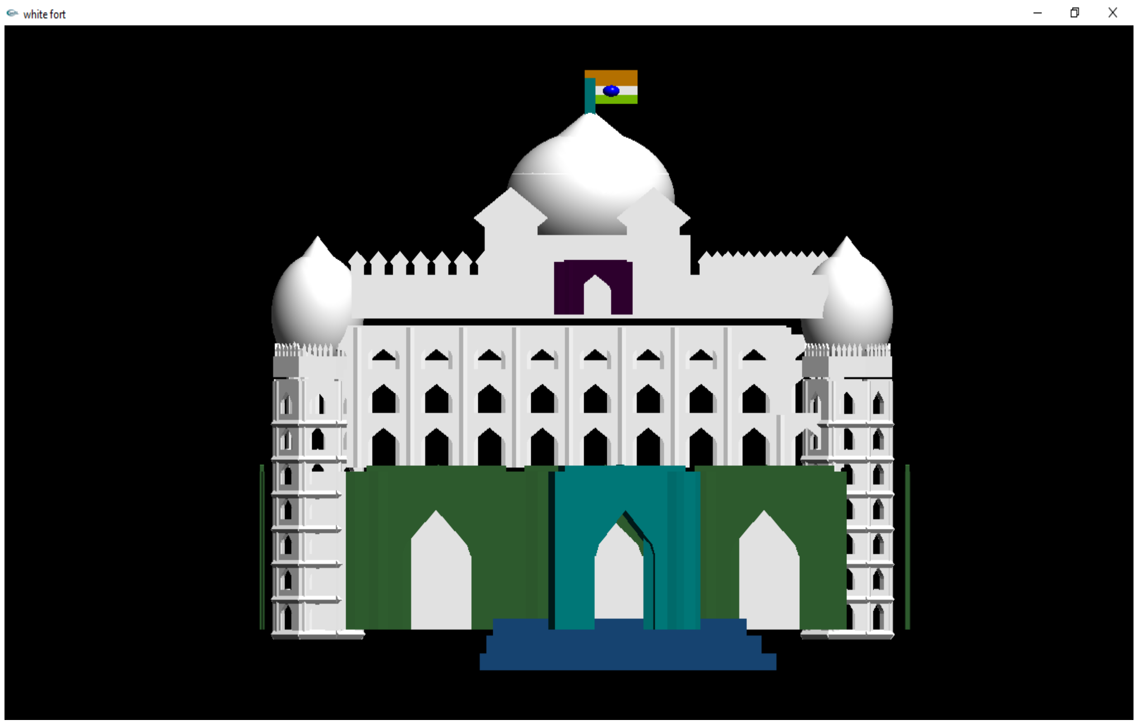
C/C++ and OpenGL as an API.

CHAPTER 4

SNAPSHOTS

The final output of the project:

Fig 4.1 Front view



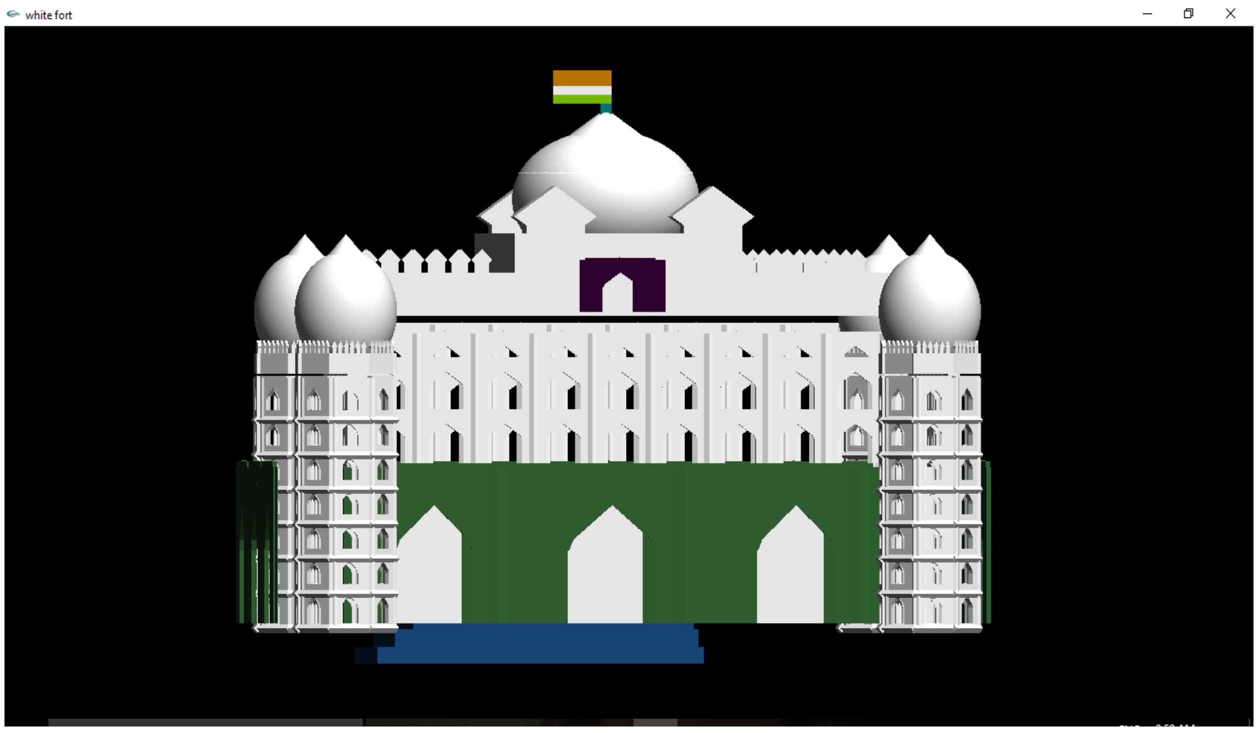


Fig 4.2 Back view

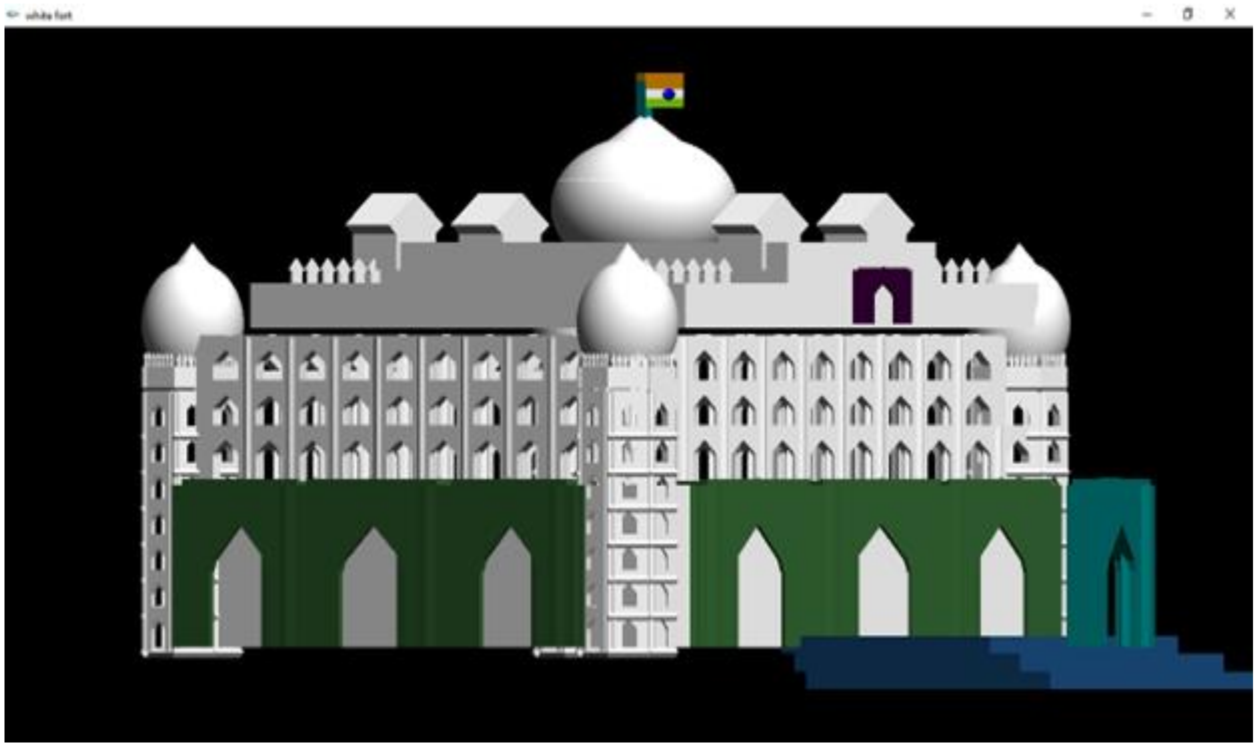


Fig 4.3 Side view

CHAPTER 5

ADVANTAGES OR FUTURE ENHANCEMENT

1.Realism: Computer graphics allows for highly detailed and realistic 3D representations. By creating a white fort object, you can accurately simulate the architectural features, textures, and lighting conditions of a real fort. This level of realism can be beneficial for various applications, such as video games, movies, architectural visualization, or virtual tourism.

2. Visualization and Design: Computer graphics enables designers and architects to visualize and iterate on their ideas more effectively. By creating a 3D model of a white fort, designers can explore different angles, perspectives, and designs, making it easier to refine the overall structure and aesthetics. It allows them to experiment with various architectural elements and assess their impact on the final outcome.

3. Preservation and Reconstruction: In cases where a fort may be damaged, destroyed, or no longer exists, computer graphics can play a vital role in preserving and reconstructing its appearance. By creating a virtual representation of a white fort based on historical data and references, it becomes possible to study, document, and share the fort's architectural and historical significance.

4. Educational Purposes: Computer graphics can be an excellent educational tool. By creating a virtual white fort, it becomes possible to provide interactive and immersive learning experiences. Students can explore the fort's layout, learn about its history, and understand the strategic elements involved in fortification. This visual approach can enhance comprehension and engagement, particularly for complex subjects like history and architecture.

5. Cost and Time Efficiency: Building physical models of forts can be time-consuming and expensive. Computer graphics provides a cost-effective and efficient alternative. Creating a white fort object digitally saves time, materials, and resources. Additionally, changes and modifications can be made easily and quickly without the need to rebuild physical structures.

6. Entertainment and Media: Computer graphics are extensively used in the entertainment industry, including video games, movies, and television shows. A white fort object can be incorporated into historical or fantasy narratives, serving as a backdrop for engaging stories and captivating visuals. It adds depth and richness to the visual experience, enhancing the overall immersion and entertainment value.

Overall, computer graphics offers numerous advantages when creating a white fort object. It enables realism, facilitates design exploration, aids in preservation and reconstruction efforts, enhances education, provides cost and time efficiency, and contributes to entertainment and media production.

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.

BIBLIOGRAPHY

Books:

Interactive Computer Graphics, 5th Edition, Edward Angel

Computer Graphics and Multimedia, Udit Sharma

Websites:

- http://www.amazon.in/Computer-Graphics-Multimedia-UditAgarwal/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