Table of Contents

List of Figures

ABSTRACT

This project aims to develop a realistic butterfly simulation using the OpenGL graphics library. The objective is to create a visually appealing and interactive 3D environment that simulates the flight patterns and behaviours of butterflies.

The simulation will include a variety of butterfly species, each characterized by their unique wing patterns, colours, and flight behaviours. The project will leverage OpenGL's capabilities for rendering realistic textures, lighting effects, and smooth animations to bring the butterflies to life. The outcome of this project will be a fully functional and visually captivating butterfly simulation, demonstrating the capabilities of OpenGL for creating realistic graphics and interactive experiences. This simulation can serve as a foundation for educational purposes, artistic representations, or entertainment applications related to the study of butterflies and their behaviour in virtual environments.

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project.

I would like to thank **Shri. Narayan Rao R Maanay**, Secretary, BNMEI, Bengaluru for providing the excellent environment and infrastructure in the college.

I would like to sincerely thank **Prof. T J Rama Murthy**, Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

I would like to sincerely thank **Dr. S Y Kulkarni** ,Additional Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

I would like to express my gratitude to **Prof. Eishwar N Maanay**, Dean, BNMIT, Bengaluru for his relentless support and encouragement.

I would like to thank **Dr. Krishnamurthy G N**, Principal, BNMIT, Bengaluru for his constant encouragement.

I would like to thank, **Dr. Chayadevi M L**, Professor & Head of the Department of Computer Science and Engineering for the encouragement and motivation she provides.

I would also like to thank **Prof. Akshitha Katkeri,** Assistant Professor, Department of Computer Science and Engineering for providing me with her valuable insight and guidance wherever required throughout the course of the project and its successful completion.

Afshan Tabassum 1BG20CS006

Anusha A 1BG20CS017

Chapter 1

INTRODUCTION

1.1 Overview

Interactive computer graphics provides us with the most natural means of communicating information through a computer. Over the years advancements in computer graphics have enabled us to not only make pictures of real-world objects but also visualize abstract, synthetic objects such as mathematical surfaces and of data that have no inherent geometry, such as survey results. Some topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. The computer on receiving signals from the input device can modify the displayed picture appropriately. To the user it appears that the picture is changing instantaneously in response to his commands.

1.2 Problem Statement

The aim of this application is to show a basic implementation of 3D Butterfly. The application will be implemented using the C programming language and the OpenGL API. The objective of the application is to demonstrate 3D butterfly operations using translation , texture and rotations. The application will also include user interaction through keyboard events; for animation, rotations and translation.

1.3 Motivation

The motivation for the 3D Butterfly project using OpenGL lies in the desire to combine the beauty of nature with the immersive capabilities of computer graphics. By creating a virtual environment where users can interact with a realistic and captivating butterfly, we aim to provide a unique and enchanting experience. This project aims to showcase the power of OpenGL in rendering intricate details, vibrant colours, and smooth animations, bringing the delicate movements and intricate patterns of a butterfly to life..

1.4 Computer Graphics

Computer graphics and multimedia technologies are becoming widely used in educational applications because they facilitate non-linear, self-learning environments that particularly suited to abstract concepts and technical information.

Computer graphics are pictures and films created using computers. Usually,the term refers to computer-generated image data created with help from specialized graphical hardware and software. It is a vast and recent area in computer science. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, though sometimes erroneously referred to as CGI. Important topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others.

The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the user. It is also used for processing image data received from the physical world. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design generally.

1.5 OpenGL API

Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware- accelerated rendering. Silicon Graphics Inc., (SGI) began developing OpenGL in 1991 and released it on June 30, 1992; applications use it extensively in the fields of computer- aided design (CAD), virtual reality, scientific visualization, information visualization,

flight simulation, and video games. Since 2006 OpenGL has been managed by the non- profit technology consortium Khronos Group.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware. The API is defined as a set of functions which may be called by the client program, alongside a set of namedinteger constants. In addition to being language-independent, OpenGL is also cross-platform.

Given that creating an OpenGL context is quite a complex process, and given that it varies between operating systems, automatic OpenGL context creation has become a common feature of several game-development and user-interface libraries, including SDL, Allegro, SFML, FLTK, and Qt. A few libraries have been designed solely to produce an OpenGL-capable window. The first such library was OpenGL Utility Toolkit (GLUT), later superseded by freeglut. GLFW is a newer alternative.

1.5.1 OpenGL API Architecture

Display Lists:

All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators:

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions.

Per Vertex Operations:

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D

world to a position on your screen.

Pixel Operation:

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped andthen either written into texture memory or sent to the rasterization step.

Rasterization:

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Colour and depth values are assigned for each fragment square.

Fragment Operations:

Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

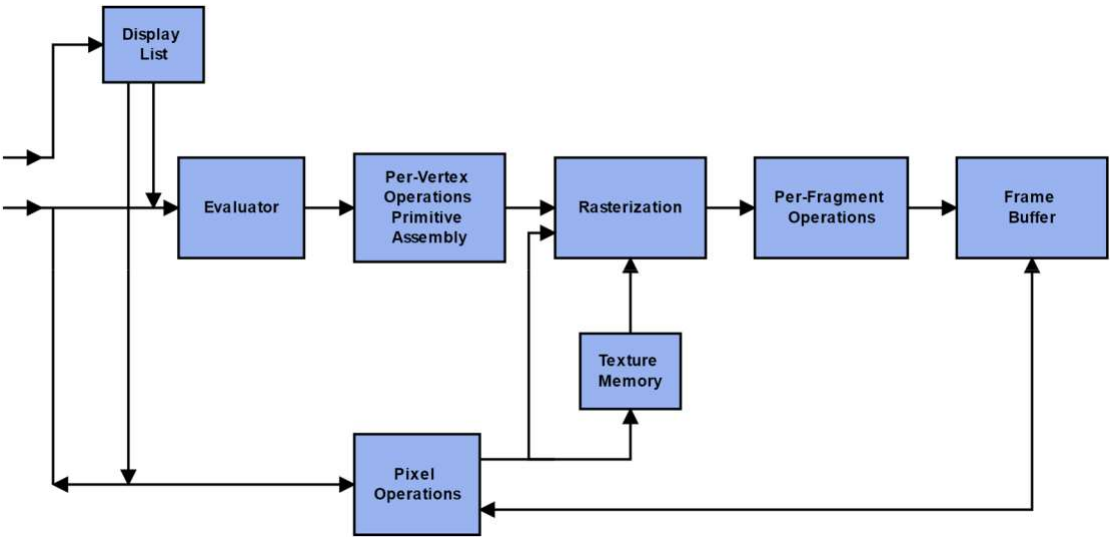

Fig 1.1 An illustration of the graphics pipeline process in OpenGL Architecture

1.6 Applications of Computer Graphics

Although many applications span two, three, or even all of these areas, the development of the field was based, for the most part, on separate work in each domain.We can classify applications of computer graphics into four main areas:

1.6.1 Display of Information

Graphics has always been associated with the display of information. Examples of the use of orthographic projections to display floorplans of buildings can be found on 4000-year-old Babylonian stone tablets. Mechanical methods for creating perspective drawings were developed during the Renaissance. Countless engineering students have become familiar with interpreting data plotted on log paper. More recently, software packages that allow interactive design of charts incorporating colour, multiple data sets, and alternate plotting methods have become the norm.

1.6.2 Design

Professions such as engineering and architecture are concerned with design. Although their applications vary, most designers face similar difficulties and use similar methodologies. One of the principal characteristics of most design problems is the lack of a unique solution. Hence, the designer will examine a potential design and then will modify it, possibly many times, in an attempt to achieve a better solution. Computer graphics has become an indispensable element in this iterative process.

1.6.3 Simulation

Some of the most impressive and familiar uses of computer graphics can be classified as simulations. Video games demonstrate both the visual appeal of computer graphics and our ability to generate complex imagery in real time. Computer-generated images are also the heart of flight simulators, which have become the standard method for training pilots.

1.6.4 User Interfaces

The interface between the human and the computer has been radically altered by the use of computer graphics. Consider the electronic office. The figures in this book were produced through just such an interface. A secretary sits at a workstation, rather than at a desk equipped with a typewriter. This user has a pointing device, suchas a mouse, that allows him to communicate with the workstation.

Chapter 2

LITERATURE SURVEY

2.1 History of Computer Graphics

The term "computer graphics" was coined in 1960 by William Fetter, a designer at Boeing, to describe his own job, the field can be said to have first arrived with the publication in 1963 of Ivan Sutherland's Sketchpad program, as part of his Ph.D. thesis at MIT. Sketchpad, as its name suggests, was a drawing program. Beyond the interactive drawing of primitives such as lines and circles and their manipulation – in particular, copying, moving and constraining – with use of the then recently invented light pen, Sketchpad had the first fully-functional graphical user interface (GUI) and the first algorithms for geometric operations such as clip and zoom. Interesting, as well, is that Sketchpad's innovation of an object-instance model to store data for geometric primitives foretold object-oriented programming. Coincidentally, on the hardware side, the year 1963 saw the invention by Douglas Engelbart at the Stanford Research Institute of the mouse, the humble device even today carrying so much of GUI on its thin shoulders.

Subsequent advances through the sixties came thick and fast: raster algorithms, the implementation of parametric surfaces, hidden-surface algorithms and the representation of points by homogeneous coordinates, the latter crucially presaging the foundational role of projective geometry in 3D graphics, to name a few. Flight simulators were the killer app of the day and companies such as General Electric and Evans & Sutherland, 6 co-founded by Douglas Evans and Ivan Sutherland, wrote simulators with real-time graphics.

The seventies, brought the Z-buffer for hidden surface removal, texture mapping, Phong's lighting model – all crucial components of the OpenGL API (Application Programming Interface) we'll be using soon – as well as keyframe-based animation. Photorealistic rendering of animated movie keyframes almost

invariably deploys ray tracers, which were born in the seventies too.

Through the nineties, as well, the use of 3D effects in movies became pervasive. The Terminator and Star Wars series, and Jurassic Park, were among the early movies to set the standard for CGI. Toy Story from Pixar, 8 released in 1995, has special importance in the history of 3D CGI as the first movie to be entirely computer- generated – no scene was ever pondered through a glass lens, nor any recorded on a photographic reel! It was cinema without film. Quake, released in 1996, the first of the hugely popular Quake series of games, was the first fully 3D game.

Another landmark from the nineties of particular relevance to us was the release in 1992 of OpenGL, the open-standard cross-platform and cross language 3D graphics API, by Silicon Graphics. OpenGL is actually a library of calls to perform 3D tasks, which can be accessed from programs written in various languages and running over various operating systems. That OpenGL was high-level (in that it frees the applications programmer from having to care about such low-level tasks as representing primitives like lines and triangles in the raster, or rendering them to the window) and easy to use (much more so than its predecessor 3D graphics API, PHIGS, standing for Programmer's Hierarchical Interactive Graphics System) first brought 3D graphics programming to the "masses". What till then had been the realm of a specialist was now open to a casual programmer following a fairly amicable learning curve.

The story of the past decade has been one of steady progress, rather than spectacular innovations in CG. Hardware continues to get faster, better, smaller and cheaper, continually pushing erstwhile high-end software down market, and raising the bar for new products. The almost complete displacement of CRT monitors by LCD and the emergence of high-definition television are familiar consequences of recent hardware evolution.

2.2 Related Work

- **Computer Aided Design (CAD):**

Most of engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircrafts, Building, in almost all kinds of Industries. Its use in designing electronic systems is known as electronic design automation (EDA). In mechanical design it is known as mechanical design automation (MDA) or computer-aided drafting (CAD), which includes the process of creating a technical drawing with the use of computer software.

- **Computer Simulation**

Computer simulation is the reproduction of the behavior of a system using a computer to simulate the outcomes of a mathematical model associated with said system. Since they allow to check the reliability of chosen mathematical models, computer simulations have become a useful tool for the mathematical modeling of many natural systems in physics (computational physics), astrophysics, climatology, chemistry, biology and manufacturing, human systems in economics, psychology, social science, health care and engineering. Simulation of a system is represented as the running of the system's model. It can be used to explore and gain new insights into new technology and to estimate the performance of systems too complex for analytical solutions.

- **Digital Art**

Digital art is an artistic work or practice that uses digital technology as part ofthe creative or presentation process. Since the 1970s, various names have been used to describe the process, including computer art and multimedia art. Digital art is itself placed under the larger umbrella term new media art. With the rise of social media and the internet, digital art application of computer graphics. After some initial resistance, the impact of digital technology has transformed activities such as painting, drawing, sculpture and music/sound art, while new

forms, such as net art, digital installation art, and virtual reality, have become recognized artistic practices. In an expanded sense, "digital art" is contemporary art that uses the methods of mass production or digital media.

- **Virtual Reality**

Virtual reality (VR) is an experience taking place within a computer-generated reality of immersive environments can be similar to or completely different from the real world. Applications of virtual reality can include entertainment (i.e. gaming) and educational purposes (i.e. medical or military training). Other, distinct types of VR style technology include augmented reality and mixed reality. Currently standard virtual reality systems use either virtual reality headsets or multi-projected environments to generate realistic images, sounds and other sensations that simulate a user's physical presence in a virtual environment. The effect is commonly created by VR headsets consisting of a head- mounted display with a small screen in front of the eyes, but can also be created through specially designed rooms with multiple large screens. Virtual reality typically incorporates auditory and video feedback, but may also allow other types of sensory and force feedback through haptic technology.

- **Video Games**

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a two- or three-dimensional video display device such as a TV screen, virtual reality headset or computer monitor. Since the 1980s, video games have become an increasingly important part of the entertainment industry, and whether they are also a form of art is a matter of dispute. The electronic systems used to play video games are called platforms. Video games are developed and released for one or several platforms and may not be available on others. These include dedicated devices such as video game consoles, as well as general-purpose computers like a laptop, desktop or handheld computing devices.

Chapter 3

SYSTEM REQUIREMENTS

3.1 Software Requirements

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

The following are the software requirements for the application :

- Operating System: Windows 10
- Compiler: GNU C/C++ Compiler
- Development Environment: Visual Studio 2019 Community Edition
- API: OpenGL API & Win32 API for User Interface and Interaction

3.2 Hardware Requirements

The most common set of requirements defined by any operating system orsoftware application is the physical computer resources, also known as hardware.

- CPU: Intel or AMD processor
- Cores: Dual-Core (Quad-Core recommended)
- RAM: minimum 4GB (>4GB recommended)
- Graphics: Intel Integrated Graphics or AMD Equivalent
- Secondary Storage: 250GB
- Display Resolution: 1366x768 (1920x1080 recommended)

Chapter

SYSTEM DESIGN

4.1 Proposed System

The proposed system for the 3D Butterfly project using OpenGL involves several key components. Firstly, we will utilize OpenGL, a powerful graphics library, to handle the rendering of the 3D butterfly model. This will enable us to leverage OpenGL's capabilities in generating realistic lighting, shading, and texturing effects, bringing the butterfly to life with stunning visual fidelity.

Secondly, the system will incorporate user interaction through input devices such as a mouse or keyboard. Users will be able to control the butterfly's flight path, speed, and direction, allowing for an interactive and engaging experience. This will be achieved by mapping user inputs to appropriate transformations and animations within the OpenGL rendering pipeline.

Lastly, the proposed system will provide options for customization, enabling users to modify the appearance of the butterfly, its wing patterns, colours, and other visual attributes. This will allow users to personalize their experience and create unique variations of the butterfly.

By integrating these components, the 3D Butterfly project using OpenGL will deliver an immersive and interactive simulation of a butterfly in a visually captivating environment, providing users with a rich and engaging experience that showcases the power and versatility of OpenGL in computer graphics applications.

Implement the butterfly animation by applying transformation matrices to the wings, allowing for rotation and translation to achieve the desired animation effect. Remember to update these transformations over time for smooth animation. Set up the rendering pipeline by enabling depth testing for proper rendering of overlapping wings, configuring lighting parameters for realistic shading, and specifying material properties such as color, reflectance, and transparency. Implement a rendering loop that clears the screen and depth buffer, applies necessary transformations to position.

4.2 Data Flow Diagram

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data- flow diagram has no control flow, there are no decision rules and no loops. For each data flow, at least one of the endpoints (source and / or destination) must exist in a process. The refined representation of a process can be done in another data-flow diagram, which subdivides this process into sub-processes. The data-flow diagram is part of the structured-analysis modelling tools. When using UML, the activity diagram typically takes over the role of the data-flow diagram.

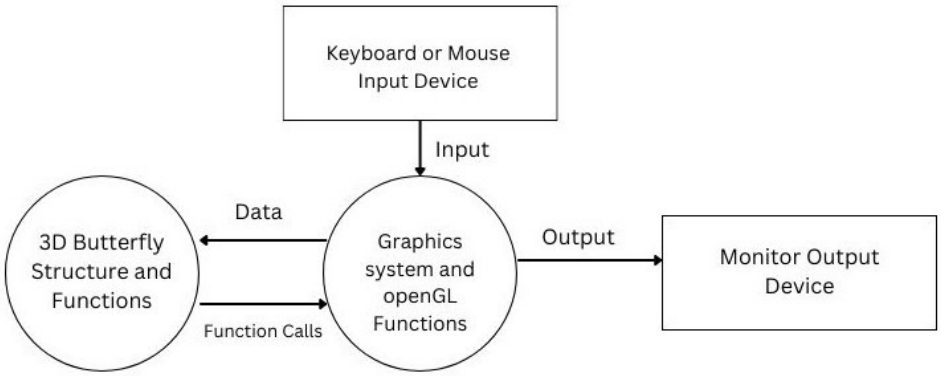

Figure 4.1 Level 0 Dataflow Diagram of the Proposed System

Figure 4.1 shows the Level 0 Dataflow diagram of the proposed application for an 3D butterfly. The keyboard and mouse devices are used for input to the application.

The graphics system processes these user keyboard/mouse interactions using built in OpenGL functions like glutKeyboardFunc(keys). 3D Butterfly is constructed in the memory using the user inputs sent to it by the Graphics System. The structure built in the memory is used by the graphics system to draw the butterfly onto the screen using OpenGL Functions.

4.3 Flowchart

A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process. Each step in the sequence is noted within a diagram shape. Steps are linked by connecting lines and directional arrows.

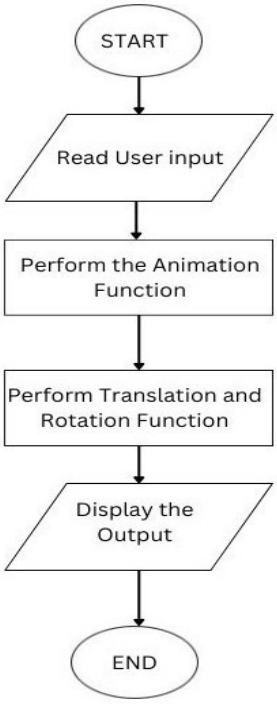

Figure 4.2 Flowchart of the Proposed System

Chapter 5

IMPLEMENTATION

5.1 Module Description

- **void draw_butterfly()**

  The code defines a function draw_butterfly() that draws a butterfly usingOpenGL

- **void spinbutterfly()**

  Spins the butterfly along different axes

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

  This function is used to accept user input for navigation an inserting nodesinto the tree.

- **void rightClickMenu (int ch)**

  This is menu function which allows the user to clear the screen, by deleting all the node or allow the user to close the application.

- **void wings_flap()**

  This function returns the flapping of wings of butterfly.

- **void movebutterfly()**

  Move the butterfly along the current direction.

5.2 High Level Code

5.2.1 Built-In Functions

- **void glClearColor(glEnum mode);**

  Clears the buffers namely color buffer and depth buffer. mode refers to GL_COLOR_BUFFER_BIT or DEPTH_BUFFER_BIT.

- **void glTranslate[fd](TYPE x, TYPE y, TYPE z);**

  Alters the current matrix by displacement of (x, y, z), TYPE is either GLfloator GLdouble.

- **void glutSwapBuffers();**

  Swaps the front and back buffers.

- **void glMatrixMode(GLenum mode);**

  Specifies which matrix will be affected by subsequent transformations, Mode can be GL_MODELVIEW or GL_PROJECTION.

- **void glLoadIdentity( );**

  Sets the current transformation matrix to identity matrix.

- **void glEnable(GLenum feature);**

  Enables an OpenGL feature. Feature can be GL_DEPTH_TEST (enables depth test for hidden surface removal).

- **void glPushMatrix() and void glPopMatrix();**

  Pushes to and pops from the matrix stack corresponding to the current matrix mode.

- **void glutBitmapCharacter(void *font, int character);**

  Without using display lists, glutBitmapCharacter renders character in named bitmap font. The fonts used are GLUT_BITMAP_TIMES_ROMAN_24.

- **void glRasterPos[234][ifd](GLfloat x, GLfloat y, GLfloat z);**

  This position, called the raster position, is used to position pixel and bitmapwrite operations.

- **void glutInit(int *argc, char **argv);**

  Initializes GLUT; the arguments from main are passed in and can be used bythe application.

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

  The value of mode is determined by the logical OR of options including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE).

- **void glutCreateWindow(char *title);**

  Creates a window on display; the string title can be used to label the window.

- **void glutMainLoop();**

  Causes the program to enter an event-processing loop.

- **void glutDisplayFunc(void (*func)(void))**
  Registers the display function func that is executed when the window needs tobe redrawn.
- **void glutMouseFunc(void *f(int button, int state, int x, int y)**
  Registers the mouse callback function f. The callback function returns thebutton (GLUT_LEFT_BUTTON,etc., the state of the button after the event(GLUT_DOWN), and the position of the mouse relative to the top-left cornerof the window.
- **void glutKeyboardFunc(void *f(char key, int width, int height))**
  Registers the keyboard callback function f. The callback function returns the ASCII code of the key pressed and the position of the mouse.
- **void glClearColor(GLclampf r, GLclampf g, GLclamp b, Glclamp a)**
  Sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.
- **void glViewport(int x ,int y, GLsizei width, GLsizei height)**
  Specifies the width*height viewport in pixels whose lower left corner is at(x,y) measured from the origin of the window.
- **void glColor3[b I f d ub us ui](TYPE r, TYPE g, TYPE b)**
  Sets the present RGB colors. Valid types are bytes(b), int(i), float(f), double(d), unsigned byte(ub), unsigned short(us), and unsigned int(ui).
- **void glutInitWindowSize(int width, int height);**
  Specifies the initial height and width of the window in pixels.
- **void glutReshapeFunc(void *f(int width, int height));**
  Registers the reshape callback function f. the callback function returns the height and width of the new window. The reshape callback invokes the displaycallback.
- **void createMenu(void);**
  This function is used to create menus which are used as options in program.

5.2.2 User Implementation

- **Reshape Function**

```
void reshape(int w, int h) //needs to some work to avoid distortion
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
   if (w >= h)
      glFrustum(-40.0, 40.0, -40.0 * (GLfloat)h / (GLfloat)w, 40.0 *
   (GLfloat)h / (GLfloat)w, 20.0,
  400.0);else
       glFrustum(-40.0, 40.0, -40.0 * (GLfloat)w / (GLfloat)h, 40.0 *
   (GLfloat)w / (GLfloat)h, 20.0,
  400.0);
  glMatrixMode(GL_MODELVIEW);
}
```

- **Display Function**

```
void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  if (frontpage == 0)
  {
    glLoadIdentity();
    gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 1.0, 0.0,
0.0);
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glRasterPos3f(30, 25, 0);
    for (j = 0; text1[j] != '\0'; j++)
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
text1[j]);
    glRasterPos3f(25, 30, 0);
    for (j = 0; text2[j] != '\0'; j++)
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
text2[j]);
    glRasterPos3f(20, 23, 0);
    for (j = 0; text3[j] != '\0'; j++)
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
text3[j]);
    glRasterPos3f(15, 20, 0);
    for (j = 0; text4[j] != '\0'; j++)
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
```

```
        text4[j]);
            glRasterPos3f(10, 60, 0);
            for (j = 0; text5[j] != '\0'; j++)
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
      text5[j]);
      glRasterPos3f(05, 40, 0);
      for (j = 0; text6[j] != '\0'; j++)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
        text6[j]);
      glRasterPos3f(0, 40, 0);
      for (j = 0; text7[j] != '\0'; j++)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
        text7[j]);
      glRasterPos3f(-5, -40, 0);
      for (j = 0; text8[j] != '\0'; j++)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
        text8[j]);
      glRasterPos3f(-10,  -40,  0);
      for (j = 0; text9[j] != '\0';
      j++)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
      text9[j]);glRasterPos3f(-15, -40, 0);
      for (j = 0; text10[j] != '\0'; j++)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
        text10[j]);
      glRasterPos3f(-5, 70, 0);
      for (j = 0; text11[j] != '\0'; j++)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
        text11[j]);
      glRasterPos3f(-10, 70, 0);
      for (j = 0; text12[j] != '\0'; j++)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
        text12[j]);
      glRasterPos3f(-15, 70, 0);
      for (j = 0; text13[j] != '\0'; j++)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
        text13[j]);
      glFlush();
    }
    else
    {
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 switch (backgroundcolor)
 {
 case 1:glClearColor(0.65625, 0.8, 0.10, 0.0);
   //bahina(green)break;
 case 2:glClearColor(0.84, 0.519, 0.23, 0.0); //Tree
   poppy(brown)break;
 case 3:glClearColor(0.86, 0.085, 0.085, 0.0);//harley davidson
   orange(red)break;
 case 4:glClearColor(0.109, 0.222, 0.73, 0.0);//persian
   bluebreak;
 case 5:glClearColor(1.0, 1.0, 1.0, 0.0);
   //whitebreak;
 case 6:glClearColor(0.0, 0.0, 0.0, 0.0);
   //blackbreak;}

GLUquadricObj* wings1, * wings2, * wings3, *
 wings4;glLoadIdentity();
 gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
 //translation functions
 glTranslatef(distance[0], distance[1], distance[2]);
 //rotate functions
 glRotatef(theta[0], 1.0, 0.0,
 0.0);
 glRotatef(theta[1], 0.0, 1.0, 0.0);
 glRotatef(theta[2], 0.0, 0.0, 1.0);
 //scale functions
 glScalef(zoom, zoom,
 zoom);
 //calling the middle body of the butterfly
 glCallList(displayListId);
 //wings part of the butterfly
 glPushMatrix(); //wings 1
 glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor1);
 glEnable(GL_TEXTURE_2D);
 glEnable(GL_COLOR_MATERIAL);
 glBindTexture(GL_TEXTURE_2D, textureId);
 glTranslatef(4.0, 0.0, 0.0);
 glRotatef(angle, -1.0, 0.0, 0.0);
```

```
glScalef(3.0, 4.0, 0.0);
wings1 =
gluNewQuadric();
gluQuadricTexture(wings1, GL_TRUE);
gluPartialDisk(wings1, 0, 5.0, 150, 150, 320,
80);glDisable(GL_TEXTURE_2D);
glDisable(GL_COLOR_MATERIAL);
glPopMatrix();
glPushMatrix(); //wings 2
 glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor1);
glEnable(GL_TEXTURE_2D);
glEnable(GL_COLOR_MATERIAL);
glBindTexture(GL_TEXTURE_2D, textureId);
glTranslatef(4.0, 0.0, 0.0);
glRotatef(angle, -1.0, 0.0, 0.0);
glScalef(2.5, 3.0, 0.0);
wings2 =
gluNewQuadric();
gluQuadricTexture(wings2, GL_TRUE);
gluPartialDisk(wings2, 0, 5.0, 150, 150, 280,
50);glDisable(GL_TEXTURE_2D);
glDisable(GL_COLOR_MATERIAL);
glPopMatrix();
glPushMatrix(); //wings 3
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor1);
glEnable(GL_TEXTURE_2D);
glEnable(GL_COLOR_MATERIAL);
glBindTexture(GL_TEXTURE_2D, textureId);
glTranslatef(4.0, 0.0, 0.0);
glRotatef(180 - angle, -1.0, 0.0, 0.0);
glScalef(2.5, 3.0, 0.0);
wings3 =
gluNewQuadric();
gluQuadricTexture(wings3, GL_TRUE);
gluPartialDisk(wings3, 0, 5.0, 150, 150, 280,
50);glDisable(GL_TEXTURE_2D);
glDisable(GL_COLOR_MATERIAL);
glPopMatrix();
glPushMatrix(); //wings 4
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
```

```
        materialColor1);
        glEnable(GL_TEXTURE_2D);
        glEnable(GL_COLOR_MATERIAL);
        glBindTexture(GL_TEXTURE_2D, textureId);
        glTranslatef(4.0, 0.0, 0.0);
        glRotatef(180 - angle, -1.0, 0.0, 0.0);
        glScalef(3.0, 4.0, 0.0);
        wings4 =
        gluNewQuadric();
        gluQuadricTexture(wings4, GL_TRUE);
        gluPartialDisk(wings4, 0, 5.0, 150, 150, 320,
        80);glDisable(GL_TEXTURE_2D);
        glDisable(GL_COLOR_MATERIAL);
        glPopMatrix();
    }
    glFlush();
    glutSwapBuffers();

}
```

- **Menumove Function**

```
void menumove(int key2)
{
    switch (key2) {
    case 0: translate = 1; rotate =
            0;break;
     case 1: translate = 0; rotate = 0;
            break;}}
```

- **Keyboard Function**

```
void keys(unsigned char key, int x, int y)// handles spinning along
differentaxis
{
  /* Use x, X, y, Y, z, and Z keys to move
  viewer */ if (rotate == 1) {
    if (key == 'x' | key == 'X') axis = 0;
    spinbutterfly(); if (key == 'y' | key == 'Y') axis
    = 1; spinbutterfly(); if (key == 'z' | key == 'Z')
    axis = 2; spinbutterfly();
  }
```

```
    if (translate == 1) {
      if (key == 'y' | key == 'Y') direction = 1; movebutterfly(); //
      Y-axis if (key == 'x' | key == 'X') direction = 0;
      movebutterfly(); // X-axisif (key == 'z' | key == 'Z') direction
      = 2; movebutterfly(); //Z-axis
    }
  if (key == 13) frontpage = 1;
 glutPostRedisplay();
```

- **Function to draw a Butterfly**

```
void draw_butterfly()
{
  //middle part of the butterfly
  glPushMatrix();
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor);
  glScalef(5.0, 1.4, 1.4);
  glutSolidSphere(1.0, 150, 80);
  glPopMatrix();
  //head part of the butterfly
  glPushMatrix();
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor);
  glTranslatef(5.0, 0.0, 0.0);
  glScalef(0.8, 1.0, 1.0);
  glutSolidSphere(1.0, 150, 80);
  glPopMatrix();
  //tail part of the butterfly
  glPushMatrix();
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor);
  glTranslatef(-4.0, 0.0, 0.0);
  glScalef(3.0, 1.4, 1.4);
  glutSolidSphere(1.0, 150, 80);
  glPopMatrix();
  //legs of the butterfly
  glPushMatrix();//legs1
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor);


 glTranslatef(1.0, 0.0, 1.4);
glRotatef(10.0, 0.0, 1.0, 0.0);
glScalef(1.4, 0.05, 0.05);
glutSolidCube(4.0); glPopMatrix();
```

```
    glPushMatrix();//legs2
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor);
    glTranslatef(1.0, 0.6, 1.4);
    glRotatef(10.0, 0.0, 1.0, 0.0);
    glScalef(1.4, 0.05, 0.05);
    glutSolidCube(4.0);
    glPopMatrix();
    glPushMatrix();//legs3
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor);
    glTranslatef(-2.5, 0.0, 1.0);
    glRotatef(10.0, 0.0, 1.0, 0.0);
    glScalef(1.6, 0.05, 0.05);
    glutSolidCube(4.0);
    glPopMatrix();
    glPushMatrix();//legs4
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor);
    glTranslatef(-2.5, 0.6, 1.0);
    glRotatef(10.0, 0.0, 1.0, 0.0);
    glScalef(1.6, 0.05, 0.05);
    glutSolidCube(4.0);
    glPopMatrix();
    //antenna part of the
    butterflyglPushMatrix();
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor);
    glTranslatef(8.0, 2.0, 0.5);
    glRotatef(60.0, 1.0, 0.0, 1.0);
    glScalef(4.3, 0.05, 0.05);
    glutSolidCube(2.0); //antenna 1
    glPopMatrix();
    glPushMatrix();
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
materialColor);
    glTranslatef(8.0, -2.0, -1.0);
    glRotatef(60.0, 1.0, 0.0, -1.0);
    glScalef(4.3, 0.05, 0.05);
    glutSolidCube(2.0); //antenna 2
    glPopMatrix();

}
```

- **Function to Spin Butterfly**

```
void spinbutterfly() //spins the butterfly along different axis
{
  theta[axis] += 1.0;
  if (theta[axis] > 360.0) theta[axis] -=
  360.0;display();
}
```

- **Wings_flap Function**

```
void wings_flap()
{
  if (flap == 1) {
    do {
      display();
      angle += speed;
    } while (angle < 75 && angle >
    0);speed *= -1;
  }
glutPostRedisplay();
}
```

- **Function to move Butterfly**

```
void movebutterfly()
{
  distance[direction] += 1.0;
  if (distance[direction] > 40.0) distance[direction] = -40.0;
  glutPostRedisplay();
}
```

- **myinit Function**

```
void myinit()
{
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_NORMALIZE);
  glShadeModel(GL_SMOOTH);
  glDisable(GL_COLOR_MATERIAL);
  displayListId = glGenLists(1);
```

```
glNewList(displayListId, GL_COMPILE); //Begin the display list
  draw_butterfly(); //Add commands for drawing butterfly to the
  display
list
  glEndList(); //End the display list
}
```

- **MainMenu Function**

```
void mainMenu(int key1)
{
  switch (key1)
  {
  case 1: exit(0);
    break;
  }
 }
```

- **MenuMove Function**

```
void menumove(int key2)
{
  switch (key2) {
  case 0: translate = 1; rotate =
    0;break;
  case 1: translate = 0; rotate =
    0;break;
  }
}
```

- **MenuSpin Function**

```
void menuspin(int key3)
{
  switch (key3) {
  case 0: translate = 0; rotate =
    1;break;
  case 1: translate = 0; rotate =
    0;break;
  }
   }
```

MenuScale Function

```
void menuscale(int key4)
{
  switch (key4) {
  case 0: zoom += 1.0;
    //glutPostRedisplay();
    break;


    case 1:zoom -= 1.0;
    //glutPostRedisplay();
    break;
  }
   }
```

- **Function to choose texture of Butterfly**

```
void chooseTex(int key5)
{
  switch (key5) {
  case 1: textureId = LoadTexture("tex1.bmp");
    //glutPostRedisplay();
    break;
  case 2: textureId = LoadTexture("tex2.bmp");
    //glutPostRedisplay();
    break;
  case 3: textureId = LoadTexture("tex3.bmp");
    //glutPostRedisplay();
    break;
  case 4: textureId = LoadTexture("tex4.bmp");
    //glutPostRedisplay();
    break;
  case 5: textureId = LoadTexture("tex5.bmp");
    //glutPostRedisplay();
    break;
  }
}
```

Function to choose Background color

```
void chooseBg(int key6)
{
  switch (key6) {
  case 1:backgroundcolor = 1;
    //bahina(green)break;
```

```
case 2:backgroundcolor = 2; //Tree
poppy(brown)break;
case 3:backgroundcolor = 3;//harley davidson
  orange(red)break;
case 4:backgroundcolor = 4;//persian
  bluebreak;
case 5:backgroundcolor = 5;
  //whitebreak;
case 6:backgroundcolor = 6;
  //blackbreak;
}
```

- **Function to Animate Butterfly**

```
void menuanimate(int key7)
{
  switch (key7) {
  case 1:flap = 1; wings_flap();
    break;
  case 2:flap = 0;
    break;
  }
}
```

Chapter 6

RESULTS

- **Start Page**

The starting page of the application, gives instructions to the user.
The Instructions are :

- Press right click to use pop-up menu and select the required option.
- Press x,y, and z to do changes along x,y and z axis respectively.

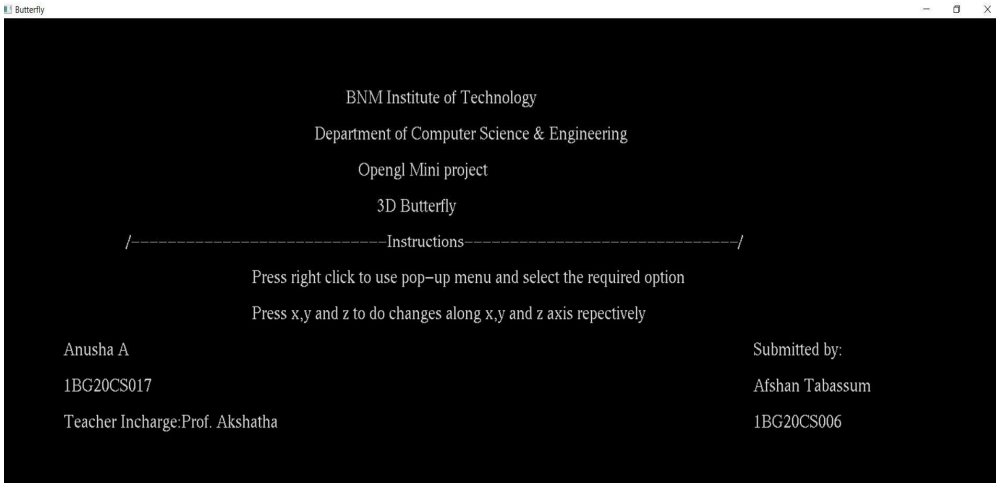

Figure 6.1 Start Page of the Application

- **Menu Page**

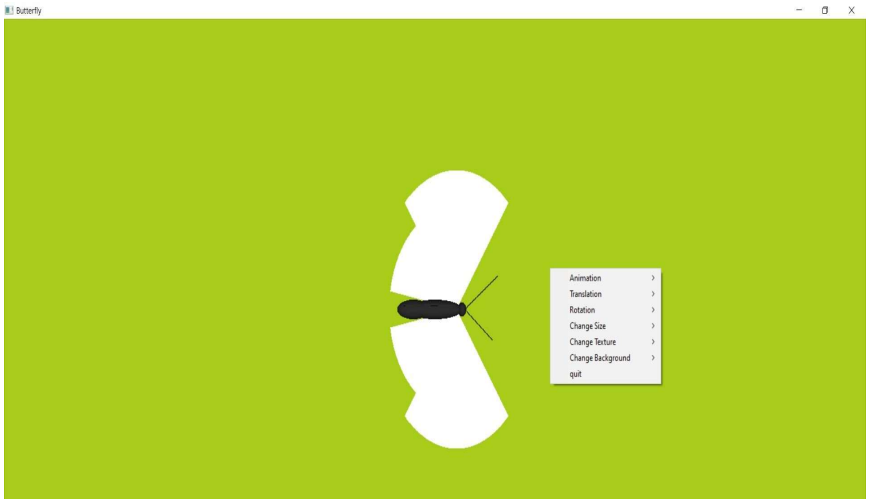

Figure 6.2 The Menu Page of the Application

The Menu page of the application, gives us information about what and all operationsor functions can be applied on the butterfly.

The Menu page contains operations like Animation, Translation, Rotation, Change size, Change texture, Change Background and quit.

- **Animation Opeartion**

In this page the user is allowed to give input i.e Animation on or off using Mouseinteraction. The flying Butterfly will appear on the screen .

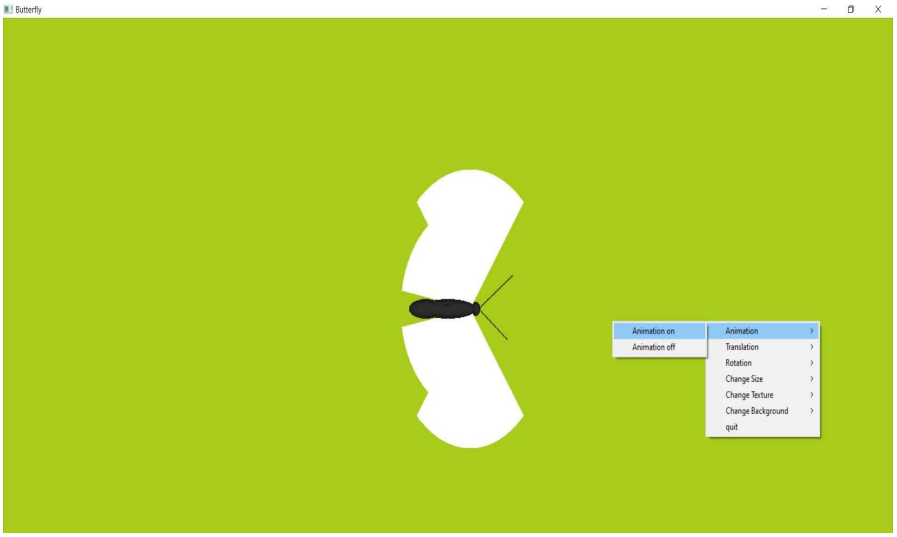

Figure 6.3 The Animation Opeartion

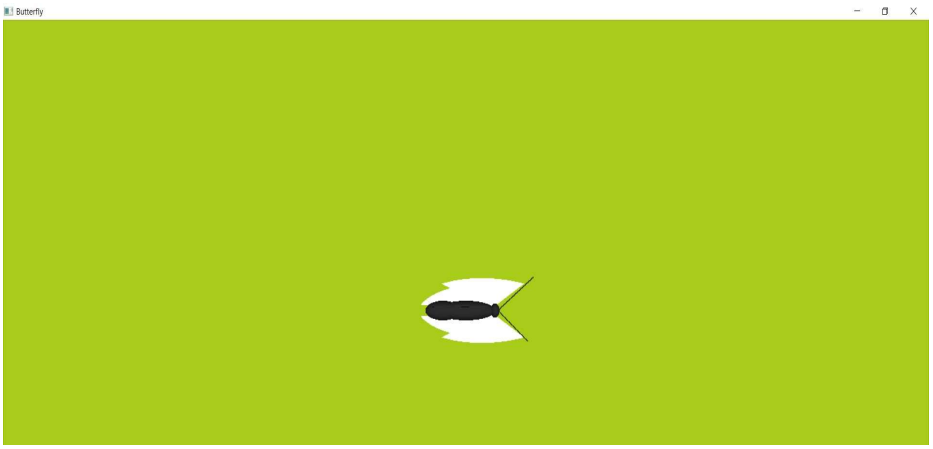

Figure 6.4 Flying Butterfly

To animate a 3D butterfly using OpenGL, you can create a range of lifelike movements and behaviors. One animation operation involves implementing wing flapping by defining rotation transformations around the wing joints. Using a sinusoidal function, you can smoothly oscillate the rotation angle over time, giving the impression of realistic wing flapping motion.

- **Translation and Rotation Opeartion**

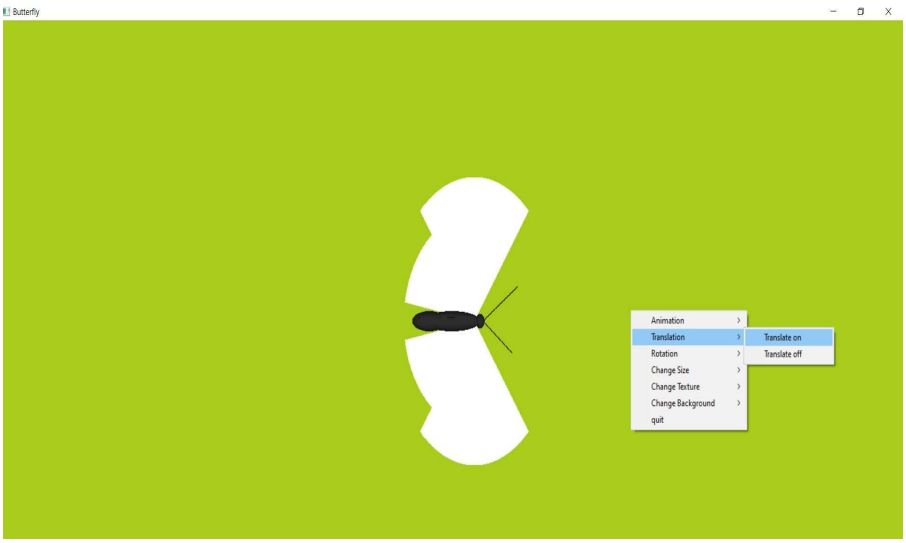

Figure 6.5 Translation operation

Translation refers to moving an object from one position to another in the 3D space. It involves shifting the coordinates of an object along the x, y, and z axes. Translation is useful for creating movement, positioning objects at specific locations, or animating objects along a path.

Figure 6.6 Translation operation along x axis

To translate the 3D butterfly in the x-axis using OpenGL, you can apply a translation transformation to the butterfly's geometry. By modifying the x-coordinate of each vertex, you can move the butterfly along the x-axis.

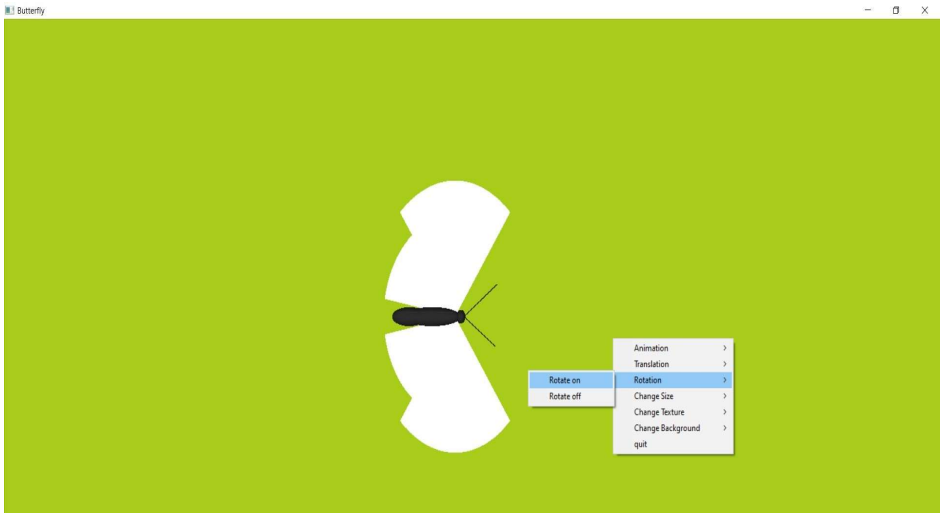

Figure 6.7 Rotation operation

Rotation involves rotating an object around a specific point or axis in the 3D space. Itchanges the orientation of an object relative to its initial position. Rotations can be applied along the x, y, and z axes and can be specified by angles or quaternions.

Rotation is commonly used for animating objects, creating spinning or orbiting motions, or aligning objects to a desired orientation.

Figure 6.8 Rotation operation along y axis

To rotate the 3D butterfly along the y-axis using OpenGL, you can apply a rotation transformation to the butterfly's geometry. By modifying the y-coordinate of each vertex, you can rotate the butterfly around the y-axis.

- **Change Size operation**

Figure 6.9 Change Size Operation

Figure 6.9 shows increase in size of the Butterfly. In OpenGL, the size of a 3D butterfly can be altered using the change size operation. The process involves modifying the scale factor applied to the butterfly's vertices or the transformation matrix that controls its overall size. By adjusting the scale factor, the butterfly can be enlarged or reduced in size.

➢ Change Background color Operation

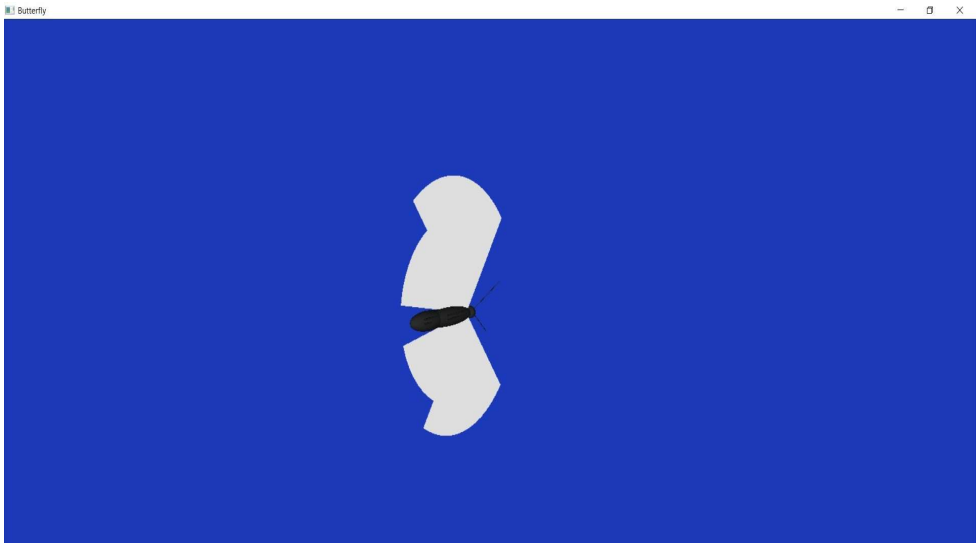

Figure 6.10 Change Background color Operation

Figure 6.10 shows that the color of background has been changed from green to blue. In OpenGL, the background color of a 3D butterfly can be changed using the change background color operation. This operation involves modifying the color value used to clear the frame buffer before rendering the butterfly. The frame buffer represents the image that is displayed on the screen. By adjusting the background color, the entire scene, including the butterfly, can be given a different visual ambiance**.**

➢ Quit Operation

The quit operation in OpenGL for a 3D butterfly involves terminating the rendering process and closing the application. When the quit operation is triggered, it signals the end of the program and releases any allocated resources.

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

The 3D Butterfly project using OpenGL offers a captivating and immersive experience that seamlessly combines the beauty of nature with the power of computer graphics. By leveraging the capabilities of OpenGL, the project showcases intricate details, vibrant colors, and smooth animations, bringing the delicate movements and intricate patterns of a butterfly to life. The interactive nature of the project allows users to control the butterfly's flight, adding an element of engagement and personalization. The incorporation of collision detection, particleeffects, and a visually appealing virtual environment enhances the realism and enchantment of the experience. Overall, the project serves as a testament to the fusion of technology and art, bridging the gap between the virtual and natural worlds and evoking a sense of wonder and appreciation for the wonders of nature. Through the 3D Butterfly project, OpenGL demonstrates its potential to create visually stunning and interactive simulations, leaving a lasting impression on users and inspiring exploration and creativity.

In the future this application can be further enhanced by adding more functionality to perform more operations on 3D Butterfly like Real-time Dynamic Lighting, Virtual Reality Integration, Interaction with the Environment and expanded Environment.

BIBLIOGRAPHY

[1] Edward Angel: Interactive Computer Graphics: A Top Down Approach 5th Edition, Addison – Wesley, 2008

[2] Donald Hearn and Pauline Baker: OpenGL, 3rd Edition, Pearson Education, 2004

[3] Wikipedia: Computer Graphics – https://en.wikipedia.org/wiki/ComputerGraphics

[4] Github link: https://github.com/nhegde610/3dbutterfly