Table of Contents

List of Figures

ABSTRACT

Origami (from ori meaning "folding", and gami meaning "paper") is the art of paper folding, which is often associated with Japanese culture. In modern usage, the word "origami" is used as an inclusive term for all folding practices, regardless of their culture of origin. The goal is to transform a flat square sheet of paper into a finished sculpture through folding and sculpting techniques. Modern origami practitioners generally discourage the use of cuts, glue, or markings on the paper. Origami folders often use the Japanese word kirigami to refer to designs which use cuts.

The small number of basic origami folds can be combined in a variety of ways to make intricate designs. The best-known origami model is the Japanese paper crane. In general, these designs begin with a square sheet of paper whose sides may be of different colours, prints, or patterns. Traditional Japanese origami, which has been practiced since the Edo period (1603–1867), has often been less strict about these conventions, sometimes cutting the paper or using non-square shapes to start with. The principles of origami are also used in stents, packaging, and other engineering applications.

Through this project, using the Paper Origami's paper folding techniques, the techniques behind designing a Paper Plane are mimicked so as to for the user to learn the basic techniques needed to achieve more complex origami work.

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, BNMIT, 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 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. Sahana D. Gowda**, Professor & Head of the Department of Computer Science and Engineering for the encouragement and motivation she provides.

I would also like to thank **Dr. Sreevidya R C,** 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.

Abhishek Anjan Pai
1BG17CS002

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 modelling, shader's, 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. He can give a series of commands, each one generating a graphical response from the computer. In this way he maintains a conversation, or dialogue, with the computer.

1.2 Problem Statement

The aim of this application is to show a basic implementation of Paper Origami used to make a Paper Plane that is implemented using the C programming language and the OpenGL API. The objective of the application is to demonstrate the paper folding technique required to form a paper plane using the principles of Computer Graphics. The application will also include user interaction through mouse events; for toggle of motion elements and navigation.

1.3 Motivation

Origami's history is rich and was a symbol of status since paper was very expensive back then. A gift of origami was a prized possession. Chinese and Japanese rulers used origami in official government papers, too. By teaching others the importance of its history, people can see its true worth and also develop the basic techniques need to master this form of art. The ability to develop this visualization using C programming and the OpenGL API serves as a motivation to develop this application.

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 modelling, shader's, 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 named integer 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 free glut. 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.

Primitive Assembly:

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half space, defined by a plane.

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 and then 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 frame buffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

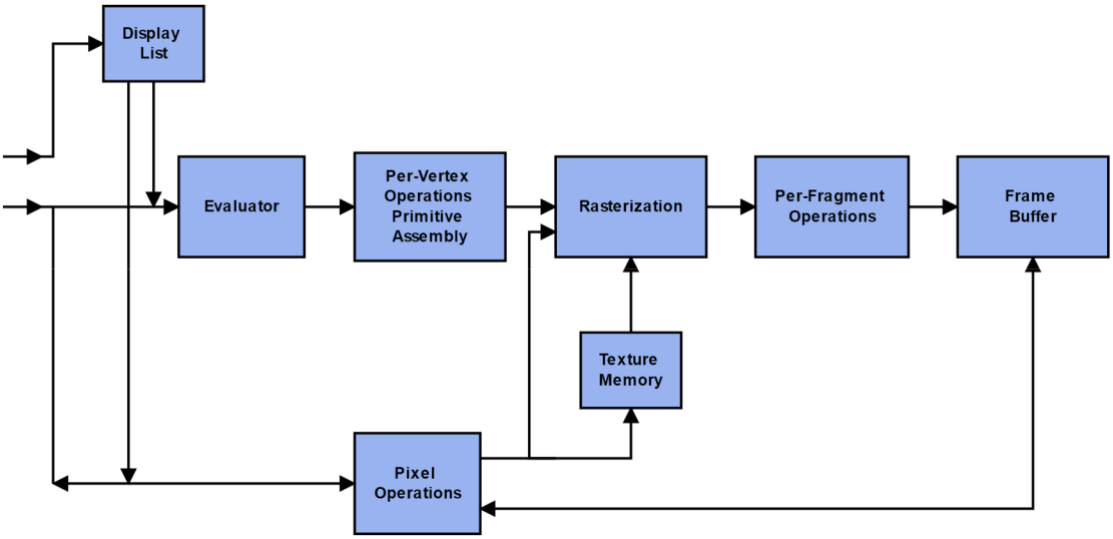

Fig 1.1 An illustration of the graphics pipeline process in OpenGL Architecture

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. In fields such as architecture and mechanical design, hand drafting is being replaced by computer-based drafting systems using plotters and workstations. Medical imaging uses computer graphics in a number of exciting ways.

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, such as 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 key frame-based animation. Photorealistic rendering of animated movie key frames 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.

Since its release OpenGL has been rapidly adopted throughout academia and industry. It's only among game developers that Microsoft's proprietary 3D API, Direct3D, which came soon after OpenGL bearing an odd similarity to it but optimized for Windows, is more popular.

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 televisions 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 of the 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.

More generally the term digital artist is used to describe an artist who makes use of digital technologies in the production of art. 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. A person using virtual reality equipment is able to look around the artificial world, move around in it, and interact with virtual features or items. 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 hap tic 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.

Specialized platforms such as arcade games, which present the game in a large, typically coin-operated chassis, were common in the 1980s in video arcades, but declined in popularity as other, more affordable platforms became available. 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 or software 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 4

SYSTEM DESIGN

4.1 Proposed System

The **Paper Origami** depicts a Three-Dimensional model of a paper plane using a simple paper and the related paper-folding techniques. A rectangular piece of paper is divided into eight sub sections that respectively form the eight base polygons. The polygon edges form the interior folding lines and the exterior paper boundary. This depiction shows the folding techniques to the User that are needed to form a paper plane.

The program starts off with a rectangular paper and its folding lines marked. The paper is folded into a paper plane at a set pace to ensure understanding and also, it is rotated so as to for the user to get a better all around look.

Once the paper is folded into a paper plane, it is unfolded back into the original rectangular paper to show both the do and undo techniques. The process repeats until the User decides to quit the program.

The user is provided with a menu upon right clicking the cursor on the simulation window. The menu consists of the options to REVERSE DIRECTION i.e. from folding state to unfolding or vice versa (the current state either folding or unfolding is specified on the simulation window name), TOGGLE MOTION i.e. to pause or un-pause completely any kind of motion, TOGGLE SPINNING i.e. to pause or un-pause the rotation keeping the folding motions going, and QUIT to exit the simulation window.

The proposed system for the application aims to simulate and demonstrate the self-constructing nature of the Paper Origami Paper Plane using C Programming and OpenGL API.

The OpenGL API provides us with built in functions to construct primitives which will be used to build the model of the Paper Plane. The Paper Plane coordinates will be constructed in the memory. The application will recursively iterate through the coordinates in the memory and draw the lines and the connecting edges on the screen.

4.2 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.

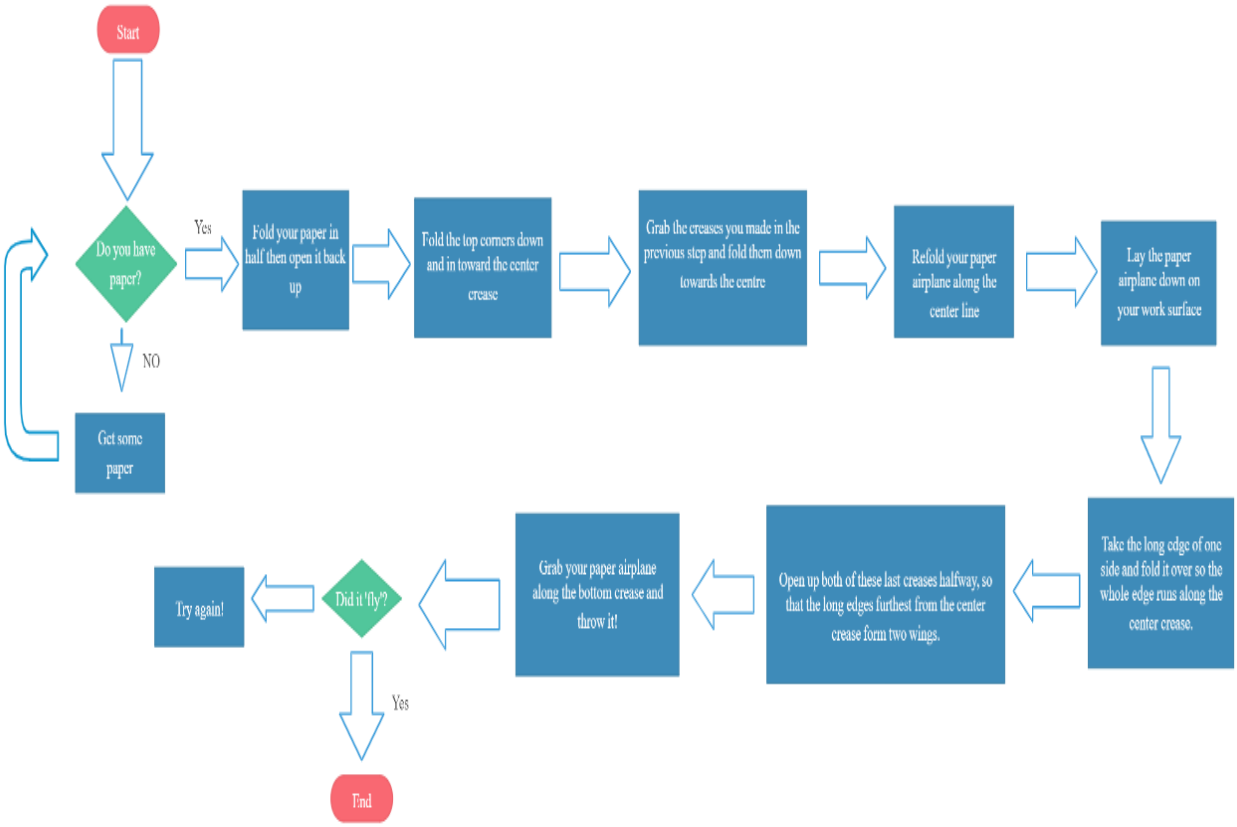

Figure 4.1 Flowchart of the Proposed System

Chapter 5

IMPLEMENTATION

5.1 Module Description

- **void polydlist()** : This method forms the initial paper along with marked set of folding lines using the base polygons.
- **void idle() :** The initial inception for each of the two folding and unfolding states. It also keeps track of the delay in between the states and state-process as well.
- **void draw_folded_plane() :** Houses the motion variables needed for the two states.
- **void menu() :** Provides User options and flexibility.
- **void display() :** To display the paper plane.

5.2 High Level Code

5.2.1 Built-In Functions

- **void glClear(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 GLfloat or 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), GL_LIGHTING (enables for lighting calculations), GL_LIGHTi (used to turn on the light for number i), etc.

- **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 any display lists, glutBitmapCharacter renders the character in the named bitmap font. The fonts used are GLUT_BITMAP_TIMES_ROMAN_24.

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

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

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

  Requests a display with the properties in the 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. The return value provides a reference to the window that can be used when there are multiple windows.

- **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 to be redrawn.

- **void glutMouseFunc(void *f(int button, int state, int x, int y)**

  Registers the mouse call-back function f. The callback function returns the button (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 corner of the window.

- **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 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). The maximum and minimum value for floating point types are 1.0 and 0.0 respectively, whereas the maximum and minimum values of the discrete types are those of the type, for eg, 255 and 0 for unsigned bytes.

- **void glutInitWindowSize(int width, int height);**

  Specifies the initial height and width of the window in pixels.

5.2.2 User Implementation

```
#include <stdlib.h>
#include <stdio.h>
#include <GL\glut.h>

int polygon_offset = 0;

enum {
   FLAT,            /* completely flat sheet of paper */
   FLAP1,           /* left flap being folded in */
   FLAP2,           /* right flap being folded in */
   CENTER2,         /* right side folded up at center */
```

```
    WING2,          /* right wing folded down */
    CENTER1,          /* left side folded up at center */
    WING1,          /* left wing folded down */
    FOLDED            /* fully folded paper airplane */
} States;

int motion = 1;
int spinning = 1;
int state = FLAT;
int click = 0;
int delay = 0;
int direction;
float flap1_angle = 0;
float flap2_angle = 0;
float center1_angle = 0;
float center2_angle = 0;
float wing1_angle = 0;
float wing2_angle = 0;

typedef GLfloat Point[2];

Point poly1[] =
{
  {-1, 0},
  {-1 / 3.0, 2 / 3.0},
  {-1, 2 / 3.0}
};

Point poly2[] =
{
  {-1, 1},
```

```
  {-1, 2 / 3.0},
  {-1 / 3.0, 2 / 3.0},
  {0, 1}
};

 Point poly3[] =
{
  {0, 1},
  {1, 1},
  {1, 2 / 3.0},
  {1 / 3.0, 2 / 3.0}
};

Point poly4[] =
{
  {1 / 3.0, 2 / 3.0},
  {1, 2 / 3.0},
  {1, 0}
};

Point poly5[] =
{
  {-1 / 3.0, 2 / 3.0},
  {0, 1},
  {0, -1.5},
  {-1 / 3.0, -1.5}
};

Point poly6[] =
{
  {0, 1},
```

```
 {1 / 3.0, 2 / 3.0},
  {1 / 3.0, -1.5},
  {0, -1.5}
};

Point poly7[] =
{
  {-1, 0},
  {-1 / 3.0, 2 / 3.0},
  {-1 / 3.0, -1.5},
  {-1, -1.5}
};

Point poly8[] =
{
  {1, 0},
  {1 / 3.0, 2 / 3.0},
  {1 / 3.0, -1.5},
  {1, -1.5}
};
void polydlist(int dlist, int num, Point points[])
{
    int i;
    glNewList(dlist, GL_COMPILE);
    glBegin(GL_POLYGON);
    for (i = 0; i < num; i++) {
        glVertex2fv(&points[i][0]);
    }
    glEnd();
    glEndList();
}
```

```
void idle(void)
{
   if (spinning)
      click++;
   switch (state) {
   case FLAT:
      delay++;
      if (delay >= 1500) {
         delay = 0;
         state = FLAP1;
         glutSetWindowTitle("origami (folding)");
         direction = 1;
      }
      break;
   case FLAP1:
      flap1_angle += 2 * direction;
      if (flap1_angle >= 180) {
         state = FLAP2;
      }
      else if (flap1_angle <= 0) {
         state = FLAT;
      }
      break;
   case FLAP2:
      flap2_angle += 2 * direction;
      if (flap2_angle >= 180) {
         state = CENTER2;
      }
      else if (flap2_angle <= 0) {
         state = FLAP1;
      }
```

```
    break;
    case CENTER2:
        center2_angle += 2 * direction;
if (center2_angle >= 84) {
            state = WING2;
        }
        else if (center2_angle <= 0) {
            state = FLAP2;
        }
        break;
    case WING2:
        wing2_angle += 2 * direction;
        if (wing2_angle >= 84) {
            state = CENTER1;
        }
        else if (wing2_angle <= 0) {
            state = CENTER2;
        }
        break;
    case CENTER1:
        center1_angle += 2 * direction;
        if (center1_angle >= 84) {
            state = WING1;
        }
        else if (center1_angle <= 0) {
            state = WING2;
        }
        break;
    case WING1:
        wing1_angle += 2 * direction;
        if (wing1_angle >= 84) {
```

```
        state = FOLDED;
        }
else if (wing1_angle <= 0) {
            state = CENTER1;
        }
        break;
    case FOLDED:
        delay++;
        if (delay >= 1500) {
            delay = 0;
            glutSetWindowTitle("origami (unfolding)");
            direction = -1;
            state = WING1;
        }
        break;
    }
    glutPostRedisplay();
}

void
draw_folded_plane(void)
{
    /* *INDENT-OFF* */
    glPushMatrix();
    glRotatef(click, 0, 0, 1);
    glRotatef(click / 5.0, 0, 1, 0);
    glTranslatef(0, .25, 0);
    glPushMatrix();
    glRotatef(center1_angle, 0, 1, 0);
    glPushMatrix();
    glTranslatef(-.5, .5, 0);
```

```
glRotatef(flap1_angle, 1, 1, 0);
glTranslatef(.5, -.5, 0);
glCallList(2);
glPopMatrix();
glCallList(5);

glPushMatrix();
glTranslatef(-1 / 3.0, 0, 0);
glRotatef(-wing1_angle, 0, 1, 0);
glTranslatef(1 / 3.0, 0, 0);

glCallList(7);
glPushMatrix();
glTranslatef(-.5, .5, 0);
glRotatef(flap1_angle, 1, 1, 0);
glTranslatef(.5, -.5, 0);
glCallList(1);
glPopMatrix();
glPopMatrix();
glPopMatrix();

glPushMatrix();
glRotatef(-center2_angle, 0, 1, 0);
glPushMatrix();
glTranslatef(.5, .5, 0);
glRotatef(-flap2_angle, -1, 1, 0);
glTranslatef(-.5, -.5, 0);
glCallList(3);
glPopMatrix();
glCallList(6);
glPushMatrix();
```

```
    glTranslatef(1 / 3.0, 0, 0);
    glRotatef(wing2_angle, 0, 1, 0);
    glTranslatef(-1 / 3.0, 0, 0);

    glCallList(8);
    glPushMatrix();
    glTranslatef(.5, .5, 0);

    glRotatef(-flap2_angle, -1, 1, 0);
    glTranslatef(-.5, -.5, 0);
    glCallList(4);
    glPopMatrix();
    glPopMatrix();
    glPopMatrix();
    glPopMatrix();
    /* *INDENT-ON* */
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glColor3ub(67, 205, 128);
    if (polygon_offset) {
#ifdef GL_VERSION_1_1
        glPolygonOffset(0.5, 0.0);
        glEnable(GL_POLYGON_OFFSET_FILL);
#else
# if GL_EXT_polygon_offset
        glPolygonOffsetEXT(0.5, 0.0);
        glEnable(GL_POLYGON_OFFSET_EXT);
```

```
# endif
#endif
}
   draw_folded_plane();
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   glColor3ub(255, 255, 255);
   if (polygon_offset) {
#ifdef GL_VERSION_1_1
      glPolygonOffset(0.5, 0);
      glDisable(GL_POLYGON_OFFSET_FILL);
#else
# if GL_EXT_polygon_offset
      glPolygonOffsetEXT(0.0, 0.0);
      glDisable(GL_POLYGON_OFFSET_EXT);
# endif
#endif
   }
   else {
      glPushMatrix();
      glTranslatef(0, 0, .05);
   }
   draw_folded_plane();
   if (!polygon_offset) {
      glPopMatrix();
   }
   glutSwapBuffers();
}

void visible(int state)
{
   if (state == GLUT_VISIBLE) {
```

```
    if (motion)
        glutIdleFunc(idle);
    }
    else {
      glutIdleFunc(NULL);
    }
}

void menu(int value)
{
    switch (value) {
    case 1:
      direction = -direction;
      if (direction > 0) {
        glutSetWindowTitle("origami (folding)");
      }
      else {
        glutSetWindowTitle("origami (unfolding)");
      }
      break;
    case 2:
      motion = 1 - motion;
      if (motion) {
        glutIdleFunc(idle);
      }
      else {
        glutIdleFunc(NULL);
      }
      break;
    case 3:
      spinning = 1 - spinning;
```

```
    break;
    case 666:
        exit(0);
    }
}

static int supportsOneDotOne(void)
{
    const char* version;
    int major, minor;
    version = (char*)glGetString(GL_VERSION);
    if (scanf(version, "%d.%d", &major, &minor) == 2) {
        return major > 1 || minor >= 1;
    }
return 0;
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
    glutCreateWindow("origami");
    glutDisplayFunc(display);
    glutVisibilityFunc(visible);
    glClearColor(.488, .617, .75, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluPerspective(40.0, 1.0, 0.1, 10.0);
    glMatrixMode(GL_MODELVIEW);
    gluLookAt(0, 0, 5.5,
        0, 0, 0,
        0, 1, 0);
```

```
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glLineWidth(2.0);
    polydlist(1, sizeof(poly1) / sizeof(Point), poly1);
    polydlist(2, sizeof(poly2) / sizeof(Point), poly2);
    polydlist(3, sizeof(poly3) / sizeof(Point), poly3);
    polydlist(4, sizeof(poly4) / sizeof(Point), poly4);
    polydlist(5, sizeof(poly5) / sizeof(Point), poly5);
    polydlist(6, sizeof(poly6) / sizeof(Point), poly6);
    polydlist(7, sizeof(poly7) / sizeof(Point), poly7);
    polydlist(8, sizeof(poly8) / sizeof(Point), poly8);
    glutCreateMenu(menu);
    glutAddMenuEntry("Reverse direction", 1);
    glutAddMenuEntry("Toggle motion", 2);
    glutAddMenuEntry("Toggle spinning", 3);
    glutAddMenuEntry("Quit", 666);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
#ifdef GL_VERSION_1_1
    polygon_offset = supportsOneDotOne();
#else
# if GL_EXT_polygon_offset
    polygon_offset = glutExtensionSupported("GL_EXT_polygon_offset");
# endif
#endif
    glutMainLoop();
    return 0;
}
```

Chapter 6

RESULTS

- **Start State (Folding State)**

  The starting state of the simulation mimics the folding techniques required to form a paper plane. The user follows these steps to form a paper plane of his / her own.

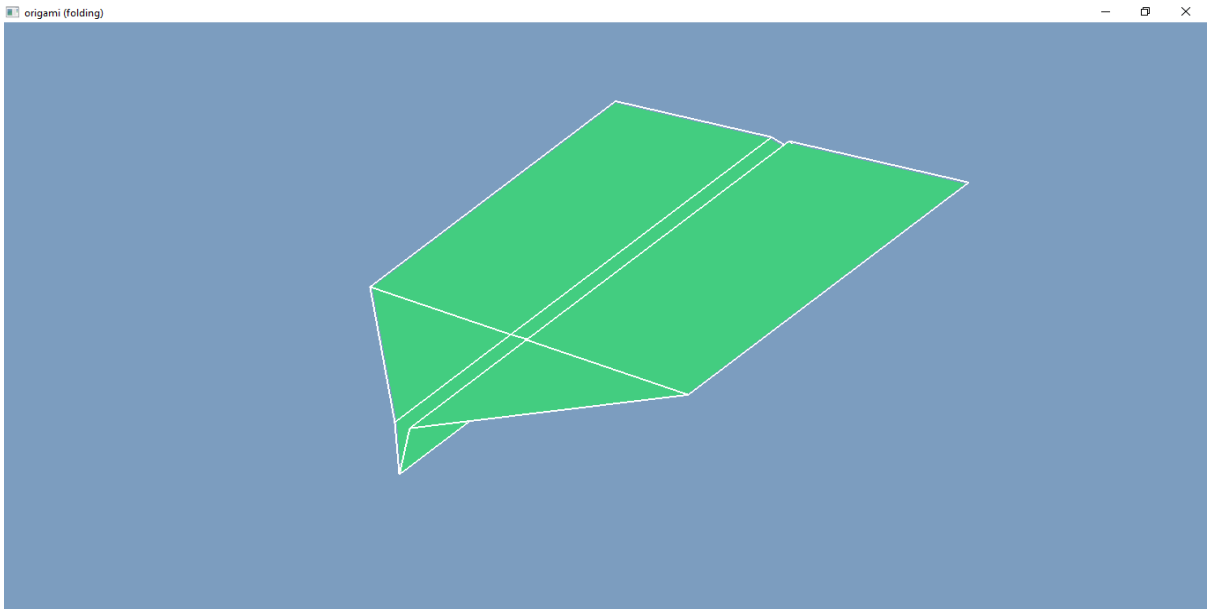

Figure 6.1 Start Page of the Simulation

- **Unfolding State**

  This state mimics the undo techniques for better user understanding.

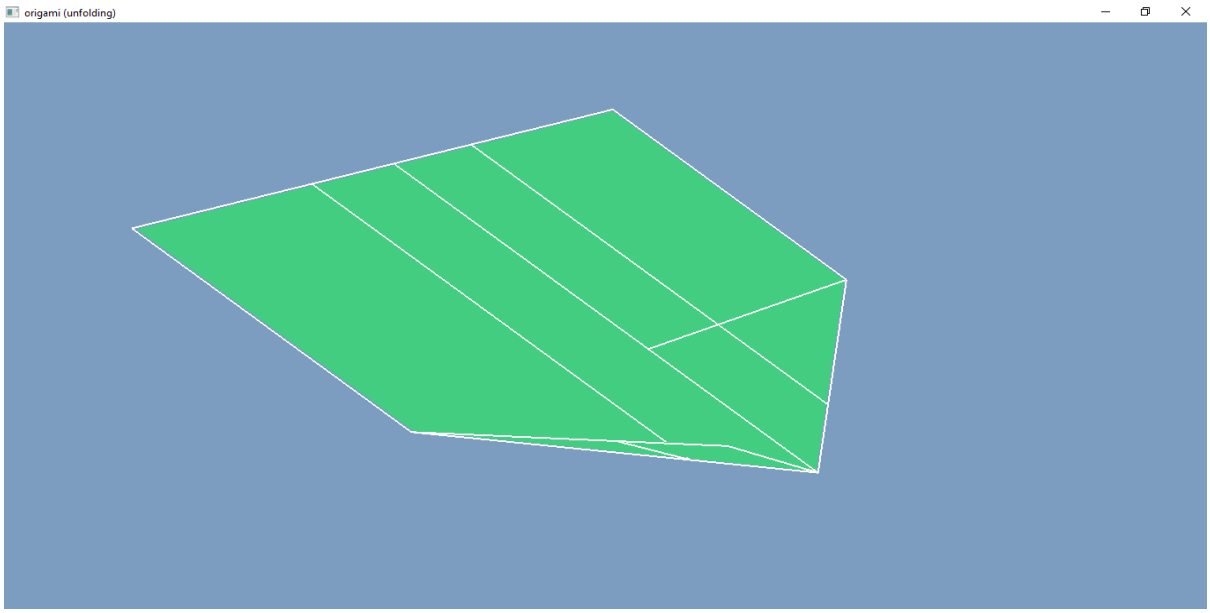

Figure 6.2 The Unfolding state in the Simulation

- **Drop Down Menu**

  The drop down menu is activated upon right mouse click over the window. It houses the options for reversing the state (Reverse direction), stopping motion completely (Toggle motion), stopping rotational motion (Toggle spinning), and to exit the simulation window (Quit).

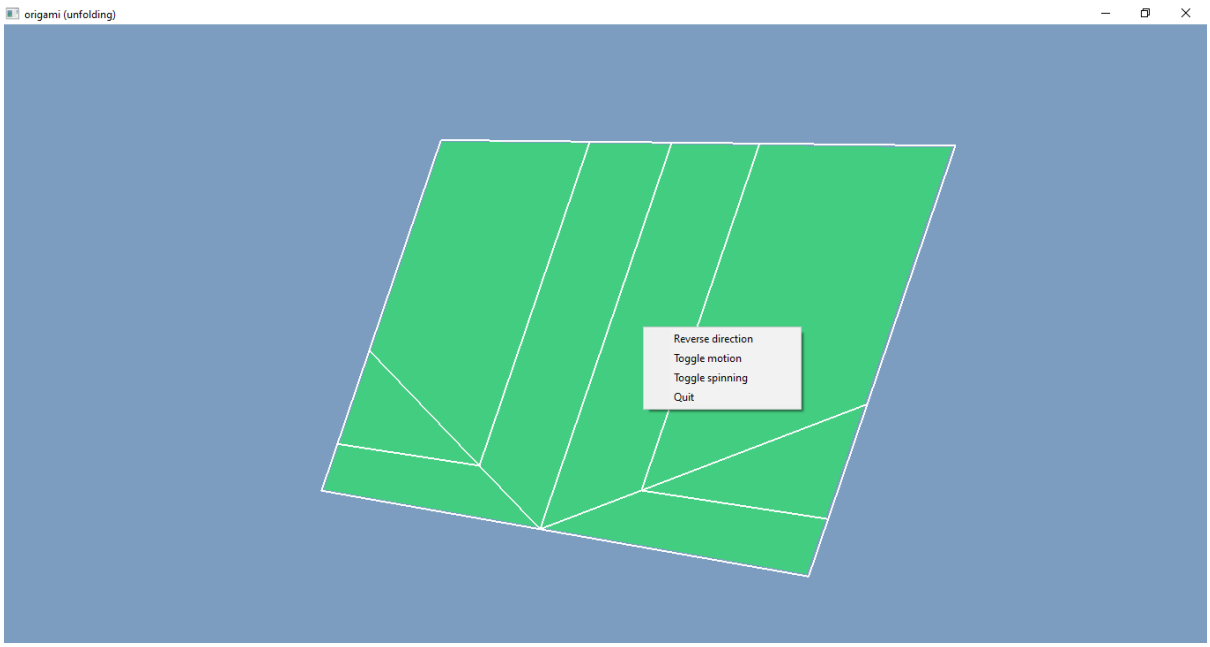

Figure 6.3 Right Mouse Click Drop Down Menu

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

Paper Origami is one of the most basic art forms that can be mastered with ease. The discipline teaches various techniques, demonstrates skills, and also teaches patience to master this craft. By practicing this demonstration, we are able to understand the basics required for any better crafted paper origami. During the course of building this application we utilised various OpenGL API Functions and variables that made it easier for us to visualize the applications working along with the working of the functions that were used.

In the future, this application can be further enhanced by adding more functionality to show more operations on that can be performed on a single paper, not just basic folding. The shadows can be animated to show how they move more realistically.

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: Paper Origami - https://en.wikipedia.org/wiki/Origami

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

[5] Columbia Ed: Paper Folding - http://www.cs.columbia.edu/cg/pdfs/115-origami-CATA-2006.pdf