VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Belgaum-590018

Project Report

On

"ORIGAMI 3D SIMULATION"

Submitted in partial fulfillment of the requirements for the VI Semester

Computer Graphics and Visualization Lab (10CSL67)

Bachelor of Engineering

IN

COMPUTER SCIENCE AND ENGINEERING

For the Academic year
2014-2015

BY

DEVANSH GULHANE	**1PE12CS045**
HIMANSHU BANSAL	**1PE12CS058**

PES Institute of Technology

Department of Computer Science and Engineering
PESIT BANGALORE SOUTH CAMPUS
HOSUR ROAD
BENGALURU-560100

PESIT BANGALORE SOUTH CAMPUS
HOSUR ROAD
BENGALURU-560100

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitle ***"ORIGAMI 3D SIMULATION"*** ***is*** *a bonafide work carried out by* ***DEVANSH GULHANE and HIMANSHU BANSAL*** *bearing USN* ***1PE12CS045 and 1PE12CS058*** respectively, *in Computer Graphics and Visualization Lab (10CSL67) for the 6th Semester in partial fulfillment for the award of Degree of Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belgaum during the year 2014-2015.*

---------- ----------

Signature of the guide ***Signature of the HOD***

Mrs. Sarasvathi V
Assistant Professor
PESIT BSC, Bengaluru.

Dr. Srikanta Murthy K
HOD, Dept. of CSE
PESIT BSC, Bengaluru.

ACKNOWLEDGEMENT

Firstly, we would like to thank our **Principal/Director Dr. J. Surya Prasad**, who gave us this wonderful opportunity and platform to experiment with our existing knowledge of the subject.

We are immensely grateful **Professor & Head of the Department** of Computer Science and Engineering, **Dr. Srikanta Murthy K**, for ensuring that we completed the project without any hassles and seamless.

We would also like to thank our project coordinator **Mrs. Sarasvathi V, Assistant Professor** for guiding us through the various stages of this project. Without their assistance and guidance, this project would not have been possible.

We would like to thank our college for providing this opportunity to create this interesting 3D simulation and providing us with the necessary infrastructure to successfully to create this project.

Support of friends and family was also of utmost importance for the completion of this project.

DEVANSH GULHANE (1PE12CS045)

HIMANSHU BANSAL (1PE12CS058)

ABSTRACT

The aim of this project is to develop a 3D simulation using graphics package which supports basic operations and includes creating objects using geometric entities. This project is implemented using OpenGL libraries underlying all basic concepts of computer graphics. The project shows the application of OpenGL in the paper boat simulation at different sets of stages. This is a 3D simulation of a paper boat which is very simple and to learn how to make it at simplest sets of steps.

This simulation includes the square paper at the start divided into 16 similar triangles. The viewer will have to press different sets of keyboard function in order to simulate the object into the desired result.

We can reverse the origami 3D simulation of the paper boat so to any steps missed by the user while making the paper boat and rotating the final object at 360 angle in open space.

TABLE OF CONTENTS

1. INTRODUCTION TO COMPUTER GRAPHICS

The concept of the computer graphics, OpenGL and GLUT are described below:

1.1 Computer graphics

Graphics created using computers and the representation of image data by a computer specifically with help from specialized graphic hardware and software is known as computer graphics. The interaction and understanding of computers and interpretation of data has been made easier because of computer graphics. A computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies and the video game industry.

Typically, the term “computer graphics” refers to several different things:

- The representation and manipulation of image data by a computer
- The various technologies used to create and manipulate images
- The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

Computer generated imagery can be categorized into several different types: two dimensional (2D), three dimensional (3D), and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with "the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time) component".

1.2 OpenGL

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer. The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although its possible for the API to be implemented entirely in software, it's designed to be implemented mostly or entirely in hardware.

OpenGL (**Open G**raphics **L**ibrary) is a cross-language, multi-platform API for rendering 2D and 3D computer graphics. The API is typically used to interact with a GPU, to achieve hardware-accelerated rendering. OpenGL was developed by Silicon Graphics Inc. (SGI) from 1991 and released in January and is widely used in

CAD, virtual reality, scientific visualization, information visualization, flight simulation, and video games.

1.3 GLUT

The **OpenGL Utility Toolkit** (**GLUT**) is a library of utilities for openGL programs, which primarily perform system-level I/O with the host OS. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres and the Utah teapot. GLUT also has some limited support for creating pop-up menus.
GLUT was written by Mark J. Kilgarard, author of OpenGL Programming for the X Window System and The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics, while he was working for Silicon Graphics Inc.
The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross-platform) and to make learning OpenGL easier. Getting started with OpenGL programming while using GLUT often takes only a few lines of code and does not require knowledge of operating system–specific windowing APIs.

All GLUT functions start with the glut prefix (for example, glutPostRedisplay marks the current window as needing to be redrawn).

The GLUT library supports the following functionality:

- Callback driven event processing.
- An `idle' routine and timers.
- Utility routines to generate various solid and wire frame objects.
- Support for bitmap and stroke fonts.
- Miscellaneous window management functions.
- Multiple windows for OpenGL rendering.

2. PROJECT DESCRIPTION

2.1 SOFTWARE AND HARDWARE REQUIREMENT

- Operating System : Ubuntu 14.04
- Code blocks with OpenGL libraries
- X86
- Mouse Driver
- Graphic Driver
- Main memory: 512 MB RAM
- Hard disk: 40 GB
- Hard disk speed in RPM: 5400 RPM
- Mouse : 2 or 3 button mouse
- Monitor: 1024*768 display resolution

2.2 USER REQUIREMENT

- Easy to understand and should be simple.
- The built-in functions should be utilized to maximum extent.
- OpenGL library facilities should be used.

3. API's DESCRIPTION

myinit():

This function initializes light source for ambient, diffuse and specular types.

glutInit(&argc,argv)

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

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

display():

This function creates and translates all the objects in a specified location in a particular order and also rotates the objects in different axes.

glClear(GL_COLOR_BUFFER_BIT);

glFlush();

glMatrixMode ():

Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The initial value is GL_MODELVIEW.Additionally, if the ARB_imaging extension is supported GL_COLOR is also accepted.

MainLoop():

This function whose execution will cause the program to begin an event processing loop.

PushMatrix():

Save the present values of attributes and matrices placing ,or pushing on the top of the stack.

PopMatrix():

We can recover them by removing them from stack.

Translated();

In translate function the variables are components of the displacement vector.

main():

The execution of the program starts from this function. It initializes the graphics system and includes many callback functions.

glLoadIdentity ():

glLoadIdentity replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix with the identity matrix

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

but in some cases it is more efficient.

glViewPort ():

Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0,0). width, height.Specify the width and height of the viewport. When a GL context is first attached to a window, width and height are set to the dimensions of that window.

gluPerspective ():

gluPerspective specifies a viewing frustum into the world coordinate system. In general, the aspect ratio in gluPerspective should match the aspect ratio of the associated viewport. For example, aspect = 2.0 means the viewer's angle of view is twice as wide in *x* as it is in *y*. If the viewport is twice as wide as it is tall, it displays the image without distortion. The matrix generated by gluPerspective is multiplied by the current matrix, just as if glMultMatrix were called with the generated matrix. To load the perspective matrix

onto the current matrix stack instead, precede the call to gluPerspective with a call to glLoadIdentity.

glBegin():

glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives. glBegin accepts a single argument that specifies in which of ten ways the vertices are interpreted. Taking n as an integer count starting at one, and N as the total number of vertices specified.

GL_POINTS

Treats each vertex as a single point. Vertex n defines point n. N points are drawn.

glRotate():

glRotate produces a rotation of angle degrees around the vector x y z . The current matrix (see glMatrixMode) is multiplied by a rotation matrix with the product replacing the current matrix, as if glMultMatrix were called .

glutSwapBuffers():

Performs a buffer swap on the *layer in use* for the *current window*. Specifically , glutSwapBuffers promotes the contents of the back buffer of the *layer in use* of the *current window* to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffers is called.

An implicit glFlush is done by glutSwapBuffers before it returns. Subsequent OpenGL commands can be issued immediately after calling glutSwapBuffers, but are not executed until the buffer exchange is completed.

If the layer in use is not double buffered, glutSwapBuffers has no effect.

glutPostRedisplay():

Mark the normal plane of *current window* as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

Logically, normal plane damage notification for a window is treated as a glutPostRedisplay on the damaged window. Unlike damage reported by the window system, glutPostRedisplay will *not* set to true the normal plane's damaged status (returned by glutLayerGet(GLUT_NORMAL_DAMAGED).

glFlush():

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

Because any GL program might be executed over a network, or on an accelerator that buffers commands, all programs should call glFlush whenever they count on having all of their previously issued commands completed. For example, call glFlush before waiting for user input that depends on the generated image.

glColor3f(GLfloat red,GLfloat blue,Glfloat green):

The GL stores both a current single-valued color index and a current four-valued RGBA color. glColor sets a new four-valued RGBA color. glColor has two major variants: glColor3 and glColor4. glColor3 variants specify new red, green, and blue values explicitly and set the current alpha value to 1.0 (full intensity) implicitly

MouseFunction:

glutMouseFunc sets the mouse callback for the *current window*. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The button parameter is one of GLUT_LEFT_BUTTON , GLUT_MIDDLE_ BUTTON, or GLUT_RIGHT_BUTTON. For systems with only two mouse buttons, it may not be possible to generate GLUT_MIDDLE_BUTTON callback. For systems with a single mouse button, it may be possible to generate only a GLUT_LEFT_BUTTON callback. The state parameter is either GLUT_UP or GLUT_DOWN indicating whether the callback was due to a release or press respectively. The x and y callback parameters indicate the window relative coordinates when the mouse button state changed. If a GLUT_DOWN callback for a specific button is triggered, the program can assume a GLUT_UP callback for the same button will be generated (assuming the window still has a mouse callback registered) when the mouse button is released even if the mouse has moved outside the window.

During a mouse callback, glutGetModifiers may be called to determine the state of modifier keys when the mouse event generating the callback occurred.

Passing NULL to glutMouseFunc disables the generation of mouse callbacks.

glutKeyboardFunc (void (*func) (unsigned char key, int x, int y)): Sets the current window's callback function for when a keyboard key is pressed. The x and y values passed to the callback function are the mouse cursor's pixel coordinates at the time of the key press.

glutMainLoop():

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

glutReshapeFunc (void (*func) (int width, int height)): Sets the resize callback function for the current window

4. PROJECT IMPLEMENTAION

The objective of the mini OpenGL project is to implement the concepts learnt in lab sessions as well as in Theory classes.

The aim of the project is to simulate a paper boat in open space using OpenGL functions.

This project is a basic implementation to simulate a paper boat in open space. For that I have used a square box divided into 16 similar triangles which is being simulated into a paper boat using keyboard keys and functions. It provides the user, ability to rotate the paper boat around at 360 angle in the open space by changing the viewing angle and position using the keyboard keys.

The program provides user to set the different stages of the paper boat simulation and to check the origami simulation of the object at each level and can view the final simulation by rotating it clockwise and anti-clockwise.

The program also provides a feature in which user can reverse the origami simulation using the keyboard functions and keys defined in the OpenGL libraries.

5. SOURCE CODE

```
//  main.cpp
//  vertex arrays
//  Created by devansh_himanshu on 19/05/15.
//  Copyright (c) 2015 pesit_bangalore_south_campus. All rights reserved.

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <GL/glut.h>

int displayFrontScreen = 1;
int idlestateangle =0;
int idlestate1 = 0, idlestate2 = 0, idlestate3 = 0;
int rotateonce =0;
int stage3_mode =0;
static int line_mid = 0, line_perp=0, line_perp1 = 0, turn1 = 0, turn2 =0;
float z= 0, x1 = 2.828, x2 = -2.828;
int j, depth = 0;
double lookatangle = 0;
int stage =1;
int state1 = 0, state2 = 0, state3= 0;
float valy = -2, valz = 0;
float transvalx = 0, transvaly = 2;
int xanglecount = 0;
int xangle =0;
int world_angles[] = {0,0,0};
int world_angles_true[] = {0,0,0};

float computeZ(float y){
   float temp = 4 - y*y;

   float z = sqrtf(temp);
   return z;
}

void triangle1_Stage3()
{
   glBegin(GL_TRIANGLES);
   glVertex3f(0, 0, 0);
   glVertex3f(-2, 0, 0);
   glVertex3f(0, 2, 0);
   glEnd();
```

```
}

void traingle2_Stage3()
{
   glBegin(GL_TRIANGLES);
   glVertex3f(0, 0, 0);
   glVertex3f(0, 2, 0);
   glVertex3f(2, 0, 0);

   glEnd();

}

void traingle3_Stage3()
{
   glBegin(GL_TRIANGLES);
   glVertex3f(2, 0, 0);
   glVertex3f(0, valy, valz);
   glVertex3f(0, 0, 0);
   glEnd();

}

void traingle4_Stage3()
{
   glBegin(GL_TRIANGLES);
   glVertex3f(0, valy, valz);
   glVertex3f(-2, 0, 0);
   glVertex3f(0, 0, 0);
   glEnd();
}

void coneTriangleOne_Stage3(){

   glBegin(GL_TRIANGLES);
   glVertex3f(-2, 0, 0);
   glVertex3f(-transvalx, transvaly, 0);
   glVertex3f(0, valy, valz);
   glEnd();
}

void coneTriangleTwo_Stage3()
{
   glBegin(GL_TRIANGLES);
   glVertex3f(2, 0, 0);
```

```
    glVertex3f(transvalx, transvaly, 0);
    glVertex3f(0, valy, valz);
    glEnd();
}

void conetraingaleThree_Stage3()
{
    glBegin(GL_TRIANGLES);
    glVertex3f(-2, 0, 0);
    glVertex3f(-transvalx, transvaly, 0);
    glVertex3f(0, valy, -valz);
    glEnd();
}

void conetraingleFour_Stage3()
{
    glBegin(GL_TRIANGLES);
    glVertex3f(2, 0, 0);
    glVertex3f(transvalx, transvaly, 0);
    glVertex3f(0, valy, -valz);
    glEnd();
}

void coneTriangle1()
{
    glBegin(GL_TRIANGLES);
    glVertex3f(0, -2, z);
    glVertex3f(x2, -2,0);
    glVertex3f(0, 0,0);
    glEnd();
}

void coneTriangle2()
{
    glBegin(GL_TRIANGLES);
    glVertex3f(x1, -2, 0);
    glVertex3f(0, -2, z);
    glVertex3f(0, 0, 0);
    glEnd();
}

void conetraingale3()
{
    glPushMatrix();
    glBegin(GL_TRIANGLES);
```

```
    glVertex3f(0, -2, -z);
    glVertex3f(x2, -2,0);
    glVertex3f(0, 0,0);
    glEnd();
    glPopMatrix();

}

void conetraingle4()
{
    glBegin(GL_TRIANGLES);
    glVertex3f(x1, -2, 0);
    glVertex3f(0, -2, -z);
    glVertex3f(0, 0, 0);
    glEnd();
}

void drawtraingleone()
{
    if (depth ==1) {
        glColor3ub(63, 81, 181);
        glPushMatrix();
        glBegin(GL_POLYGON);
        glVertex3f(0, 0,-2);
        glVertex3f(-2, 0,0);
        glVertex3f(-2, 2,0);
        glEnd();
        glPopMatrix();

    }else{
    glColor3ub(63, 81, 181);
    glPushMatrix();
    glBegin(GL_POLYGON);
    glVertex3f(0, 0,0);
    glVertex3f(-2, 0,0);
    glVertex3f(-2, 2,0);
    glEnd();
    glPopMatrix();
    }
}

void drawtriangletwo()
{
    if (depth ==1) {
        glColor3ub(86, 119, 252);
```

```
        glPushMatrix();
        glBegin(GL_POLYGON);
        glVertex3f(0, 0,2);
        glVertex3f(-2, 2,0);
        glVertex3f(0, 2,0);
        glEnd();
        glPopMatrix();
    }else{
    glColor3ub(86, 119, 252);
    glPushMatrix();
    glBegin(GL_POLYGON);
    glVertex3f(0, 0,0);
    glVertex3f(-2, 2,0);
    glVertex3f(0, 2,0);
    glEnd();
    glPopMatrix();
    }
}

void drawtrianglethree()
{
    if (depth ==1) {
        glColor3ub(0, 188, 212);
        glPushMatrix();
        glBegin(GL_TRIANGLES);
        glVertex3f(0, 0,-2);
        glVertex3f(0, 2,0);
        glVertex3f(2, 2,0);
        glEnd();
        glPopMatrix();
    }
    glColor3ub(0, 188, 212);
    glPushMatrix();
    glBegin(GL_TRIANGLES);
    glVertex3f(0, 0,0);
    glVertex3f(0, 2,0);
    glVertex3f(2, 2,0);
    glEnd();
    glPopMatrix();
}

void drawtrianglefour()
{
    if (depth ==1) {
        glColor3ub(3, 169, 15);
```

```
        glPushMatrix();
        glBegin(GL_TRIANGLES);
        glVertex3f(0, 0,-2);
        glVertex3f(2, 2,0);
        glVertex3f(2, 0,0);
        glEnd();
        glPopMatrix();
    }
    glColor3ub(3, 169, 15);
    glPushMatrix();
    glBegin(GL_TRIANGLES);
    glVertex3f(0, 0,0);
    glVertex3f(2, 2,0);
    glVertex3f(2, 0,0);
    glEnd();
    glPopMatrix();
}

void drawsquareone(int flag)
{
    if (flag ==2) {
        glPushMatrix();
        drawtraingleone();
        glPushMatrix();
        glRotatef(turn1, -1, 1, 0);
        drawtriangletwo();
        glPopMatrix();
        glPopMatrix();
    }
    else{
    glPushMatrix();
    drawtraingleone();
    drawtriangletwo();
    glPopMatrix();
    }
}

void drawsquaretwo(int flag)
{
    if (flag ==1) {
        glPushMatrix();
        glPushMatrix();
        glRotatef(turn1, 1, 1, 0);
        drawtrianglethree();
        glPopMatrix();
```

```
        drawtrianglefour();
        glPopMatrix();

    }
    else{
        glPushMatrix();
        drawtrianglethree();
        drawtrianglefour();
        glPopMatrix();
    }

}

void drawrectangletop(int flag)
{
    if (flag ==1) {
        glPushMatrix();
        glPushMatrix();
        glRotatef(turn1, 1, 1, 0);
        drawsquareone(0);
        glPopMatrix();
        drawsquaretwo(1);
        glPopMatrix();
    }
    else if(flag == 2){
        glPushMatrix();
        drawsquareone(2);
        glPushMatrix();
        glRotatef(turn1, -1, 1, 0);
        drawsquaretwo(0);
        glPopMatrix();
        glPopMatrix();

    }
    else{
    glPushMatrix();
    drawsquareone(0);
        glPushMatrix();
    drawsquaretwo(0);
        glPopMatrix();
    glPopMatrix();
    }
}

void drawrectanglebottom(int flag)
```

```
{
   if (flag ==2) {
      glPushMatrix();
      glTranslated(-2, 0,0);
      drawsquaretwo(0);
      glTranslated(4, 0, 0);
      drawsquareone(2);
      glPopMatrix();
   }
   else if (flag ==1) {
      glPushMatrix();
      glTranslated(-2, 0,0);
      drawsquaretwo(1);
      glTranslated(4, 0, 0);
      drawsquareone(0);
      glPopMatrix();
   }
   else{
      glPushMatrix();
      glTranslated(-2, 0,0);
      drawsquaretwo(0);
      glTranslated(4, 0, 0);
      drawsquareone(0);
      glPopMatrix();
   }
}

int i;

void displaytext(int x, int y,char const* s)
{
   int len;
   glRasterPos2d(x,y);
   len=strlen(s);
   for(i=0;i<len;i++)
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,s[i]);
   glFlush();

}

void secondInit()
{
   glClearColor(0, 1, 0, 0);
   glShadeModel(GL_SMOOTH);
   glMatrixMode(GL_PROJECTION);
```

```
    glLoadIdentity();
    gluPerspective(65.0, 1, 1.0, 20.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0, 0, -15);
}

void init()
{
    glClearColor(0, 1, 0, 0);
    glShadeModel(GL_SMOOTH);
    gluOrtho2D(0,100,0,100);
}

void message()
{
    glColor3f(0.0,0.0,0.0);
    displaytext(38,65,"Simulation of a Paper Boat");
}

void displayFirstScreen()
{
    glPushMatrix();
    glColor3f(1.0,1.0,1.0);
    displaytext(40,90,"COMPUTER GRAPHICS PROJECT");
    displaytext(40,80,"ORIGAMI 3D SIMULATION");
    message();
    glColor3f(1.0,1.0,1.0);
    displaytext(4,35,"DEVANSH GULHANE");
    displaytext(4,32,"1PE12CS045");
    displaytext(4,25,"HIMANSHU BANSAL");
    displaytext(4,22,"1PE12CS058");
    glColor3f(1.0,1.0,1.0);
    displaytext(4,15,"Press 'z' to move towards the Simulation");
    glutSwapBuffers();
    glPopMatrix();
}

void displayStage1(){
    glPushMatrix();//origin at 0,0 at this point;first push; make changes and pop returns
here
    if (turn2 ==1) {
        glRotatef(45, 0, 0, 1);
    }
    glPushMatrix();//push for rotate about xaxis;
```

```
    glRotatef(line_perp, 1, 0, 0);
    //top-left quadrant
    glPushMatrix();//push to draw the topleft;context = rotateabout x
    glTranslated(-2, 2, 0);
    drawrectangletop(state3);
    glTranslated(0, -2, 0);
    drawrectanglebottom(state3);
    //end
    glPopMatrix();//translation back to rotation about x
    glPopMatrix();//backto origin;
    //bottom-left quadrant
    glPushMatrix();//push to bottom -left
    glTranslated(-2, -2, 0);
    glRotatef(180, 0, 0, 1);
    glPushMatrix();
    drawrectangletop(2);
    glTranslated(0, -2, 0);
    drawrectanglebottom(2);
    //end
    glPopMatrix();//back to origin
    glPopMatrix();
    glPushMatrix();//push to change state for rotation ;all the stuff inside this context will
by rotated by y =0.
    glRotatef(line_mid, 0, 1, 0);//rotate the entire right side
    glPushMatrix();//push to add rotate about x
    glRotatef(line_perp, 1, 0, 0);//rotation about x
    glPushMatrix();//current context, rotate; push to topright
    //top-right quadrant
    glTranslated(2, 2, 0);
    drawrectangletop(2);
    glTranslated(0, -2, 0);
    drawrectanglebottom(2);
    //end
    glPopMatrix();//back to rotation about x
    glPopMatrix();//back to roaton right
    glPushMatrix();
    glRotatef(180, 0, 0, 1);
    glTranslated(-4, 4, 0);
    //bottom-right quadrant
    glPushMatrix();//rotate, add translaton for bottom quad
    glTranslated(2, -2, 0);
    drawrectangletop(1);
    glTranslated(0, -2, 0);
    drawrectanglebottom(1);
    //end
```

```
    glPopMatrix();//return to rotate
    glPopMatrix();

    glPopMatrix();//rotate context over;context back t origin
    glPopMatrix();//extra context buffer
}

void displayStage2()
{
    glPushMatrix();
    glColor3ub(86, 119, 252);
    coneTriangle1();
    coneTriangle2();
    coneTriangle1();
    coneTriangle2();
    conetraingale3();
    conetraingle4();
    conetraingale3();
    conetraingle4();
    glPopMatrix();
}

void displayStage3()
{
    if (stage3_mode == 1) {
    }
    if (!rotateonce) {
    }

    glPushMatrix();
    triangle1_Stage3();
    traingle2_Stage3();
    glPushMatrix();
    glRotated(180, 0, 1, 0);
    glColor3ub(86, 119, 252);
    traingle3_Stage3();
    traingle4_Stage3();
    glPopMatrix();
    glColor3f(1, 1, 1);
    traingle3_Stage3();
    traingle4_Stage3();
    glColor3ub(86, 119, 252);
    coneTriangleOne_Stage3();
    coneTriangleTwo_Stage3();
    conetraingaleThree_Stage3();
```

```
    conetraingleFour_Stage3();
    glPopMatrix();
}

void rotateWorld(int mode)
{
    if (mode ==1) {
        glRotatef(world_angles[0], world_angles_true[0], 0, 0);
        glRotatef(world_angles[1], 0, world_angles_true[1], 0);
        glRotatef(world_angles[2], 0, 0, world_angles_true[2]);
    }

}

void idle(void){
    if (idlestate1) {

    }else if (idlestate2){

    }else if (idlestate3){

    }
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glPolygonMode(GL_FRONT, GL_LINE);
    glColor3f(1.0, 1.0, 1.0);
    //if condition for the first screen
    if (displayFrontScreen) {
        // do something
        displayFirstScreen();
    }
    else
    {
        if(stage3_mode == 1)
        {
            rotateWorld(stage3_mode);
        }
        if (stage == 1) {
            displayStage1();
        }else if (stage ==2){
            displayStage2();
        }else if (stage == 3){
```

```
            displayStage3();
        }
        glColor3f(0, 0, 0);
        glBegin(GL_POINTS);
        glVertex2f(0, 0);
        glEnd();
        glutSwapBuffers();
    }
}

void reshape(int w, int h)
{
    glViewport(0, 0, GLsizei(w), GLsizei(h));
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(65.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0, 0, -15);
}

void keyboard(unsigned char key, int x, int y)
{
    if (stage == 1) {
        stage3_mode = 0;
        //to-do: all the stage one keyboard conditions go here
        switch (key) {
            case 'd':
                line_mid = (line_mid + 3) % 180;
                if (line_mid >170) {
                    state1 = 1;
                }
                if (state1 &&state2) {
                    state3 = 1;
                }
                glutPostRedisplay();
                break;

            case 'D':line_mid = (line_mid - 10) % 180;
                glutPostRedisplay();
                break;
            case 'p':
                line_perp = (line_perp + 3) % 180;
                line_perp1 = (line_perp1 - 3) % 180;
                if (line_perp >170) {
```

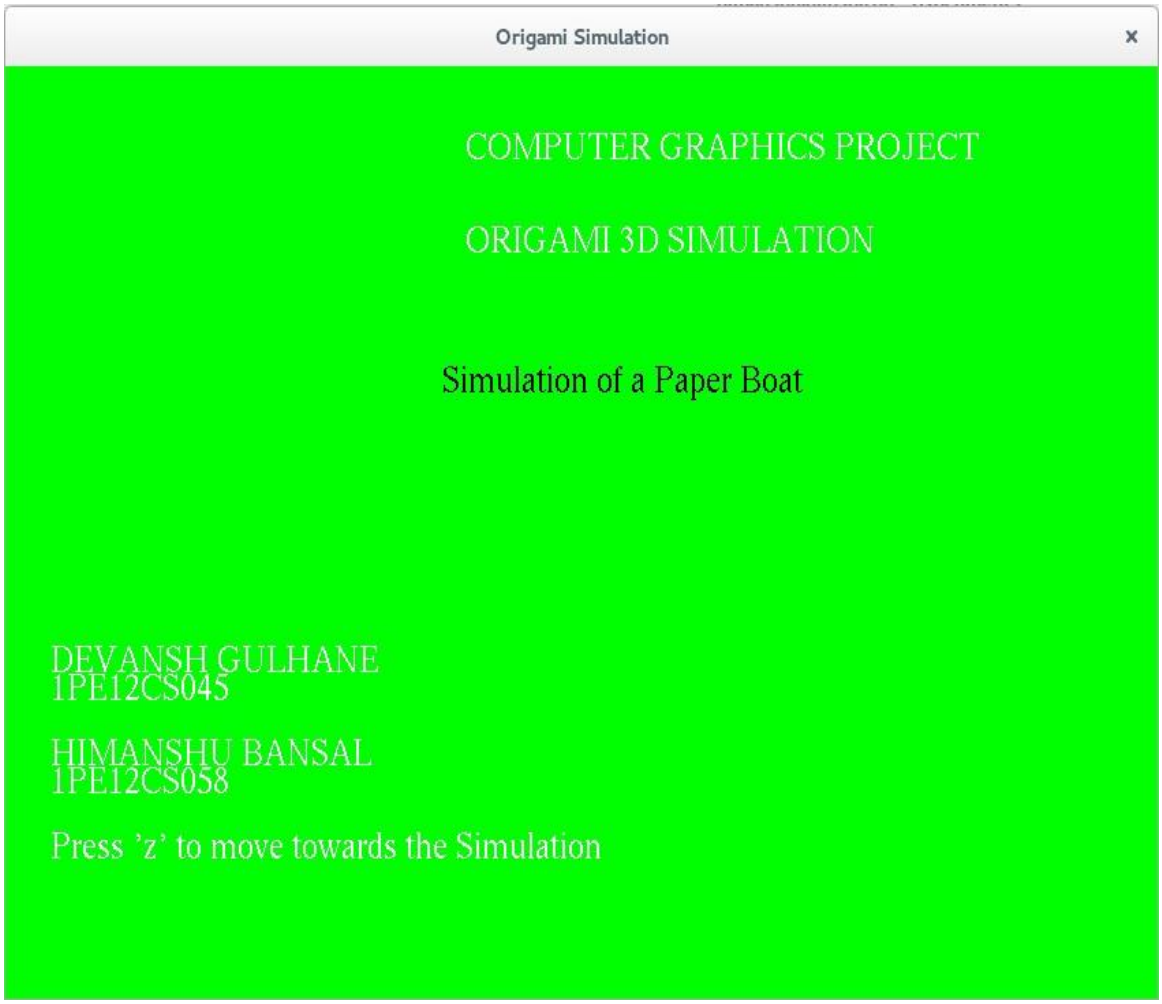
```
            state2 = 1;
         }
         if (state1 &&state2) {
            state3 = 1;
         }
         glutPostRedisplay();
         break;

      case 'P':line_perp = (line_perp - 10) % 180;
         line_perp1 = (line_perp1 + 3) % 180;
         glutPostRedisplay();
         break;
      default:
         break;
   }
   if (state3 > 0)
   {
      switch (key)
      {
         case 'y': turn1 = (turn1 + 5) % 180;
            if (turn1>170) {
               //change from one stage to another
               stage =2;
               idlestate2 = 1;

            }
            glutPostRedisplay();
            break;

         case 'Y': turn1 = (turn1 -5)%360;
            glutPostRedisplay();
            break;
         default:
            break;
      }
   }
}
else if (stage == 2){
   stage3_mode = 0;
   if (x1>0) {

      switch (key) {
         case 'd': z = z + 0.2;
            x1 = x1 - 0.2;
            x2 = x2 + 0.2;
```

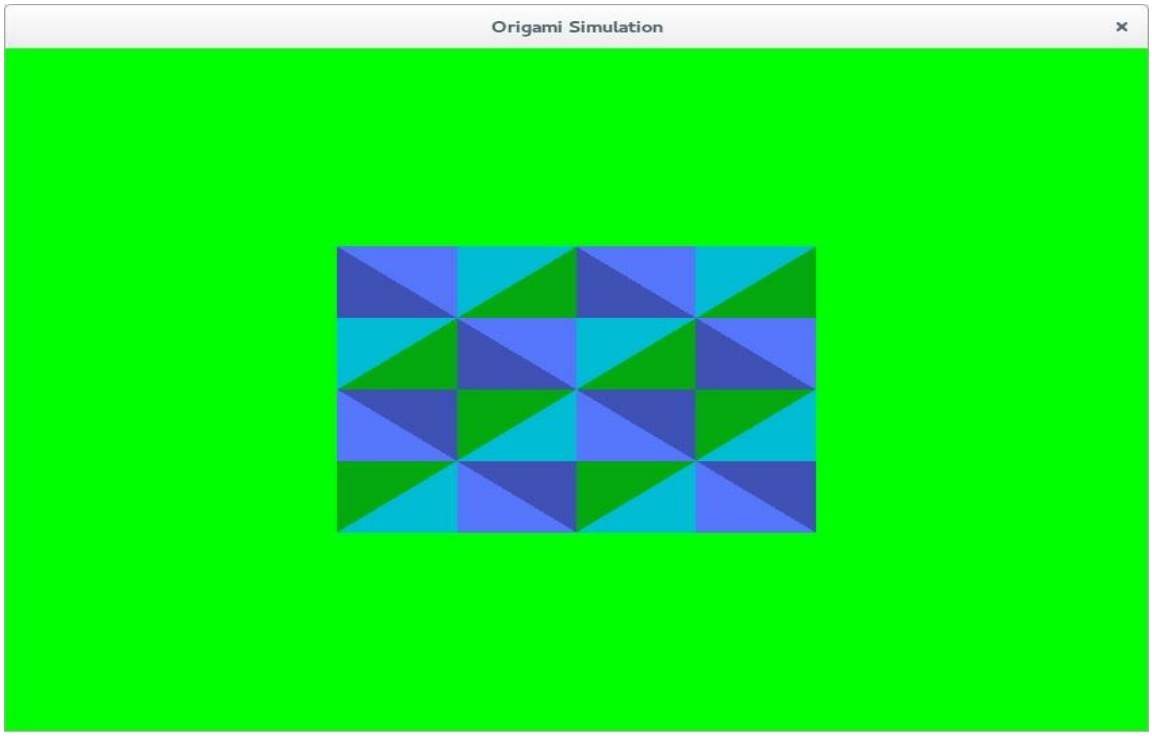
```
            glutPostRedisplay();
            break;

        case 'D':line_mid = (line_mid - 10) % 180;
            glutPostRedisplay();
            break;
        case 'p':
            line_perp = (line_perp + 3) % 180;
            line_perp1 = (line_perp1 - 3) % 180;
            if (line_perp >170) {
                state2 = 1;
            }
            if (state1 &&state2) {
                state3 = 1;
            }
            glutPostRedisplay();
            break;

        case 'P':line_perp = (line_perp - 10) % 180;
            line_perp1 = (line_perp1 + 3) % 180;
            glutPostRedisplay();
            break;
        default:
            break;
    }

}
else if (x1 <= 0){

    if (key == 'p') {
        if(lookatangle<5){
            lookatangle = lookatangle + .4;
            stage =3;
            glutPostRedisplay();

        }
    }
}
if (state3 > 0)
{
    switch (key)
    {
        case 'y': turn1 = (turn1 + 5) % 180;
            if (turn1>170) {
                turn2 = 1;
```

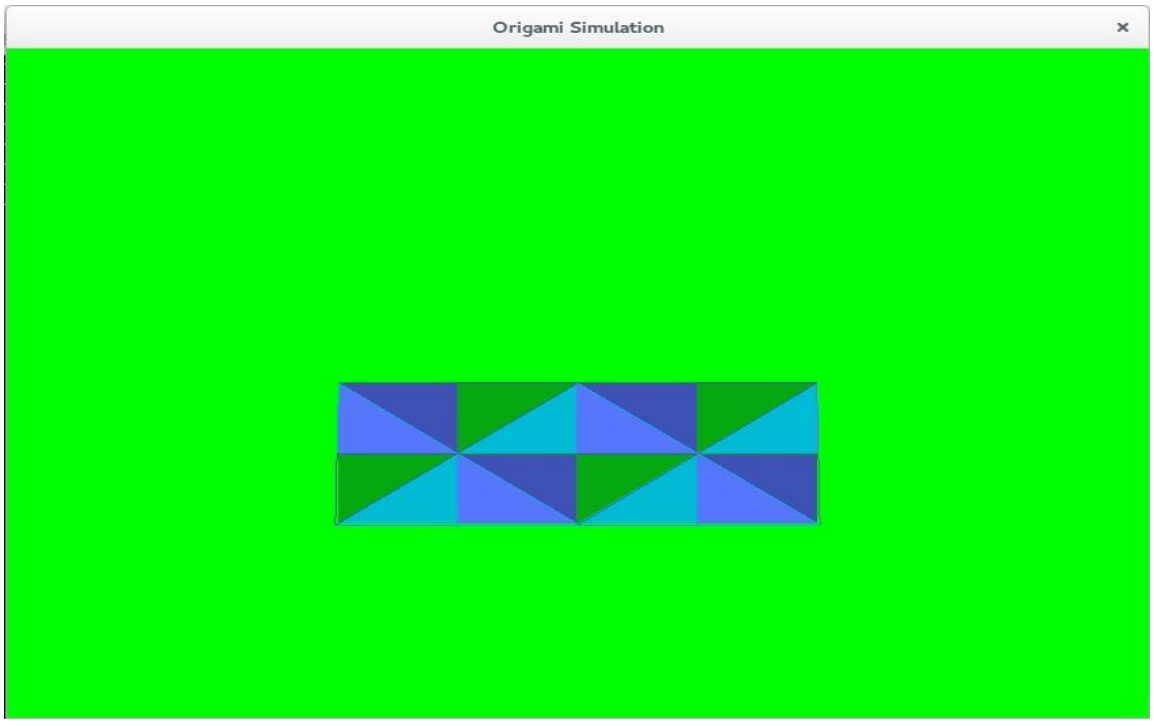
```
            }
            glutPostRedisplay();
            break;

        case 'Y': turn1 = (turn1 -5)%360;
            glutPostRedisplay();
            break;
        default:
            break;
      }
   }
}
else if (stage == 3){

   stage3_mode = 0;
   switch (key) {
      case 'd':

         xanglecount+= 2;
         xangle = xanglecount % 90;
         valy +=.2;
         valz = computeZ(valy);
         transvalx += .2;
         transvaly+= .007;

         glutPostRedisplay();
         break;

      default:
         break;
   }

}

switch (key) {

      case 'i':
               stage3_mode = 1;
         world_angles_true[0] = 1;
         world_angles[0] += 1;
         world_angles[0] % 360;
         glutPostRedisplay();
         break;
      case 'j':
               stage3_mode = 1;
```

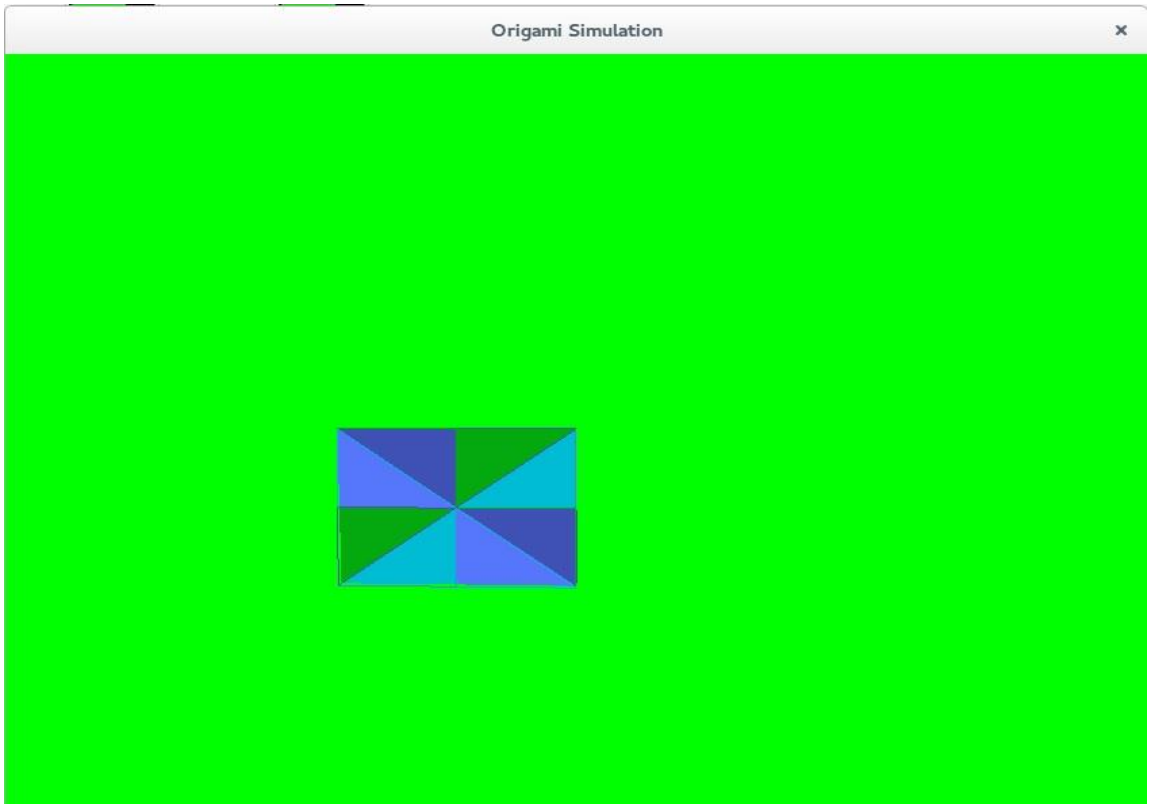
```
            world_angles_true[1] = 1;
            world_angles[1] += 1;
            world_angles[1] % 360;
            glutPostRedisplay();
            break;
         case 'k':
                  stage3_mode = 1;
            world_angles_true[2] = 1;
            world_angles[2] += 1;
            world_angles[2] % 360;
            glutPostRedisplay();
            break;
         case 'z':
         displayFrontScreen = 0;
         secondInit();
         glutPostRedisplay();
         glFlush();
         break;

            default: stage3_mode = 0;
            break;
   }
}

int main(int argc, char ** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
   glutInitWindowPosition(50, 50);
   glutInitWindowSize(600, 600);
   glutCreateWindow("Origami Simulation");
   init();
   glEnable(GL_DEPTH_TEST);
   glutDisplayFunc(display);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
return 0;
}
```

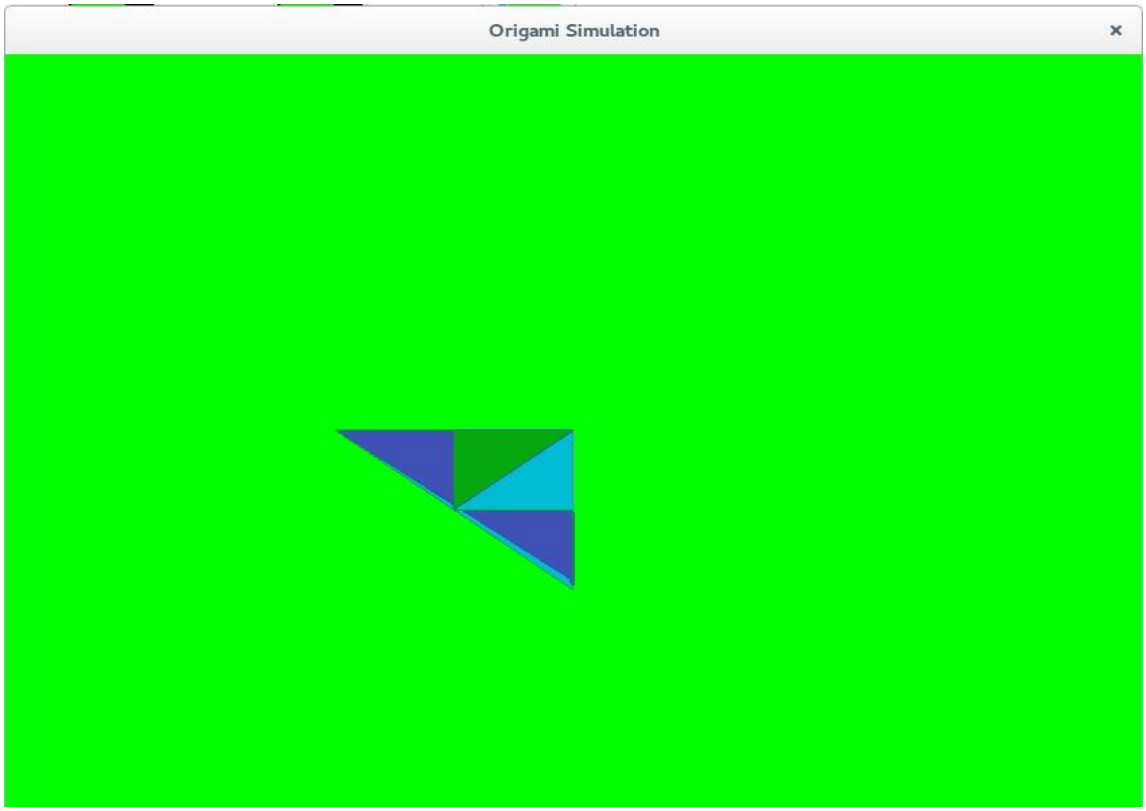
6. SAMPLE OUTPUT

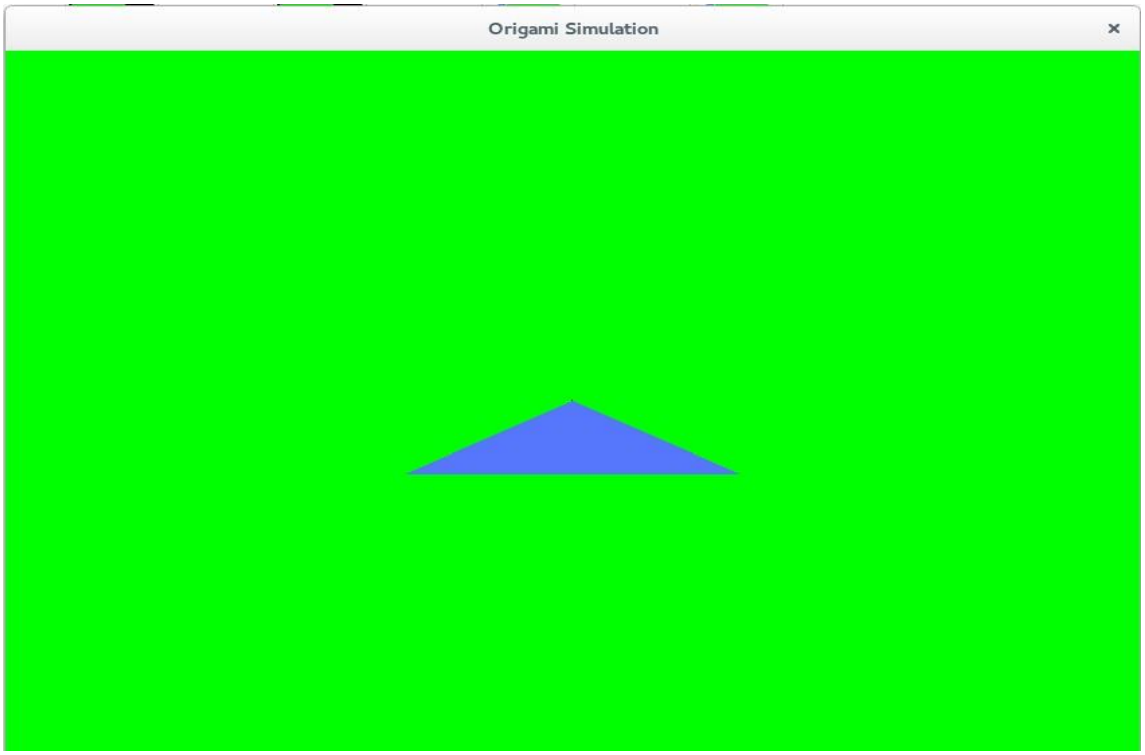


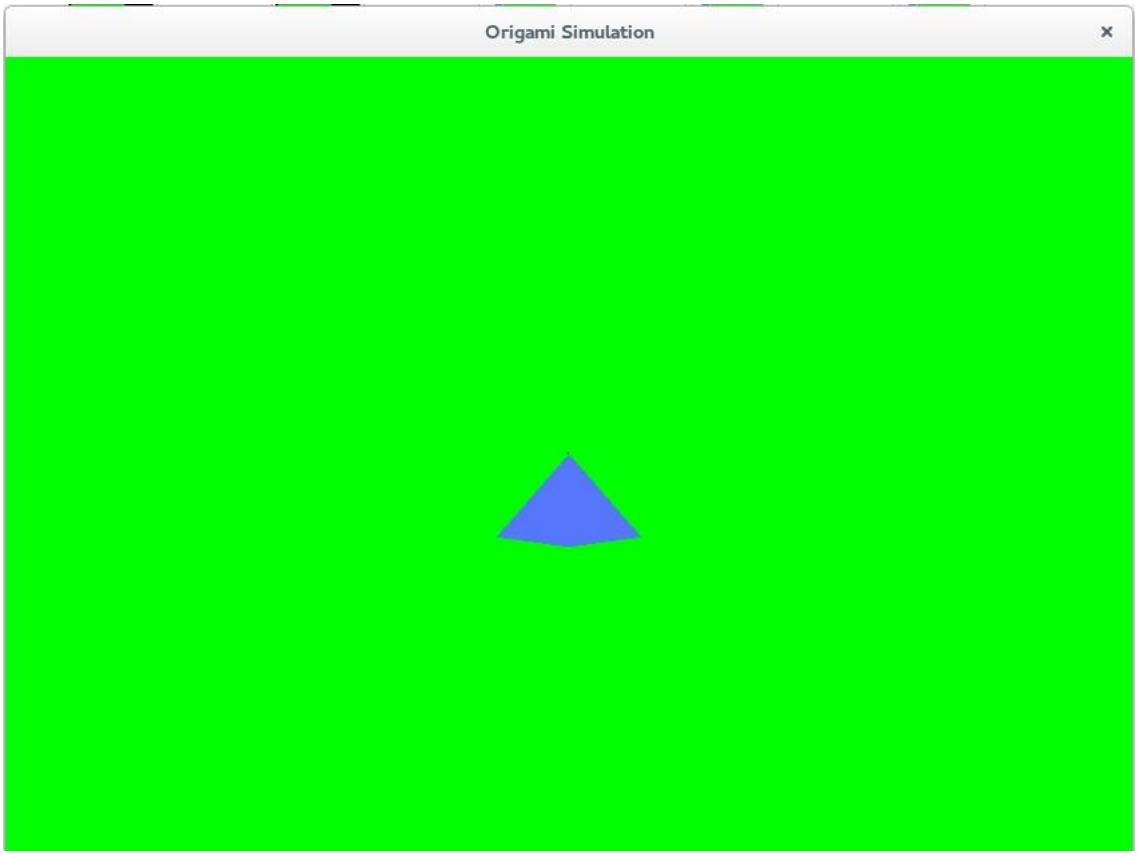

Origami Simulation

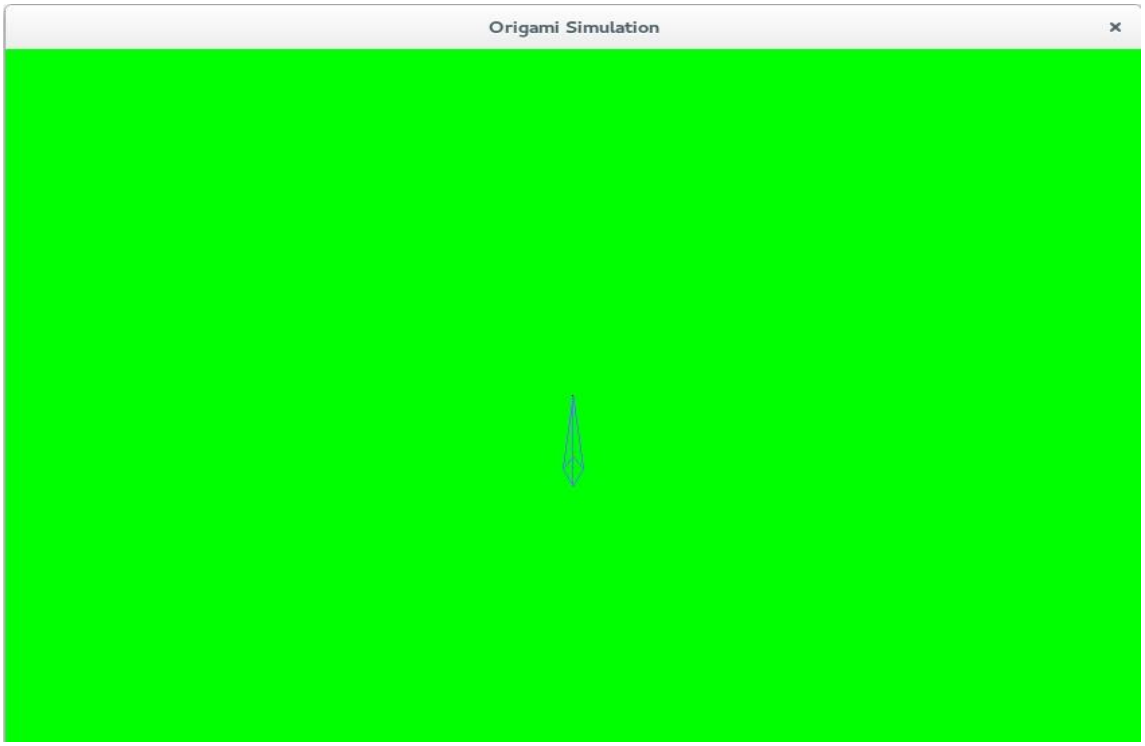

Origami Simulation

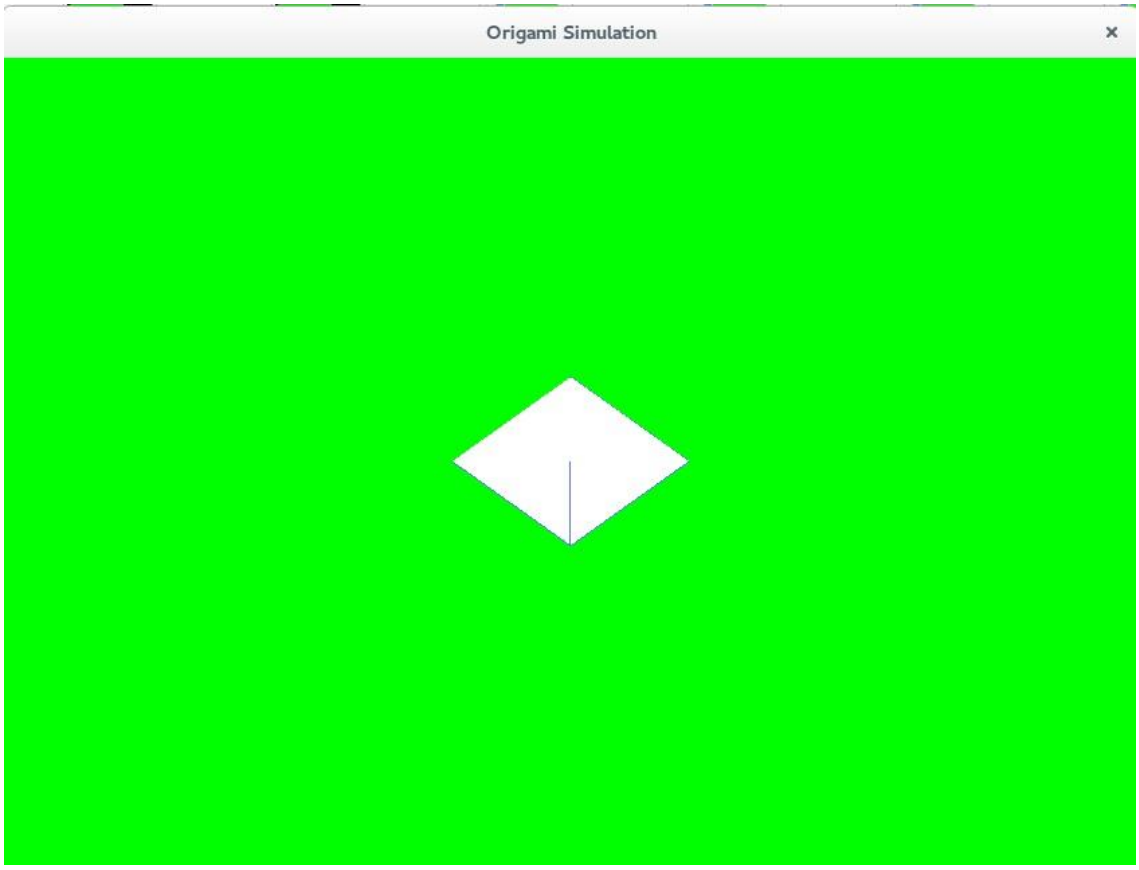

Origami Simulation

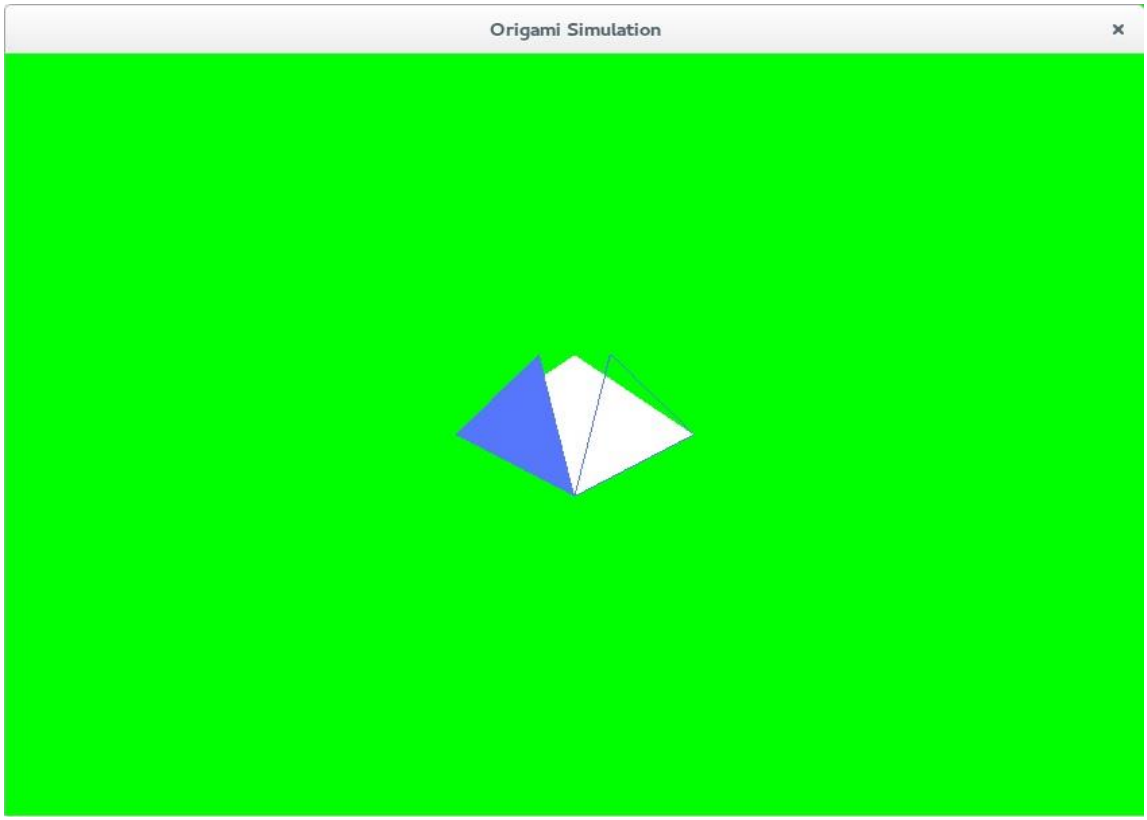

Origami Simulation

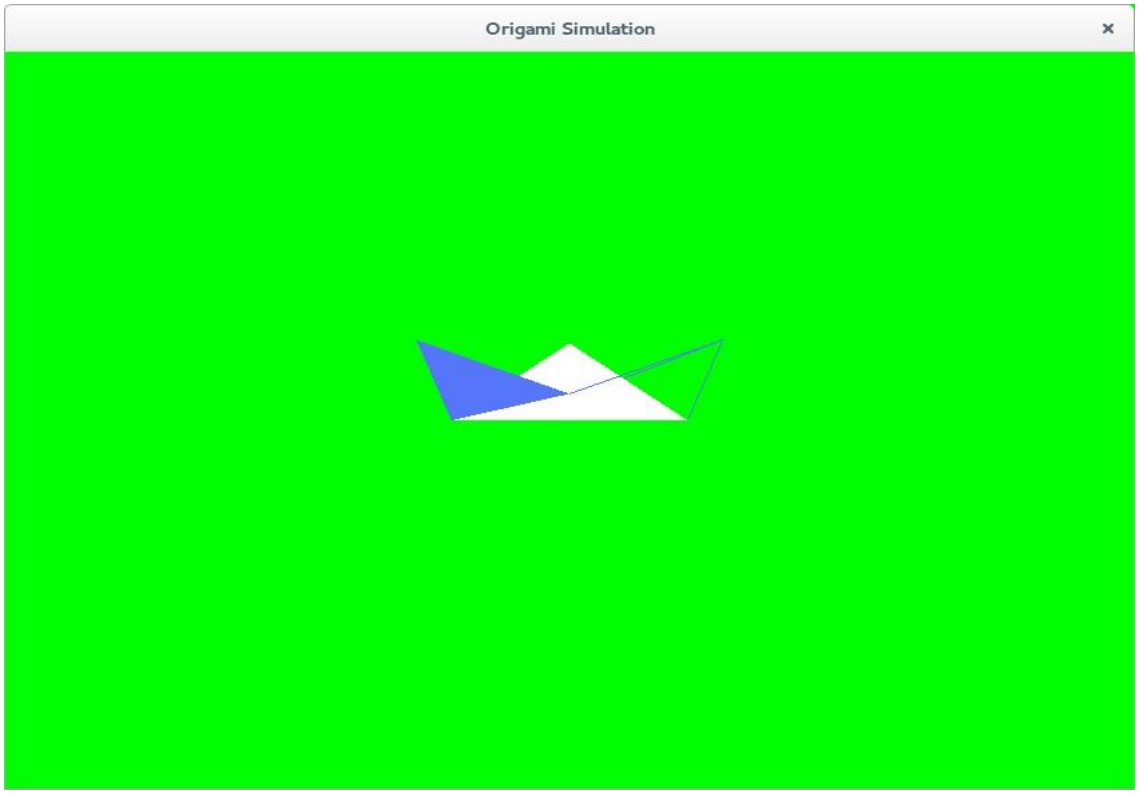

Origami Simulation

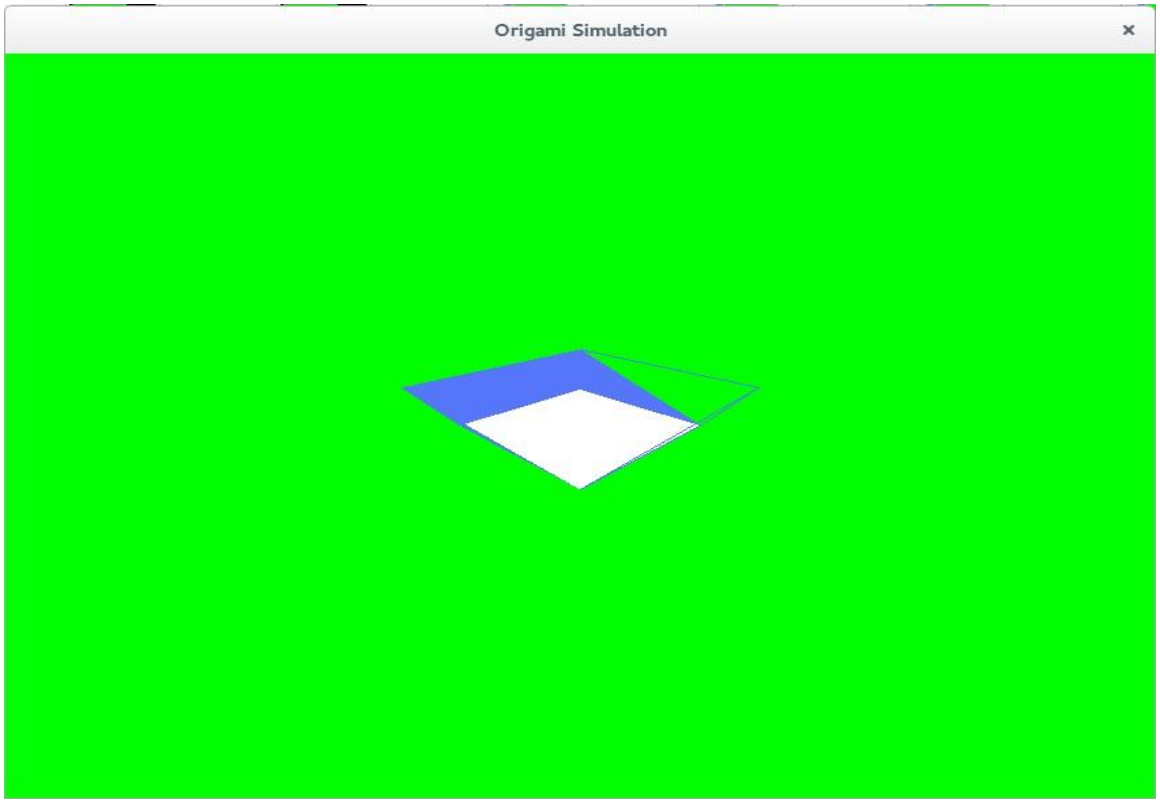

Origami Simulation

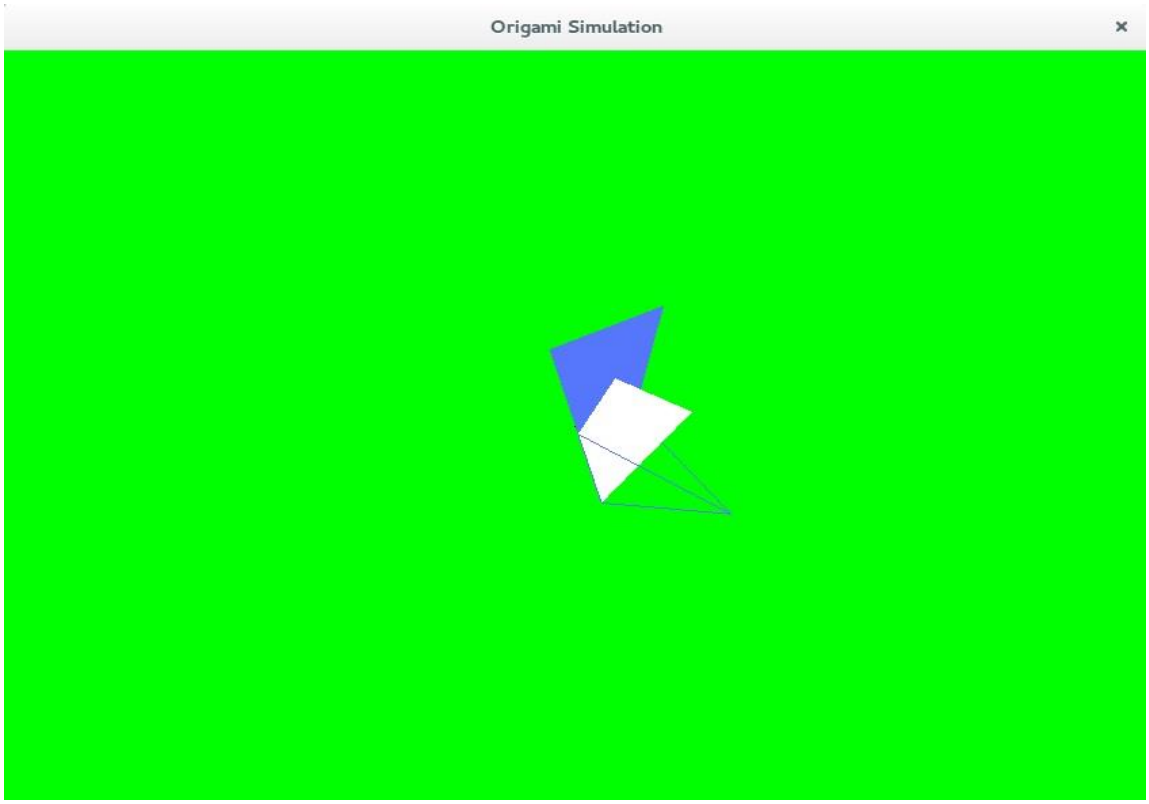

Origami Simulation


Origami Simulation


Origami Simulation


Origami Simulation


Origami Simulation


Origami Simulation

7. CONCLUSION

This project has helped me to understand the different concept used in Computer Graphics and Visualization lab. The use of translation and rotation throughout the context of the program has helped in better understanding the ModelView transformations and Matrix transformations in general.

While developing the project , we came across various methods to implement paper folding mechanism using rectangular mesh. Binding the coordinates of respective triangles when making the folds using constraint equations has been proves really helpful. Users can tap over various keys in the keyboard to perform specific actions.

8. BIBLIOGRAPHY

- **Reference Books**

1. Interactive Computer Graphics A Top-Down Approach with OpenGL- Edward Angel, 5th Edition, Addison-Wesley, 2008
2. Computer Graphics Using OpenGL – F.S Hill, Jr 2nd Edition, Pearson Education, 2001.
3. OpenGL Programming Guide or RED BOOK.
4. Computer Graphics: Principles and Practice in C: by Addison Wesley.

- **Reference Links**

1. http://en.wikipedia.org/wiki/OpenGL
2. http://www.opengl.org/
3. http://www.opengl.org/resourcers/libraries/glut