PROGRASSIVE EDUCATION SOCIETY'S

MODERN COLLEGE

OF

ARTS, SCIENCE & COMMERCE

Pune – 411 005

ACADEMIC YEAR 2016-2017

Project Report On

"**FIGHTER JET SIMULATION**"

BY

SR No.	Name	Roll Number
1	Vishal S. Kokane.	54065
2	Saurabh V. Deshpande.	54026
3	Vijay K. Ghule.	54037

TABLE OF CONTENTS

INTRODUCTION

This report contains implementation of 'FIGHTER JET SIMULATION ' using a set of OpenGL functions.. We are mainly using keyboard and mouse as interface to control the jet, to accelerate, control it's movements. The objects are drawn by using GLUT functions. This project has been developed using FEDORA 8 with OpenGL package.

1.1 Computer Graphics

Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

Computer graphics started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computers themselves. It has grown to include the creation, storage, and manipulation of models and images of objects. These models come from a diverse and expanding set of fields, and include physical, mathematical, engineering, architectural, and even conceptual structures, natural phenomena, and so on. Computer graphics today is largely interactive. The user controls the contents, structure, and appearance of the objects and of their displayed images by using input devices, such as keyboard, mouse, or touch-screen. Due to

close relationships between the input devices and the display, the handling of such devices is included in the study of computer graphics. The advantages of the interactive graphics are many in number. Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D patter-recognition abilities allow us to perceive and process data rapidly and efficiently. In many design, implementation, and construction processes today, the information pictures can give is virtually indispensable. Scientific visualization became an important field in the 1980s when the scientists and engineers realized that they could not interpret the prodigious quantities of data produced in supercomputer runs without summarizing the data and highlighting trends and phenomena in various kinds of graphical representations.

1.2OpenGL Interface

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects. Most of our applications will be designed to access OpenGL directly through functions in three libraries. They are:

1. Main GL: Library has names that begin with the letter gl and are stored in a library usually referred to as GL.

2. OpenGL Utility Library (GLU): This library uses only GL functions but contains code for creating common objects and simplifying viewing.

3. OpenGL Utility Toolkit(GLUT): This provides the minimum functionality that should be accepted in any modern windowing system.

1.3OpenGL Overview :

- OpenGL (Open Graphics Library) is the interface between a graphic program and graphics hardware. It is streamlined. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.
- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- It is *system-independent*. It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- It is a *state machine*. At any moment during the execution of a program there is a current model transformation
- It is a *rendering pipeline*. The rendering pipeline consists of the following steps:
    - Defines objects mathematically
    - Arranges objects in space relative to a viewpoint.
    - Calculates the color of the objects
    - Rasterizes the objects.

Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics is the most important means of producing pictures since the invention of

photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results..

OpenGL (open graphics library) is a standard specification defining a cross language cross platform API for writing applications that produce 2D and 3D computer graphics. OpenGL was developed by silicon graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization and flight simulation. It is also used in video games.

OpenGL serves two main purpose:

- To hide the complexities of interfacing with different 3D accelerators, by presenting programmer with a single, uniform API
- To hide the differing capabilities of hardware platforms, by requiring that all Implementations support the full openGL, feature set.

OpenGL has historically been influential on the development of 3D accelerator, promoting a base level of functionality that is now common in consumer level hardware:

- Rasterized points, lines and polygons are basic primitives.
- A transform and lighting pipeline.
- Z buffering.
- Texture Mapping.
- Alpha
- Blending

SYSTEM REQUIREMENTS SPECIFICATION

2.1 HARDWARE REQUIREMENTS

- Microprocessor: 1.0 GHz and above CPU based on either AMD or INTEL Microprocessor Architecture
- Main memory : 2 GB RAM
- Hard Disk : 40 GB
- Hard disk speed in RPM:5400 RPM
- Keyboard: QWERTY Keyboard
- Mouse :2 or 3 Button mouse
- Monitor : 1024 x 768 display resolution

2.2 SOFTWARE REQUIREMENTS

- Programming language – C/C++ using OpenGL
- Operating system – Linux operating system
- Compiler – C Compiler
- Graphics library – GL/glut.h
- OpenGL 2.0

2.3:FUNCTIONAL REQUIREMENTS:

OpenGL APIs:

If we want to have a control on the flow of program and if we want to interact with the window system then we use OpenGL API'S. Vertices are represented in the same manner internally, whether they are specified as two-dimensional or three-dimensional entities, everything that we do are here will be equally valid in three dimensions. Although OpenGL is easy to learn, compared with other APIs, it is nevertheless powerful. It supports the simple three dimensional programs and also supports the advanced rendering techniques.

GL/glut.h:

We use a readily available library called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system. The application program uses only GLUT functions and can be recompiled with the GLUT library for other window system. OpenGL makes a heavy use of macros to increase code readability and avoid the use of magic numbers. In most implementation, one of the include lines

ABOUT THE PROJECT

3.1 Overview:

This Project is on "**FIGHTER JET SIMULATION**" Computer Graphics using *OpenGL* Functions. It is a User interactive program where in the User can view the required display by making use of the input devices like Keyboard and Mouse. This project mainly consists of an jet. The jet movement is done with the help of the Keyboard.

3.2 User interface

A set of keys are used to change the following:

- Acceleration of the jet is controlled by 'up arrow'.
- De-acceleration is done using the key 'down arrow'.
- Movement of the jet is controlled by the keys 'left arrow'-left and 'right arrow'-right.
- Z or z: add 20 points in the direction of motion
- F or f: Enter game mode
- L:Exit game mode
- P or p:Pause the game
- Q or q and space :Fire bullet from jet
- S or s :Set up shield flag

3.3 OBJECTIVE:

- As Linux doesn't provide graphics editor, it should be designed in such a way that it provides a very useful graph implementation interface.
- It should be easy to understand, user interactive interface.
- Creation of primitives, i.e. polygons
- Providing human interaction through Mouse and keyboard.

Methods Used in the Project:

The main() Program consist of following functions:

void glutInit (int *argc, charargv):-**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 mode. The value of mode is determined by the logical OR of options including the color model and buffering

void glutInitWindowSize (int width, int height):- Specifies the initial position of the topleft corner of the window in pixels

glutInitCreateWindow (char *title):-A window on the display.The string title can be used to label the window. The return value provides references to the window that can be used when there are multiple windows.

void glutMainLoop ():- Cause the program to enter an event-processing loop.It should be the last statement in main function

glColor3f (float, float, float):-This function will set the current drawing color

glClear( ):-Takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.

glClearColor ():-Specifies the red, green, blue, and alpha values used by glClear to clear the color buffers.

GlLoadIdentity( ):-the current matrix with the identity matrix

glMatrixMode(mode):-Sets the current matrix mode, mode can be GL_MODELVIEW, GL_PROJECTION or GL_TEXTURE.

void glutKeyboardFunc(void(*func) (void)):-This function is called every time when you press ‘P’ key to resume the game or or when you press esc key to exit from the application.

void glutDisplayFunc (void (*func) (void)):-Register the display function func that is executed when the window needs to be redrawn

TESTING

Testing process started with the testing of individual program units such as functions or objects. These were then integrated into sub-systems and systems, and interactions of these units were tested.

Testing involves verification and validation.

Validation: “Are we building right product?”

Verification: “Are we building the product right?”

The ultimate goal of the verification and validation process is to establish confidence that the software system is ‘fit for purpose’. The level of required confidence depends on the system’s purpose, the expectations of the system users and the current marketing environment for the system.

With the verification and validation process, there are two complementary approaches to the system checking and analysis:

Software inspections or peer reviews analyses and check system representations such as the requirements document, design diagrams, and the program source code. Software testing involves running an implementation of the software with test data.

SOURCE CODE:

❖ **File name:FighterJet.c**

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <glut.h>
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif
void initWindow(void);
float angle = 0.0;
int left, right;
int leftTime, rightTime;
int thrust, thrustTime;
float x = 20, y = 20, xv, yv, v;
int shield = 0, joyShield = 0, cursor = 1;
int lastTime;
int paused = 0;
int resuming = 1;
int originalWindow = 0, currentWindow;
```

```
typedef struct
 {
  int inuse;
 float x;
  float y;
  float v;
  float xv;
  float yv;
  int expire;
} Bullet;

#define BULLETS 100
Bullet bullet[BULLETS];
int
allocBullet(void)
{
  int i;
  for (i=0; i<BULLETS; i++)
        {
              if (!bullet[i].inuse)
               {
                    return i;
```

```
                }
        }
  return -1;
}
void
initBullet(int i, int time)
{
  float c = cos(angle*M_PI/180.0);
  float s = sin(angle*M_PI/180.0);
  bullet[i].inuse = 1;
  bullet[i].x = x + 3.5 * c;
  bullet[i].y = y + 3.5 * s;
  bullet[i].v = 0.025;
  bullet[i].xv = xv + c * bullet[i].v;
  bullet[i].yv = yv + s * bullet[i].v;
  bullet[i].expire = time + 600;
}
void
advanceBullets(int delta, int time)
{
  int i;
  for (i=0; i<BULLETS; i++)
```

```
        {
            if (bullet[i].inuse)
            {
                float x, y;
                if (time > bullet[i].expire)
                {
                    bullet[i].inuse = 0;
                    continue;
                }
                x = bullet[i].x + bullet[i].xv * delta;
                y = bullet[i].y + bullet[i].yv * delta;
                x = x / 40.0;
                bullet[i].x = (x - floor(x))*40.0;
                y = y / 40.0;
                bullet[i].y = (y - floor(y))*40.0;
            }
        }
    }
void
shotBullet(void)
{
    int entry;
```

```
  entry = allocBullet();
  if (entry >= 0)
        {
              initBullet(entry, glutGet(GLUT_ELAPSED_TIME));
        }
}
void
drawBullets(void)
{
  int i;
  glBegin(GL_POINTS);
  glColor3f(1.0, 0.0, 1.0);
  for (i=0; i<BULLETS; i++) {
    if (bullet[i].inuse)
        {
              glVertex2f(bullet[i].x, bullet[i].y);
        }
  }
    glEnd();
}
```

```
void
drawshield(void)
{
        float rad;
         glColor3f(0.1, 0.1, 1.0);
         glBegin(GL_LINE_LOOP);
        for (rad=0.0; rad<11.0; rad += 1.0)
        {
                glVertex2f(3.75 * cos(2*rad/M_PI)+0.4, 3.0 *
sin(2*rad/M_PI));
        }
         glEnd();
}
void
drawJet(float angle)
{
  glPushMatrix();
  glTranslatef(x, y, 0.0);
  glRotatef(angle, 0.0, 0.0, 1.0);
  if (thrust)
        {
                glColor3f(1.0, 0.0, 0.0);
                glBegin(GL_LINE_STRIP);
```

```
            glVertex2f(-0.75, -0.5);
            glVertex2f(-1.75, 0);
            glVertex2f(-0.75, 0.5);
            glEnd();
        }
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(3.5,0.0);
    glVertex2f(0.5, -0.25);
    glVertex2f(-1.0, -1.0);
    glVertex2f(-0.5, 0.0);
    glVertex2f(-1.0, 1.0);
    glVertex2f(0.5,0.25);
    glVertex2f(3.5, 0.0);
    glEnd();
    if (shield)
        {
            drawshield();
         }
        glPopMatrix();
}
```

```
void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  drawJet(angle);
  drawBullets();
  glutSwapBuffers();
}
void idle(void)
{
  int time, delta;
  time = glutGet(GLUT_ELAPSED_TIME);
  if (resuming)
        {
              lastTime = time;
              resuming = 0;
        }
  if (left)
       {
              delta = time - leftTime;
              angle = angle + delta * 0.4;
              leftTime = time;
       }
```

```
if (right)
    {
        delta = time - rightTime;
        angle = angle - delta * 0.4;
        rightTime = time;
    }
if (thrust)
    {
        delta = time - thrustTime;
        v = delta * 0.00004;
        xv = xv + cos(angle*M_PI/180.0) * v;
        yv = yv + sin(angle*M_PI/180.0) * v;
        thrustTime = time;
    }
delta = time - lastTime;
x = x + xv * delta;
y = y + yv * delta;
x = x / 40.0;
x = (x - floor(x))*40.0;
y = y / 40.0;
y = (y - floor(y))*40.0;
lastTime = time;
```

```
  advanceBullets(delta, time);
  glutPostWindowRedisplay(currentWindow);
}

void visible(int vis)
{
  if (vis == GLUT_VISIBLE)
        {
              if (!paused)
                    {
                          glutIdleFunc(idle);
                    }
        }
 else
         {
              glutIdleFunc(NULL);
         }
}
void key(unsigned char key, int px, int py)
{
  switch (key)
         {
```

```
        case 27:
                exit(0);
                break;
        case 'S':
        case 's':
                shield = 1;
                break;
        case 'C':
        case 'c':
                cursor = !cursor;
                glutSetCursor(cursor ?
GLUT_CURSOR_INHERIT : GLUT_CURSOR_NONE);
                break;
        case 'z':
        case 'Z':
                x = 20;
                y = 20;
                xv = 0;
                yv = 0;
                break;
        case 'f':
        case 'F':
```

```
                glutGameModeString("400x300:16@60");
                glutEnterGameMode();
                initWindow();
                break;
        case 'l':
                if (originalWindow != 0 && currentWindow != originalWindow)
                {
                glutLeaveGameMode();
                currentWindow = originalWindow;
                }
                break;
        case 'P':
        case 'p':
                paused = !paused;
                if (paused)
                 {
                        glutIdleFunc(NULL);
                }
                 else
                 {
                        glutIdleFunc(idle);
                        resuming = 1;
```

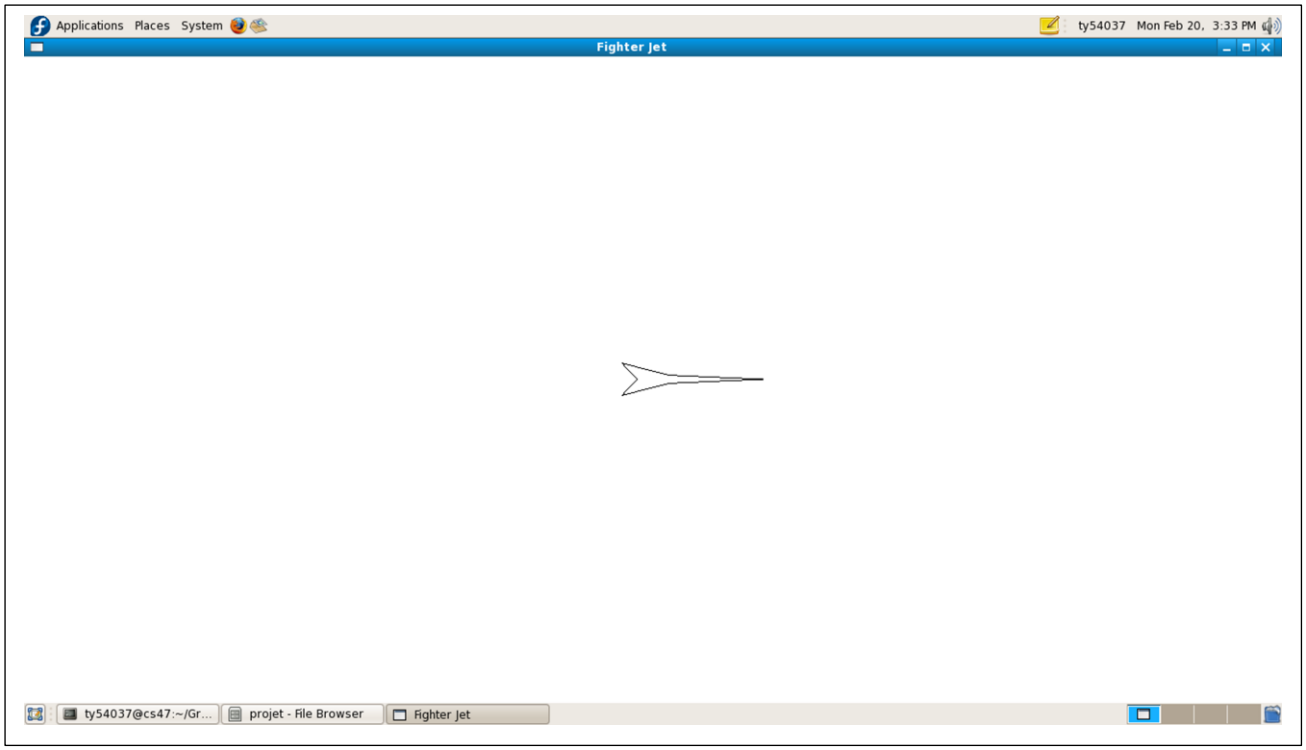
```
                    }
                    break;
        case 'Q':
        case 'q':
        case ' ':
                shotBullet();
                break;
  }
}
void keyup(unsigned char key, int x, int y)
{
  switch (key)
         {
        case 'S':
        case 's':
        shield = 0;
        break;
        }
}

void special(int key, int x, int y)
{
```

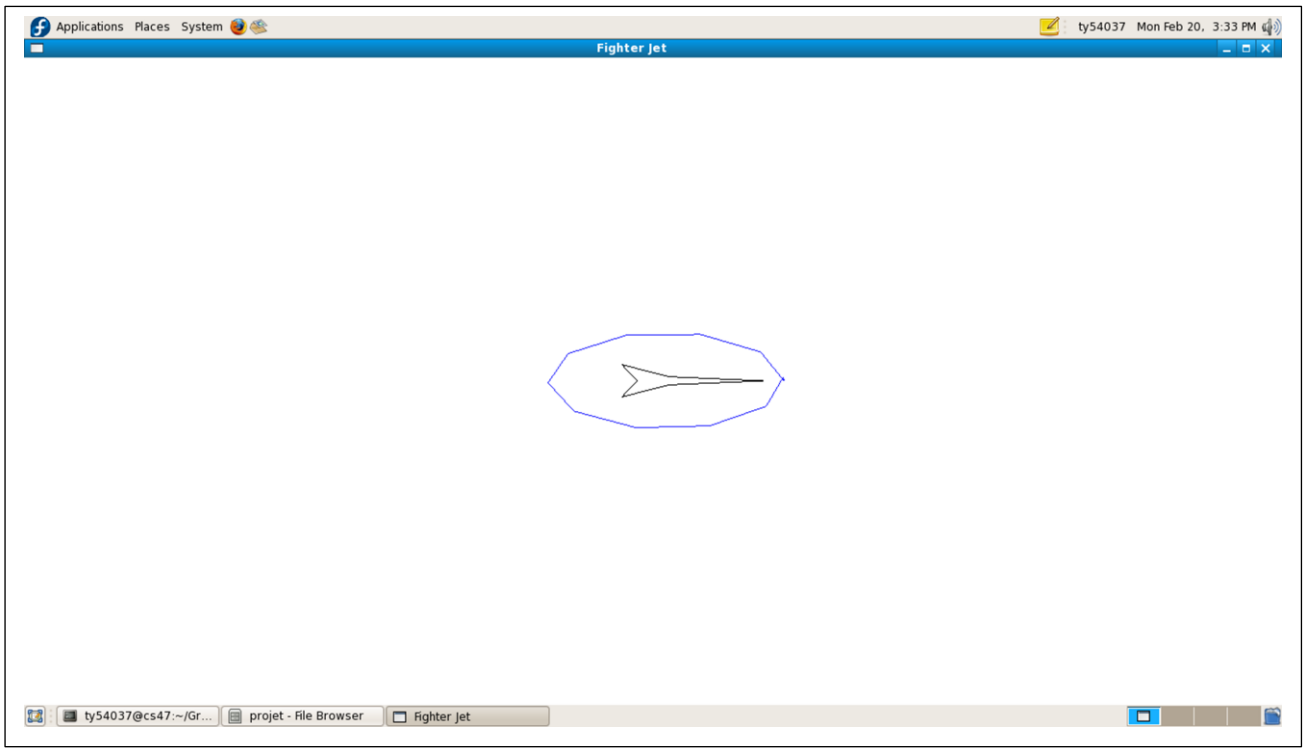
```
  switch (key)
        {
       case GLUT_KEY_UP:
             thrust = 1;
             thrustTime = glutGet(GLUT_ELAPSED_TIME);
             break;
       case GLUT_KEY_LEFT:
             left = 1;
             leftTime = glutGet(GLUT_ELAPSED_TIME);
             break;
       case GLUT_KEY_RIGHT:
             right = 1;
             rightTime = glutGet(GLUT_ELAPSED_TIME);
             break;
       }
}
void specialup(int key, int x, int y)
{
  switch (key)
        {
       case GLUT_KEY_UP:
             thrust = 0;
```

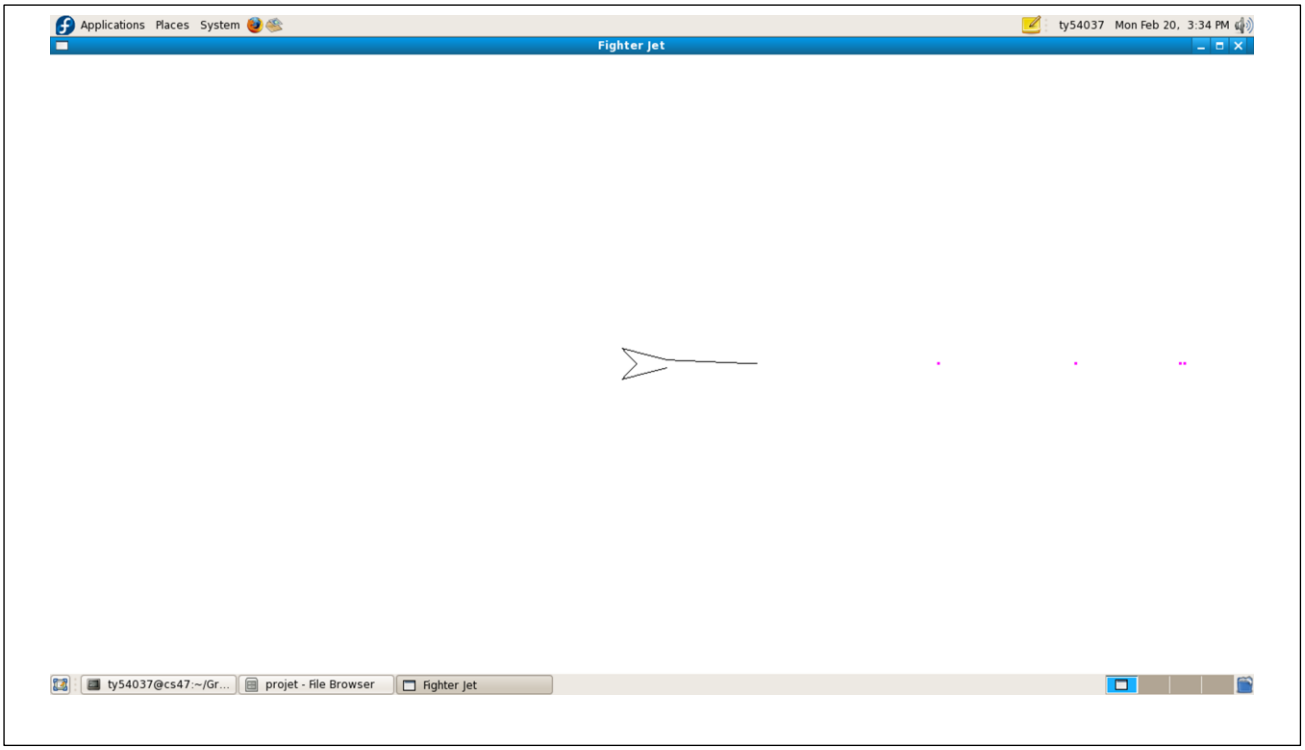
```
            break;
        case GLUT_KEY_LEFT:
            left = 0;
            break;
        case GLUT_KEY_RIGHT:
            right = 0;
            break;
        }
}
void initWindow(void)
{
  glutIgnoreKeyRepeat(1);
  glutDisplayFunc(display);
  glutVisibilityFunc(visible);
  glutKeyboardFunc(key);
  glutKeyboardUpFunc(keyup);
  glutSpecialFunc(special);
  glutSpecialUpFunc(specialup);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0, 40, 0, 40, 0, 40);
  glMatrixMode(GL_MODELVIEW);
```

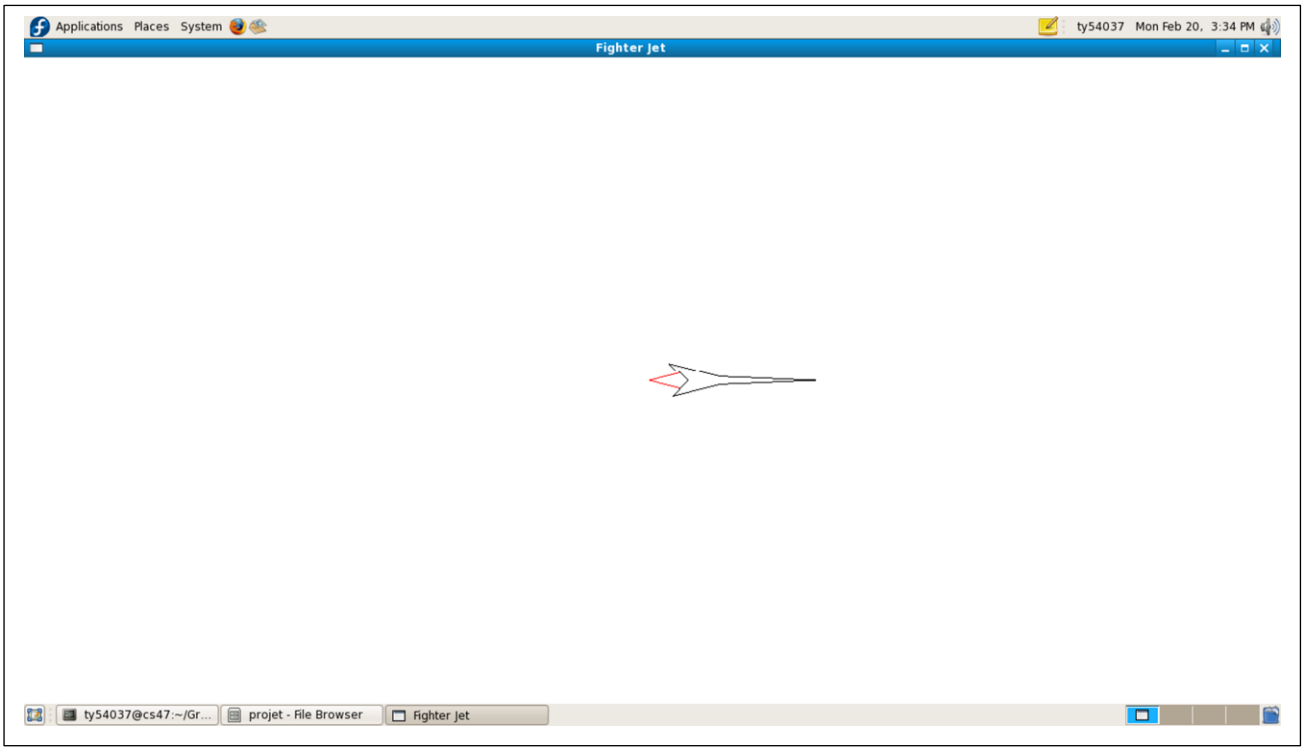
```
  glPointSize(3.0);
  currentWindow = glutGetWindow();
}
int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  if (argc > 1 && !strcmp(argv[1], "-fullscreen"))
         {
         glutGameModeString("640x480:16@60");
         glutEnterGameMode();
         }
 else
         {
         originalWindow = glutCreateWindow("Fighter Jet");
         }
  initWindow();
  glutMainLoop();
  return 0;
}
```

SNAPSHOTS


Applications Places System
ty54037 Mon Feb 20, 3:33 PM
Fighter Jet
ty54037@cs47:~/Gr...
projet - File Browser
Fighter Jet


Applications Places System
ty54037 Mon Feb 20, 3:33 PM
Fighter Jet
ty54037@cs47:~/Gr...
projet - File Browser
Fighter Jet


Applications Places System
ty54037 Mon Feb 20, 3:34 PM
Fighter Jet
ty54037@cs47:~/Gr...
projet - File Browser
Fighter Jet


Applications Places System
ty54037 Mon Feb 20, 3:34 PM
Fighter Jet
ty54037@cs47:~/Gr...
projet - File Browser
Fighter Jet

CONCLUSION

7.1 CONCLUSION

The FIGHTER JET SIMULATION has been tested under FEDORA 8, and has been found to provide ease of use and manipulation to the user. The FIGHTER JET SIMULATION created for FEDORA 8 operating system can be used to draw lines, boxes, circles, ellipses, and polygons. It has a very simple and effective user interface.

We found designing and developing this FIGHTER JET as a very interesting and learning experience. It helped us to learn about computer graphics, design of Graphical User Interfaces, interface to the user, user interaction handling and screen management. The graphics editor provides all and more than the features that have been detailed in the university syllabus.

7.2 FUTURE ENHANSMENT

These are the features that are planned to be supported in the future

* Simulating smoke from exhaust pipe

* Perform movement by jet

* To improve the looks of the object

BIBLIOGRAPHY

To Build this Project we have seeked help from following sites:

1.www.OpenGL.com

2.www.graphics.com

Project on 'FIGHTER JET SIMULATION'

Commands to execute:

In terminal, type the following command to compile the program

- "**gcc project.c -lglut -lGL –lGLU -lm**"

After compiling to run the program type

- **"./a.out"**