VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnana Sangama", Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V. Puram, Bengaluru-560 004

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

COMPUTER GRAPHICS & VISUALIZATION MINI PROJECT (17CSL68)

"AIRPLANE"

Submitted By

PRAJWAL P **1BI17CS111**

for the academic year 2019-20

Under the guidance of

Prof. N. Thanuja
Assistant Professor

Prof. Shruthiba A
Assistant Professor

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnana Sangama", Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V. Puram, Bengaluru-560 004

Department of Computer Science & Engineering

Certificate

This is to certify that the implementation of **Computer Graphics and Visualization MINI PROJECT (17CSL68)** entitled **"AIRPLANE"** has been successfully completed by **1BI17CS111 PRAJWAL P** of VI semester B.E. for the partial fulfillment of the requirements for the Bachelor's degree in **Computer Science & Engineering** of the **Visvesvaraya Technological University** during the academic year **2019-2020**.

Lab In-Charges:

Prof. N. Thanuja	**Prof. Shruthiba A**	**Dr. Asha T.**
Assistant Professor	Assistant Professor	Professor and HOD
Dept. of CS&E, BIT	Dept. of CS&E, BIT	Dept. of CS&E, BIT

Examiners: 1) 2)

ACKNOWLEDGEMENT

The knowledge & satisfaction that accompany the successful completion of any task would be incomplete without mention of people who made it possible, whose guidance and encouragement crowned my effort with success. I would like to thank all and acknowledge the help I have received to carry out this Mini Project.

I would like to convey my thanks to **Dr. ASHA T.,** Professor and Head, Dept. of CSE, BIT, for being kind enough to provide the necessary support to carry out the Mini project.

I am most humbled to mention the enthusiastic influence provided by the lab in-charges **Prof. N. THANUJA** and **Prof. SHRUTHIBA A**, on the project for their ideas, time to time suggestions for being a constant guide and co-operation showed during the venture and making this project a great success.

I would also take this opportunity to thank my friends and family for their constant support and help. I'm very much pleasured to express my sincere gratitude to the friendly co-operation showed by all the **staff members** of Computer Science Department, BIT.

PRAJWAL P
1BI17CS111

Table of contents

Chapter-1

INTRODUCTION

1.1 Computer Graphics

Computer graphics is an art of drawing pictures, lines, charts, using computers with the help of programming. Computer graphics is made up of number of pixels. Pixel is the smallest graphical picture or unit represented on the computer screen. Basically, there are 2 types of computer graphics namely,

Interactive Computer Graphics involves a two-way communication between computer and user. The observer is given some control over the image by providing him with an input device. This helps him to signal his request to the computer.

Non-Interactive Computer Graphics otherwise known as passive computer graphics it is the computer graphics in which user does not have any kind of control over the image. Image is merely the product of static stored program and will work according to the instructions given in the program linearly. The image is totally under the control of program instructions not under the user. Example: screen savers.

1.2 Applications of Computer Graphics

Scientific Visualization

Scientific visualization is a branch of science, concerned with the visualization of three-dimensional phenomena, such as architectural, meteorological, medical, biological systems.

Graphic Design

The term graphic design can refer to a number of artistic and professional disciplines which focus on visual communication and presentation

Computer-aided Design

Computer-aided design (CAD) is the use of computer technology for the design of objects, real or virtual. The design of geometric models for object shapes, in particular, is often called computer-aided geometric design (CAGD). The manufacturing process is tied in to the computer description of the designed objects so that the fabrication of a product can be automated using methods that are referred to as CAM, computer-aided manufacturing.

Web Design

Web design is the skill of designing presentations of content usually hypertext or hypermedia that is delivered to an end-user through the World Wide Web, by way of a Web browser.

Digital Art

Digital art most commonly refers to art created on a computer in digital form.

Video Games

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a raster display device.

Virtual Reality

Virtual reality (VR) is a technology which allows a user to interact with a computer simulated environment. The simulated environment can be similar to the real world. This allows the designer to explore various positions of an object. Animations in virtual reality environments are used to train heavy equipment operators or to analyze the effectiveness of various cabin configurations and control placements.

Computer Simulation

A computer simulation, a computer model or a computational model is a computer program, or network of computers, that attempts to simulate an abstract model of a particular system.

Education and Training

Computer simulations have become a useful part of mathematical modeling of many natural systems in physics, chemistry and biology, human systems in economics, psychology, and social science and in the process of engineering new technology, to gain insight into the operation of those systems, or to observe their behavior. Most simulators provide screens for visual display of the external environment with multiple panels is mounted in front of the simulator.

Image Processing

The modification or interpretation of existing pictures such as photographs and TV scans, is called image processing. In computer graphics, a computer is used to create a picture. Image processing techniques, on the other hand, are used to improve picture quality, analyze images, or recognize visual patterns for robotics applications.

1.3 OpenGL

OpenGL has become a widely accepted standard for developing graphics applications. Most of our applications will be designed to access OpenGL directly through functions in the three libraries. Functions in main GL libraries have names that begin with the letters gl and are stored in a library usually referred to as GL.

The second is the OpenGL Utility Library (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library. The GLU library is available in all OpenGL implementations. Functions in the GLU library starts with the letters glu.

The third is the OpenGL Utility Toolkit (GLUT). It provides the minimum functionality that should be formulated in modern windowing systems.

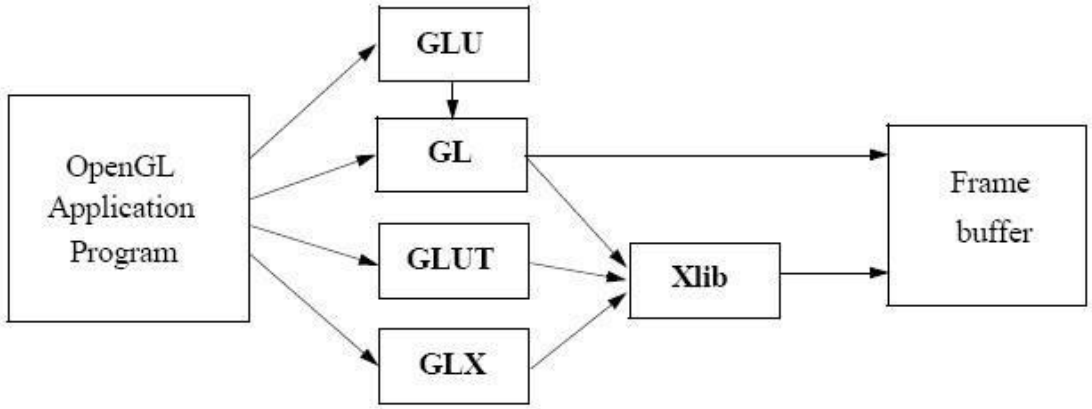

Figure 1.1: Basic block diagram of OpenGL

1.4 Problem Statement

The goal of this game is to fly as far as possible, this is one of those “easy to learn, difficult to master” situations. You’ll first select a plane and a scene. Click to start make your little guy take off from an airport, and it’s on!!

As you fly, left click to begin ascending; the plane descends automatically. So it’s critical to master this bouncy method of flying.

Missiles of varying sizes are shot at you. So, you’ll spend most of your time dodging these. Keep a close eye at the fuel gauge! You’ll steadily be running out of fuel as you try to gain height. Hit a missile, run out of fuel, or fly into the bottom edge of the screen, and you’ll crash and burn!

As far as the game interface is concerned, you’ll be greeted with a loading screen followed by a splash screen with the name of the developers (that’s Prajwal and Revanth).

You'll then see a menu with screen with the following items:

- Play
- Settings
- Instructions
- Credits
- High Scores
- Exit

The user can either start the game directly, if he knows how the game works or has played it before, by clicking in the box with "Play" option. Otherwise, he can view the instruction as to the game works by clicking on the "Instruction" button. If the player has changed his mind to play the game sometime later, he can click on "Exit" button which terminates the game.

When the player hits on the Instruction button, another page which describes how the game works appears. Similarly, credits page will show you the name of the developers. Setting page will allow you to choose the plane and environment scene of your wish. Pressing escape at any point of time will bring you to the main page.

We will be using C/C++ along with OpenGL to develop this project.

1.5 Objectives of the Project

- To guide the airplane before you run out of fuel!
- To show the implementation of Textures for a better appearance of the real-world objects.
- To show the implementation of the OpenGL transformation functions.
- To let the user play in full screen and specified window size.
- To let user choose plane and background scene.
- To show high score of the game.

1.6 Organization of the Report

At some point during the implementation of a project, a project report has to be generated in order to paint a mental image of the whole project. Ultimately, a project report must maximize the insight gained with minimal effort from the reader. Apart from describing its results, it must also explain the implications of those results to the organization.

Chapter-2

SYSTEM REQUIREMENTS

2.1 Software Requirements

Operating System	Windows 10
Editor	Visual Studio 2019
Toolkit	Freeglut, Glew, Soil2
Language	C++

2.2 Hardware Requirements

CPU	Intel/AMD CPU
Secondary Memory	10 MB
RAM	512 MB
Cache Memory	256 KB
Mouse	Standard mouse
Keyboard	Standard qwerty keyboard
Monitor	1366*768 display resolution

Chapter-3

DESIGN

3.1 Flow Diagram

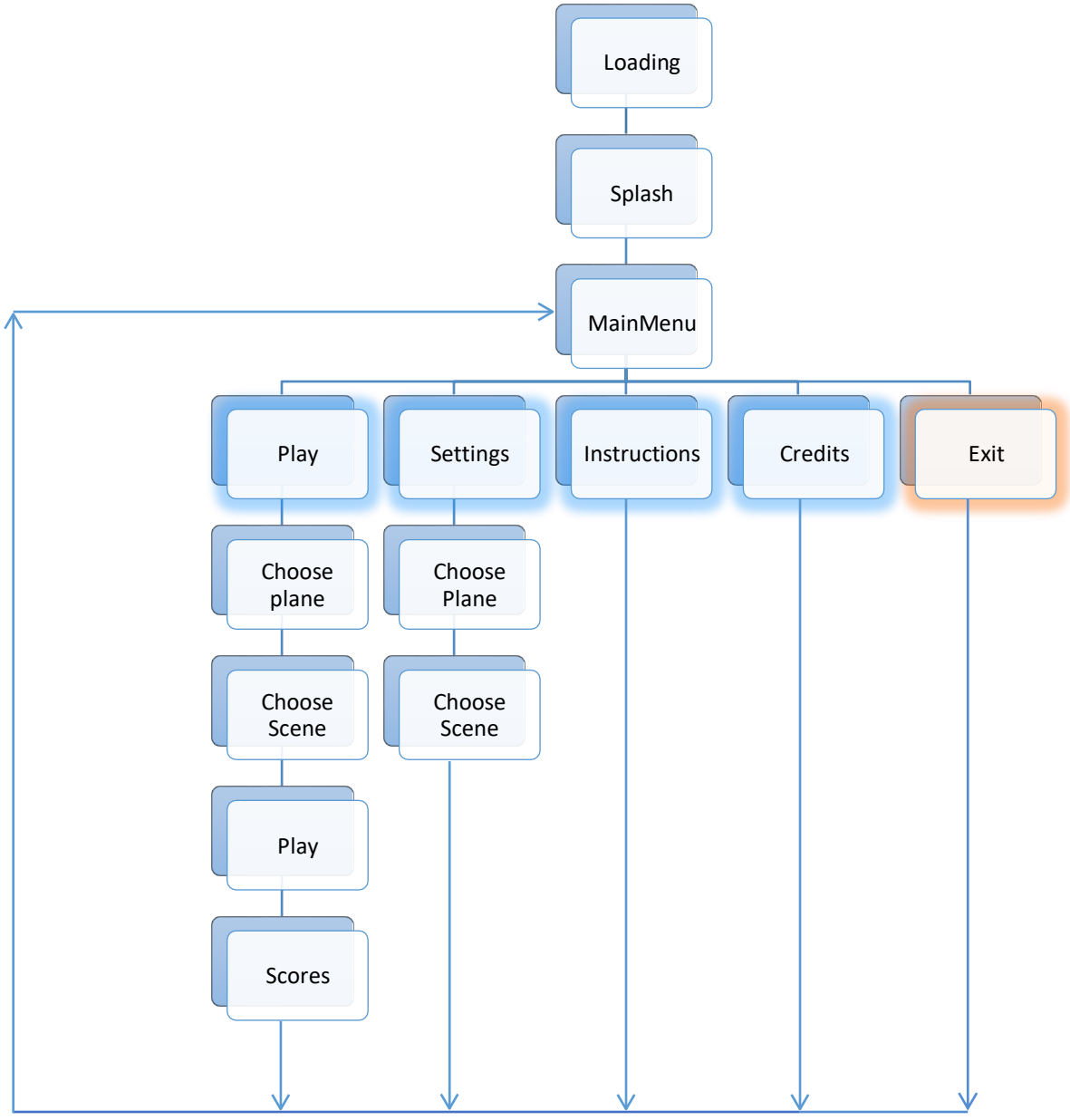

Figure 3.1: Flow diagram of Scene Change

3.2 Description of Flow Diagram

The description of the flow diagram is as follows:

Step1: Start

Step2: Loading screen

Step3: Splash screen

Step4: Menu screen - User can choose to play, settings, view instructions, credits, high scores or exit

Step5: If user chooses

Play - User can play the game

Settings - User can choose plane and background scene

Instructions - User can see how to play

Credits - User can see details of developers

High Score - User can view the highest score

Exit – User exits from the application

Step6: Stop

Chapter – 4

IMPLEMENTATION

4.1 Built In Functions

The following APIs have been used in this project. The API name along with its description is as follows:

1. glutMainLoop(void);

It causes the program to begin an event-processing loop.

2. glutReshapeFunc(void (GLUTCALLBACK *func)(int width, int height));

The reshape event is generated whenever the window is resized, such as by a user interaction.

3. glutSwapBuffers(void);

We can swap the front and back buffers at will from the application programs**.**

4. glutStrokeCharacter(void *font, int character);

Without using any display lists, glutStrokeCharacter renders the character in the named stroke font.

5. glPushMatrix();

Set current matrix on the stack

6. glPopMatrix();

Pop the old matrix without the transformations.

7. void glTranslatef(GLfloat x, GLfloat y, GLfloat z);

glTranslate produces a translation by (x, y, z).

8. void glLineWidth(GLfloat width);

Specifies the rasterized width of both aliased and antialiased lines.

9. void glBindTexture(GLenum target, GLuint texture);

Lets you create or use a named texture.

10. void glTexEnvf(GLenum target, GLenum pname, GLfloat param);

Specifies the texture environment.

11. void glEnable(GLenum cap);

Enable server-side GL capabilities.

12. void glDisable(GLenum cap);

Disable server-side GL capabilities.

13. SetFont(string family [], string style [], float size);

Sets the font used to print character strings.

14. void glutTimerFunc(unsigned int msecs,void (*func)(int value), value);

Registers the timer callback function to be triggered in at least msecs milliseconds.

15. void glutPostRedisplay(void);

Mark the normal plane of current window as needing to be redisplayed.

16.void glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha);

Specifies the red, green, blue, and alpha values used by glClear to clear the color buffers.

17. void glutSpecialFunc(void (*func)(int key, int x, int y));

Sets the special keyboard callback for the current window.

18. void glutMainLoop(void);

Enters the GLUT event processing loop. This routine should be called at most once in a GLUT program.

19. glColor3f (GLfloat red, GLfloat green, GLfloat blue);

It sets color to current drawing.

20. glLoadIdentity (void);

To initialize the current transform matrix to the identity transform.

21. glMatrixMode (GLenum mode);

It switches matrix mode between the two matrices –

- MODEL_VIEW (GL_MODELVIEW)
- PROJECTION (GL_PROJECTION)

22. glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar);

It establishes as a view volume a parallelepiped that extends from left to right in x, bottom to top in y and near to far in z.

23. glVertex3f (GLfloat x, GLfloat y, GLfloat z);

It is used to represent vertex.

24. glViewport (GLint x, GLint y, GLsizei width, GLsizei height);

It specifies that the viewport will have lower left corner (x,y) in screen co-ordinates and will be width pixels wide and height pixels high.

25. glutBitmapCharacter(void *font, int character);

The character is placed at the present raster position on the display, is measured in pixels and can be altered by the various forms of the function glRasterPos*.

26. glutCreateWindow(const char *title);

It creates and opens OpenGL window with the title passed as the argument.

27. glutDisplayFunc(void (GLUTCALLBACK *func)(void));

It sends graphics to screen.

28. glutIdleFunc(void (GLUTCALLBACK *func)(void));

It is used to increase theta by fixed amount whenever nothing else is happening.

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

It initiates interaction between windowing system and OpenGL.

30. glutInitDisplayMode(unsigned int mode);

This function specifies how the display should be initialized.The constants GLUT_SINGLE and GLUT_RGB, which are ORed together, indicate that a single display buffer should be allocated and the colors are specified using desired amount of red, green and blue.

31. glutInitWindowSize(int width, int height);

It sets the size of created window.

32. glutKeyboardFunc(void (GLUTCALLBACK *func)(unsigned char key, int x, int y));

The keyboard event is generated when the mouse is in the window and one of the key is pressed or released. This GLUT function is the call back for event generated by pressing a key.

4.2 User-Defined Function

1. void drawString(float x,float y,float z,char* string);

Prints a string of Bitmap characters onto the screen at position determined by the coordinates (x,y,z)

2. void draw_fin_text();

Displays the score and number of missiles dodged, in the finish screen.

3. void draw_credit_text();

Displays credits in the credit screen (when page=22)

4. void draw_high_text();

Displays high scores and maximum missiles dodged by the player.

5. void draw_menu_text();

Draws menu items on the screen.

6. void draw_inst_text();

Displays instructions onto the screen

7. void draw_chScene_text();

Allows you to select the scene (background)

8. void draw_chPlane_text();

Allows you to choose the plane of your choise.

9. void draw_score();

Displays scores during game play

10. void select_scene();

Sets the scene variable to contain the name of the background image file based on the value of ch_scene variable.This variable (ch_scene) is updated in the specialkeys function.

11. void rocket1(int x_cor, int y_cor);

Draws the first kind of rocket onto the screen at position determined by (x,y) coordinates.

12. void rocket2(int x_cor, int y_cor);

Draws the second kind of rocket onto the screen at position determined by (x,y) coordinates.

13. void rocket3(int x_cor, int y_cor);

Draws the third kind of rocket onto the screen at position determined by (x,y) coordinates.

14. void draw_rockets();

Determines the number of rocket to be displayed on the screen at any point of time can then calls the rocket<X> function. (Here 1<=X<=3)

15. void draw_chosen_plane();

Display the chosen plane on the screen where you have to select a polane of your choice.

16. void drawLogo();

Displays the college logo in the credit page.

17. void RenderScene();

RenderScene is the display callback function which is responsible to display various onscreen elements and call the above functions based on the value of page variable.

Source Code

```
void keyboard(unsigned char key, int x, int y)
{
	switch (key)
	{
		//27 is the ASCII value of the ESC key
	case 27:
		x = 0.0;
		if (page != 2 && page != 1)
		{
			i_bck = 0, i_mis1 = 0, i_mis2 = 0, i_mis3 = 0, i_plane = 0, i_sel31 =
0, i_sel32 = 0, i_fin4 = 0;
			i_23 = 0, i_0 = 0, i_1 = 0;
			missiles = 0;
			dist = 0;
			setting = 0;
			hit_missile = 0;
			y_pos = 0;
			missile_x = 250;
			fuel = 98;
			y_cre = 0;
			x = 0.0;
			cout << "full =" << full << endl;
			cout << "x = " << x << endl;
			page = 2;
		}
		else
			//exit game
			exit(0);
		break;

	case 'f':	//full screen
		if (full == 0)
		{
			glutFullScreen();
			full = 1;
		}
		else
		{
			glutReshapeWindow(800, 450);
			glutPositionWindow(320, 150);
			full = 0;
		}
	}
}
```

```
//Special keys have been used to choose scene (in page 32)
void SpecialKeys(int key, int x, int y)
{
	if (page == 32)
	{
		switch (key)
		{
		case GLUT_KEY_UP:
		case GLUT_KEY_RIGHT:
			if (ch_scene < 9)
			{
				ch_scene++;
				i_s = 0;
				cout << ch_scene << endl;
			}
			break;
		case GLUT_KEY_DOWN:
		case GLUT_KEY_LEFT:
			if (ch_scene > 1)
			{
				ch_scene--;
				i_s = 0;
				cout << ch_scene << endl;
			}
			break;
		}
		glutPostRedisplay();
	}
}

//Determines the action on mouse click event
void Mouse(int button, int m_state, int m_x, int m_y)
{
	if (page == 1)
	{
		if (m_state == GLUT_UP)
		{
			cout << m_x << " " << m_y << endl;
			if(m_x > 620 && m_y > 260)
			page = 2;
		}
	}
	else if (page == 2)
	{
		if (full == 0)
		{
			if (m_state == GLUT_UP)
				cout << "full = " << full << endl;
				cout << m_x << " " << m_y << endl;
			if (m_y > 92 && m_y < 116 && m_state == GLUT_UP)
```

```
		{
			cout << "play" << endl;
			page = 31;
		}
		if (m_y > 126 && m_y < 149 && m_state == GLUT_UP)
		{
			cout << "Settings" << endl;
			page = 31;
			setting = 1;
		}
		if (m_y > 158 && m_y < 182 && m_state == GLUT_UP)
		{
			cout << "Instructions" << endl;
			page = 21;
		}
		if (m_y > 193 && m_y < 216 && m_state == GLUT_UP)
		{
			cout << "Credit" << endl;
			page = 22;
		}
		if (m_y > 226 && m_y < 256 && m_state == GLUT_UP)
		{
			cout << "High" << endl;
			page = 23;
			//exit(0);
		}
		if (m_y > 260 && m_state == GLUT_UP)
		{
			cout << "exit" << endl;
			exit(0);
		}
	}
	else
	{
		if (m_state == GLUT_UP)
			cout << "full = " << full << endl;
			cout << m_x << " " << m_y << endl;
		if (m_y > 154 && m_y < 195 && m_state == GLUT_UP)
		{
			cout << "play" << endl;
			page = 31;
		}
		if (m_y > 213 && m_y < 251 && m_state == GLUT_UP)
		{
			cout << "Settings" << endl;
			page = 31;
			setting = 1;
		}
		if (m_y > 269 && m_y < 310 && m_state == GLUT_UP)
		{
```

```
                        cout << "Instructions" << endl;
                        page = 21;
                }
                if (m_y > 329 && m_y < 366 && m_state == GLUT_UP)
                {
                        cout << "Credit" << endl;
                        page = 22;
                }
                if (m_y > 387 && m_y < 425 && m_state == GLUT_UP)
                {
                        cout << "High" << endl;
                        page = 23;
                }
                if (m_y > 430 && m_state == GLUT_UP)
                {
                        cout << "exit" << endl;
                        exit(0);
                }

        }

}
else if (page == 31)
{
        if (button == GLUT_LEFT_BUTTON && m_state == GLUT_UP)
        {
                if (full == 0)
                {
                        cout << m_x << " " << m_y << endl;
                        if (m_y > 168 && m_y < 205)
                        {
                                plane_choice = 1;
                                i_plane = 0;
                                glutPostRedisplay();
                        }
                        if (m_y > 242 && m_y < 287)
                        {
                                plane_choice = 2;
                                i_plane = 0;
                                glutPostRedisplay();
                        }
                        if (m_y > 300)
                        {
                                cout << "next" << endl;
                                i_plane = 0;
                                page = 32;
                        }
                }
                else
                {
```

```
                        if (m_y > 307 && m_y < 347)
                        {
                                plane_choice = 1;
                                i_plane = 0;
                                glutPostRedisplay();
                        }
                        if (m_y > 416 && m_y < 463)
                        {
                                plane_choice = 2;
                                i_plane = 0;
                                glutPostRedisplay();
                        }
                        if (m_y > 470)
                        {
                                cout << "next" << endl;
                                i_plane = 0;
                                page = 32;
                        }
                }
        }
}
else if (page == 32)
{
        if (button == GLUT_LEFT_BUTTON && m_state == GLUT_UP)
        {
                if (full == 0)
                {
                        if (m_y > 300)
                        {
                                cout << "next" << endl;
                                i_plane = 0;
                                if (setting == 1)
                                {
                                        setting = 0;
                                        page = 2;
                                }
                                else
                                        page = 3;
                        }
                }
                else
                {
                        if (m_y > 470)
                        {
                                cout << "next" << endl;
                                i_plane = 0;
                                if (setting == 1)
                                {
                                        setting = 0;
                                        page = 2;
```

```
                                        }
                                        else
                                                page = 3;
                                }
                        }
                }
        }
        else if (page == 3)
        {
                if (button == GLUT_LEFT_BUTTON && m_state == GLUT_DOWN)
                {
                        state = UP;
                        cout << "Going Up!" << endl;
                }
                else if (button == GLUT_LEFT_BUTTON && m_state == GLUT_UP)
                {
                        state = DOWN;
                        cout << "Going Down" << endl;
                }
        }
}
void draw_rockets()
{
        if (missile_x > 200)
        {
                no_of_missiles = rand() % MAX_MISSILES + 1;
        }
        if (missile_x >= 195 && missile_x <= 200)
        {
                for (int k = 1; k <= no_of_missiles; k++)
                        missile_y[k] = -101 + rand() % 165;
        }
        switch (no_of_missiles)
        {
        case 1:
                rocket1(missile_x, missile_y[1]);
                break;
        case 2:
                rocket1(missile_x, missile_y[1]);
                rocket2(missile_x, missile_y[2]);
                break;
        case 3:
                rocket1(missile_x, missile_y[1]);
                rocket2(missile_x, missile_y[2]);
                rocket3(missile_x, missile_y[3]);
                break;
        case 4:
                rocket1(missile_x, missile_y[1]);
                rocket3(missile_x, missile_y[2]);
                rocket2(missile_x, missile_y[3]);
```

```
			rocket3(missile_x, missile_y[4]);
			break;
		default:
		case 5:
			rocket1(missile_x, missile_y[1]);
			rocket3(missile_x, missile_y[2]);
			rocket2(missile_x, missile_y[3]);
			rocket3(missile_x, missile_y[4]);
			rocket1(missile_x, missile_y[5]);
			break;

		}
}
void RenderScene()
{
		if (page == 0)
		{
			glClear(GL_COLOR_BUFFER_BIT);

			//load .png image
			glEnable(GL_TEXTURE_2D);
			glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
			if (i_0 == 0)
			{
				tex_2d_0 = SOIL_load_OGL_texture
				(
					"res/loading.png",
					SOIL_LOAD_RGBA,
					SOIL_CREATE_NEW_ID,
					SOIL_FLAG_NTSC_SAFE_RGB
				);
				i_0 = 1;
			}
			glBindTexture(GL_TEXTURE_2D, tex_2d_0);
			glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);


			glBegin(GL_POLYGON);
			glTexCoord2f(0.0, 1.0);
			glVertex2f(-190.0f, -190.0f);
			glTexCoord2f(1.0, 1.0);
			glVertex2f(190.0f, -190.0f);
			glTexCoord2f(1.0, 0.0);
			glVertex2f(190.0f, 190.0f);
			glTexCoord2f(0.0, 0.0);
			glVertex2f(-190.0f, 190.0f);
			glEnd();
			glDisable(GL_TEXTURE_2D);
```

```
		//draw loading bar
		glColor3f(1, 1, 1);
		glBegin(GL_POLYGON);
		glVertex2f(-171.0f, -4.4f);
		glVertex2f(0.0f + x_step, -4.4f);
		glVertex2f(0.0f + x_step, 1.3f);
		glVertex2f(-171.0f, 1.3f);
		glEnd();
		drawfps();
		glutSwapBuffers();
		glFlush();
	}
	if (page == 1)
	{
		glClear(GL_COLOR_BUFFER_BIT);

		//load .png image
		glEnable(GL_TEXTURE_2D);
		glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
		if (i_1 == 0)
		{
			tex_2d_1 = SOIL_load_OGL_texture
			(
				"res/paper.jpg",
				SOIL_LOAD_RGBA,
				SOIL_CREATE_NEW_ID,
				SOIL_FLAG_NTSC_SAFE_RGB
			);
			i_1 = 1;
		}
		glBindTexture(GL_TEXTURE_2D, tex_2d_1);
		glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);


		glBegin(GL_POLYGON);
		glTexCoord2f(0.0, 1.0);
		glVertex2f(-178.0f, -100.0f);
		glTexCoord2f(1.0, 1.0);
		glVertex2f(178.0f, -100.0f);
		glTexCoord2f(1.0, 0.0);
		glVertex2f(178.0f, 100.0f);
		glTexCoord2f(0.0, 0.0);
		glVertex2f(-178.0f, 100.0f);
		glEnd();
		glDisable(GL_TEXTURE_2D);

		setFont(GLUT_BITMAP_TIMES_ROMAN_24);
		glColor3f(0.0, 0.0, 0.0);
		if (!full)
```

```
		{
			drawString(-129.0, 65.0, 0.0, "Bangalore Institute of Technology");
			setFont(GLUT_BITMAP_HELVETICA_18);
			glColor3f(0.0, 0.0, 0.0);
			drawString(-90.0, 55.0, 0.0, "Made By:");
			setFont(GLUT_BITMAP_HELVETICA_18);
			glColor3f(0.0, 0.0, 0.0);
			drawString(-110.0, 45.0, 0.0, "Revanth P N - 1BI17CS123");
			setFont(GLUT_BITMAP_HELVETICA_18);
			glColor3f(0.0, 0.0, 0.0);
			drawString(-105.0, 35.0, 0.0, "Prajwal P - 1BI17CS111");
			setFont(GLUT_BITMAP_HELVETICA_18);
			glColor3f(0.0, 0.0, 0.0);
			drawString(130.0, -50.0, 0.0, "Next!");
		}
		else
		{
			drawString(-105.0, 65.0, 0.0, "Bangalore Institute of Technology");
			setFont(GLUT_BITMAP_HELVETICA_18);
			glColor3f(0.0, 0.0, 0.0);
			drawString(-80.0, 55.0, 0.0, "Made By:");
			setFont(GLUT_BITMAP_HELVETICA_18);
			glColor3f(0.0, 0.0, 0.0);
			drawString(-95.0, 45.0, 0.0, "Revanth P N - 1BI17CS123");
			setFont(GLUT_BITMAP_HELVETICA_18);
			glColor3f(0.0, 0.0, 0.0);
			drawString(-92.0, 35.0, 0.0, "Prajwal P - 1BI17CS111");
			setFont(GLUT_BITMAP_TIMES_ROMAN_24);
			glColor3f(0.0, 0.0, 0.0);
			drawString(125.0, -50.0, 0.0, "Click Here!");
			setFont(GLUT_BITMAP_HELVETICA_18);
			glColor3f(0.0, 0.0, 0.0);
			drawString(135.0, -55.0, 0.0, "Next!");
		}

		drawfps();
		glutSwapBuffers();
	}
	if (page == 2)
	{

		glClear(GL_COLOR_BUFFER_BIT);

		//load .jpg image
		glEnable(GL_TEXTURE_2D);
		glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
		if (i_2 == 0)
		{
			tex_2d_2 = SOIL_load_OGL_texture
			(
```

```
                        "res/plane_menu.jpg",
                        SOIL_LOAD_RGBA,
                        SOIL_CREATE_NEW_ID,
                        SOIL_FLAG_NTSC_SAFE_RGB
                );
                i_2 = 1;
        }
        glBindTexture(GL_TEXTURE_2D, tex_2d_2);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);


        glBegin(GL_POLYGON);
        glTexCoord2f(0.0, 1.0);
        glVertex2f(-178.0f, -100.0f);
        glTexCoord2f(1.0, 1.0);
        glVertex2f(178.0f, -100.0f);
        glTexCoord2f(1.0, 0.0);
        glVertex2f(178.0f, 100.0f);
        glTexCoord2f(0.0, 0.0);
        glVertex2f(-178.0f, 100.0f);
        glEnd();
        glDisable(GL_TEXTURE_2D);

        draw_menu_text();
        drawfps();
        glutSwapBuffers();
    }
    if (page == 21)
    {
        glClear(GL_COLOR_BUFFER_BIT);

        glEnable(GL_TEXTURE_2D);

        glColor4f(1.0f, 1.0f, 1.0f, 1.0f);

        if (i_inst21 == 0)
        {
            tex_2d_21 = SOIL_load_OGL_texture
            (
                "res/instructions2.png",
                SOIL_LOAD_RGBA,
                SOIL_CREATE_NEW_ID,
                SOIL_FLAG_NTSC_SAFE_RGB
            );
            i_inst21 = 1;
        }

        glBindTexture(GL_TEXTURE_2D, tex_2d_21);
```

```
		glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);

		glBegin(GL_POLYGON);
		glTexCoord2f(0.0, 1.0);
		glVertex2f(-178.0f, -100.0f);
		glTexCoord2f(1.0, 1.0);
		glVertex2f(178.0f, -100.0f);
		glTexCoord2f(1.0, 0.0);
		glVertex2f(178.0f, 100.0f);
		glTexCoord2f(0.0, 0.0);
		glVertex2f(-178.0f, 100.0f);
		glEnd();
		glDisable(GL_TEXTURE_2D);

		draw_inst_text();
		glutSwapBuffers();
	}
	if (page == 22)
	{
		glClear(GL_COLOR_BUFFER_BIT);

		//load .png image
		glEnable(GL_TEXTURE_2D);
		glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
		if (i_cre22 == 0)
		{
			tex_2d_22 = SOIL_load_OGL_texture
			(
				"res/cre.jpg",
				SOIL_LOAD_RGBA,
				SOIL_CREATE_NEW_ID,
				SOIL_FLAG_NTSC_SAFE_RGB
			);
			i_cre22 = 1;
		}
		glBindTexture(GL_TEXTURE_2D, tex_2d_22);
		glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);


		glBegin(GL_POLYGON);
		glTexCoord2f(0.0, 1.0);
		glVertex2f(-178.0f, -100.0f);
		glTexCoord2f(1.0, 1.0);
		glVertex2f(178.0f, -100.0f);
		glTexCoord2f(1.0, 0.0);
		glVertex2f(178.0f, 100.0f);
		glTexCoord2f(0.0, 0.0);
		glVertex2f(-178.0f, 100.0f);
```

```
		glEnd();
		glDisable(GL_TEXTURE_2D);
		drawLogo();
		draw_credit_text();
		glutSwapBuffers();
	}
	if (page == 23)
	{
		glClear(GL_COLOR_BUFFER_BIT);

		//load .jpg image
		glEnable(GL_TEXTURE_2D);
		glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
		if (i_23 == 0)
		{
			tex_2d_23 = SOIL_load_OGL_texture
			(
				"res/scene2.png",
				SOIL_LOAD_RGBA,
				SOIL_CREATE_NEW_ID,
				SOIL_FLAG_NTSC_SAFE_RGB
			);
			i_23 = 1;
		}
		glBindTexture(GL_TEXTURE_2D, tex_2d_23);
		glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);


		glBegin(GL_POLYGON);
		glTexCoord2f(0.0, 1.0);
		glVertex2f(-178.0f, -100.0f);
		glTexCoord2f(1.0, 1.0);
		glVertex2f(178.0f, -100.0f);
		glTexCoord2f(1.0, 0.0);
		glVertex2f(178.0f, 100.0f);
		glTexCoord2f(0.0, 0.0);
		glVertex2f(-178.0f, 100.0f);
		glEnd();
		glDisable(GL_TEXTURE_2D);

		draw_high_text();
		glutSwapBuffers();
	}
	if (page == 31)
	{
		glClear(GL_COLOR_BUFFER_BIT);


		glEnable(GL_TEXTURE_2D);
```

```
                glColor4f(1.0f, 1.0f, 1.0f, 1.0f);

                if (i_sel31 == 0)
                {
                        tex_2d_31 = SOIL_load_OGL_texture
                        (
                                "res/bck_plane.jpg",
                                SOIL_LOAD_RGBA,
                                SOIL_CREATE_NEW_ID,
                                SOIL_FLAG_NTSC_SAFE_RGB
                        );
                        i_sel31 = 1;
                }

                glBindTexture(GL_TEXTURE_2D, tex_2d_31);

                glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);

                glBegin(GL_POLYGON);
                glTexCoord2f(0.0, 1.0);
                glVertex2f(-178.0f, -100.0f);
                glTexCoord2f(1.0, 1.0);
                glVertex2f(178.0f, -100.0f);
                glTexCoord2f(1.0, 0.0);
                glVertex2f(178.0f, 100.0f);
                glTexCoord2f(0.0, 0.0);
                glVertex2f(-178.0f, 100.0f);
                glEnd();
                glDisable(GL_TEXTURE_2D);

                draw_chPlane_text();
                draw_chosen_plane();

                glutSwapBuffers();
        }
        if (page == 32)
        {
                glClear(GL_COLOR_BUFFER_BIT);

                glEnable(GL_TEXTURE_2D);

                glColor4f(1.0f, 1.0f, 1.0f, 1.0f);

                select_scene();
                if (i_s == 0)
                {
                        tex_2d = SOIL_load_OGL_texture
                        (
```

```
                        scene,
                        SOIL_LOAD_RGBA,
                        SOIL_CREATE_NEW_ID,
                        SOIL_FLAG_NTSC_SAFE_RGB
                );
                i_s = 1;
        }

        glBindTexture(GL_TEXTURE_2D, tex_2d);

        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);

        glBegin(GL_POLYGON);
        glTexCoord2f(0.0, 1.0);
        glVertex2f(-178.0f, -100.0f);
        glTexCoord2f(1.0, 1.0);
        glVertex2f(178.0f, -100.0f);
        glTexCoord2f(1.0, 0.0);
        glVertex2f(178.0f, 100.0f);
        glTexCoord2f(0.0, 0.0);
        glVertex2f(-178.0f, 100.0f);
        glEnd();
        glDisable(GL_TEXTURE_2D);

        draw_chScene_text();

        glutSwapBuffers();
    }
    if (page == 3)
    {
        glClear(GL_COLOR_BUFFER_BIT);

        glEnable(GL_TEXTURE_2D);
        glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
        if (i_bck == 0)
        {
                tex_2d = SOIL_load_OGL_texture
                (
                        scene,
                        SOIL_LOAD_RGBA,
                        SOIL_CREATE_NEW_ID,
                        SOIL_FLAG_NTSC_SAFE_RGB
                );
                i_bck = 1;
        }
        glBindTexture(GL_TEXTURE_2D, tex_2d);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);
        glBegin(GL_POLYGON);
```

```
            glTexCoord2f(0.0, 1.0);
            glVertex2f(-190.0f + x, -100.0f);
            glTexCoord2f(1.0, 1.0);
            glVertex2f(890.0f + x, -100.0f);
            glTexCoord2f(1.0, 0.0);
            glVertex2f(890.0f + x, 100.0f);
            glTexCoord2f(0.0, 0.0);
            glVertex2f(-190.0f + x, 100.0f);
            glEnd();
            glDisable(GL_TEXTURE_2D);

            //fuel indicator outline
            glColor3f(0, 0, 0);
            glLineWidth(3);
            glBegin(GL_LINE_LOOP);
            glVertex2f(-50 - 100, 80);
            glVertex2f(50 - 100, 80);
            glVertex2f(50 - 100, 70);
            glVertex2f(-50 - 100, 70);
            glEnd();

            //fuel indicator
            glColor3f(1, 1, 1);
            glBegin(GL_POLYGON);
            glVertex2f(-49 - 100, 79);
            glVertex2f(-49 + fuel - 100, 79);
            glVertex2f(-49 + fuel - 100, 71);
            glVertex2f(-49 - 100, 71);
            glEnd();

            //seprator--to seprate score and game screen
            glLineWidth(1);
            glColor3f(1, 1, 1);
            glBegin(GL_LINES);
            glVertex2f(-200, 64);
            glVertex2f(200, 64);
            glEnd();

            draw_score();

            draw_rockets();

            glPushMatrix();
            glTranslatef(-130, y_pos, 0);
            glRotatef(theta, 0, 0, 1);
            plane1.draw_plane();
            glPopMatrix();
            drawfps();
            glutSwapBuffers();
    }
```

```
	if (page == 4)
	{
		glClear(GL_COLOR_BUFFER_BIT);

		//load .png image
		glEnable(GL_TEXTURE_2D);
		glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
		if (i_fin4 == 0)
		{
			tex_2d_4 = SOIL_load_OGL_texture
			(
				"res/finish.png",
				SOIL_LOAD_RGBA,
				SOIL_CREATE_NEW_ID,
				SOIL_FLAG_NTSC_SAFE_RGB
			);
			i_fin4 = 1;
		}
		glBindTexture(GL_TEXTURE_2D, tex_2d_4);
		glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);


		glBegin(GL_POLYGON);
		glTexCoord2f(0.0, 1.0);
		glVertex2f(-178.0f, -100.0f);
		glTexCoord2f(1.0, 1.0);
		glVertex2f(178.0f, -100.0f);
		glTexCoord2f(1.0, 0.0);
		glVertex2f(178.0f, 100.0f);
		glTexCoord2f(0.0, 0.0);
		glVertex2f(-178.0f, 100.0f);
		glEnd();
		glDisable(GL_TEXTURE_2D);
		//drawText();
		draw_fin_text();
		glutSwapBuffers();
	}

}
```

Chapter-5

SNAPSHOTS

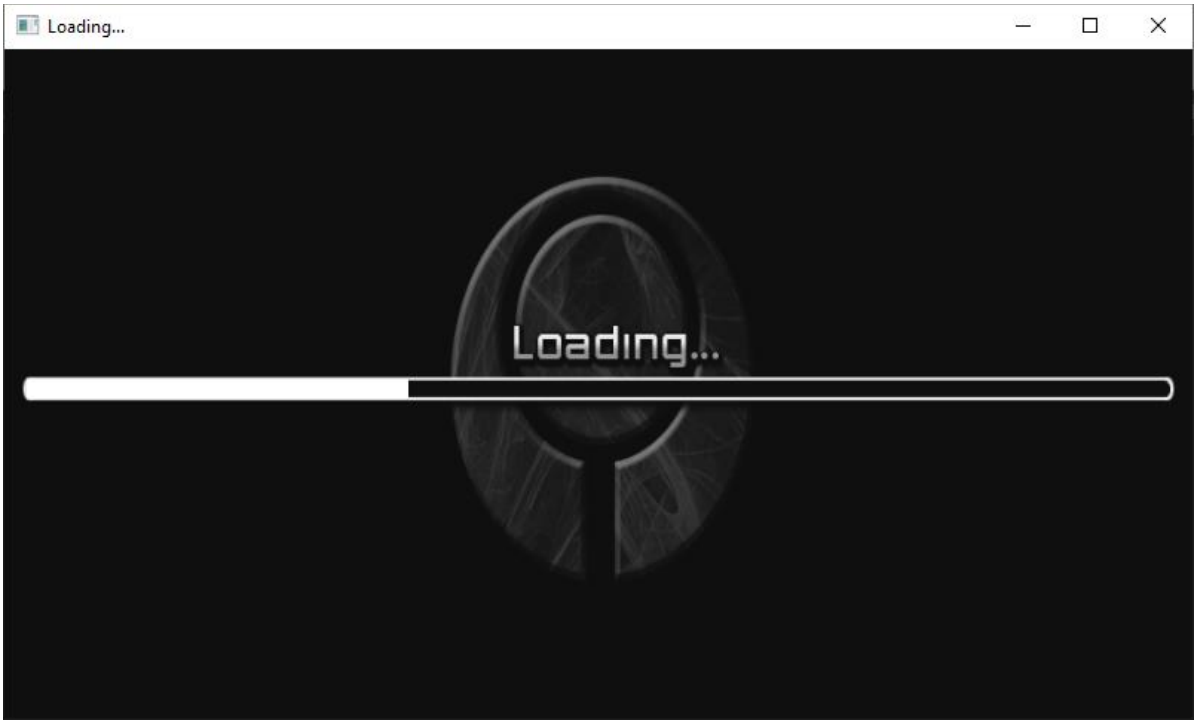

Fig 5.1: Loading Screen

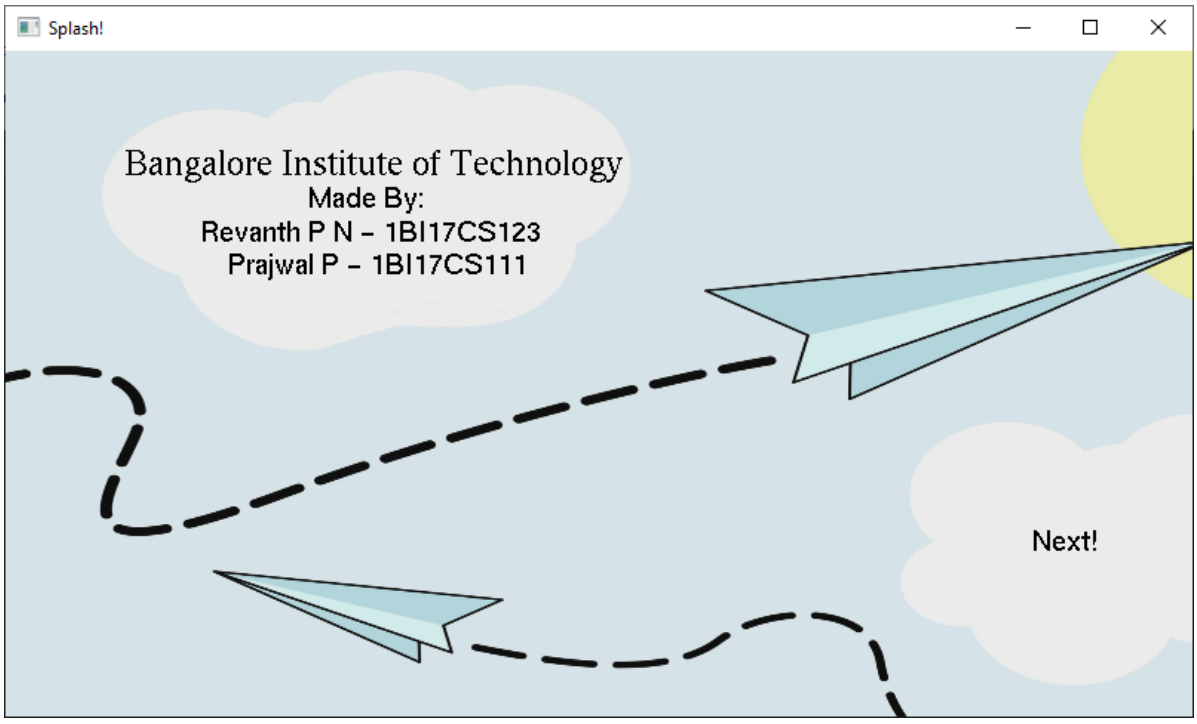

Fig 5.2: Splash Screen

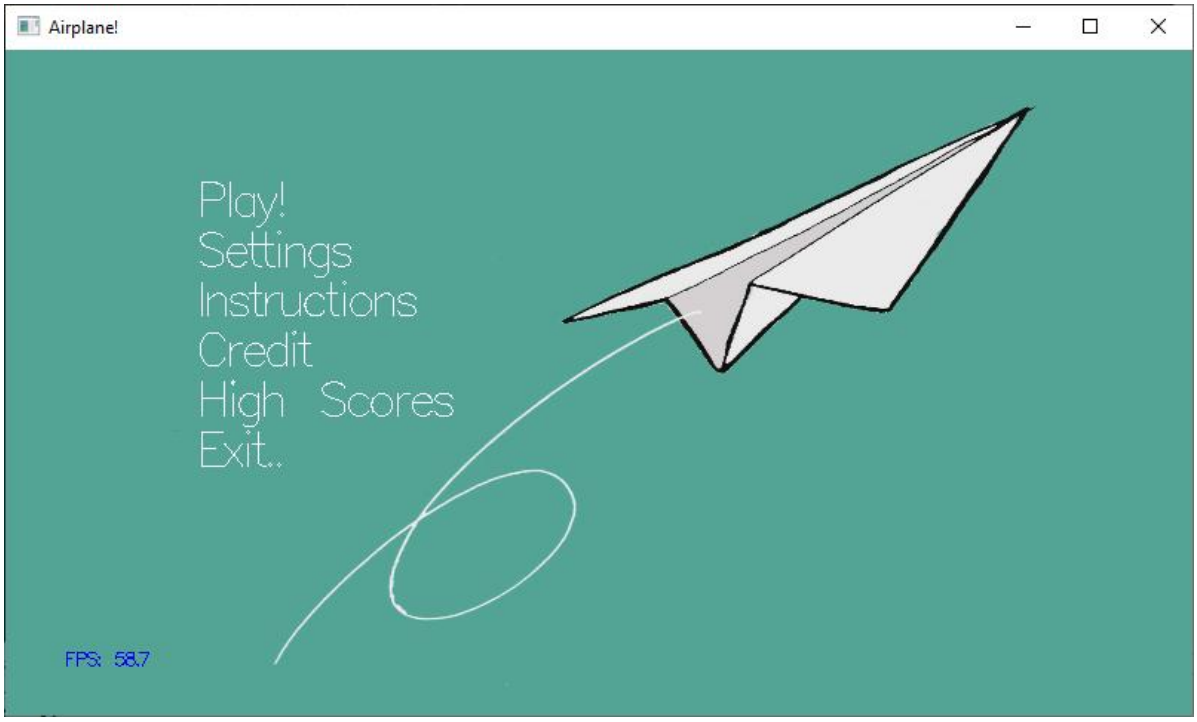

Fig 5.3: Main Menu

Fig 5.4: Choosing Plane 1

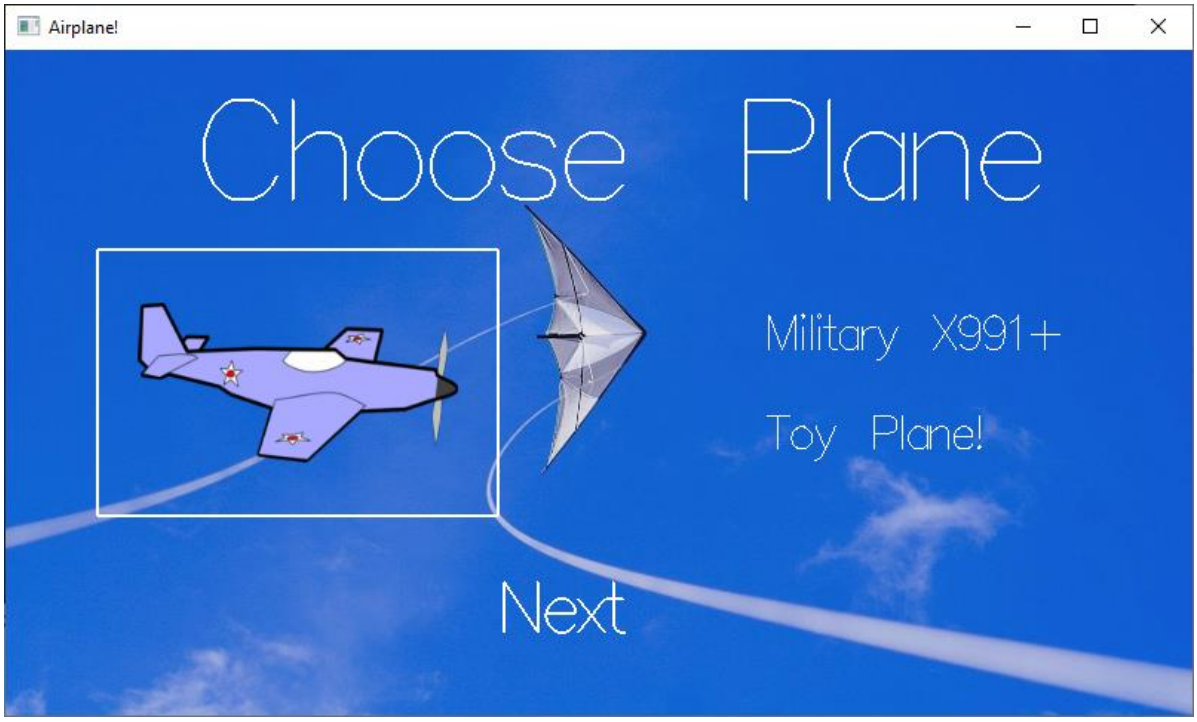

Fig 5.5: Choosing Plane 2

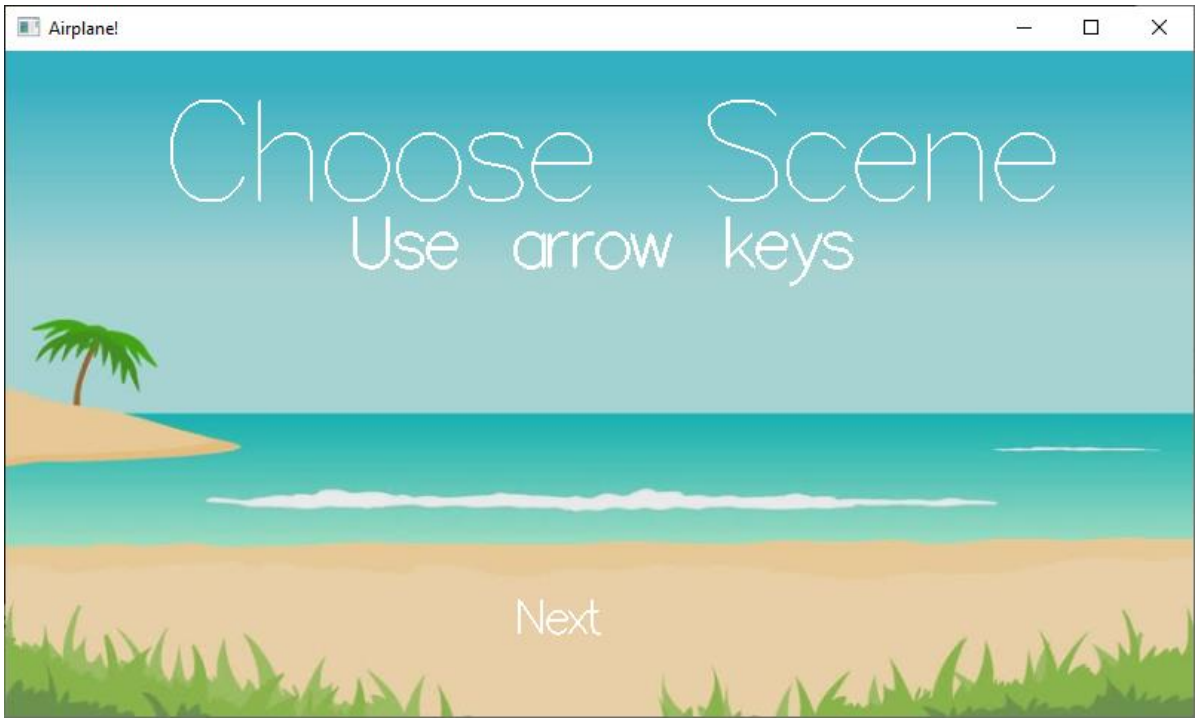

Fig 5.6: Choosing scenery

Fig 5.7: Choosing another scenery

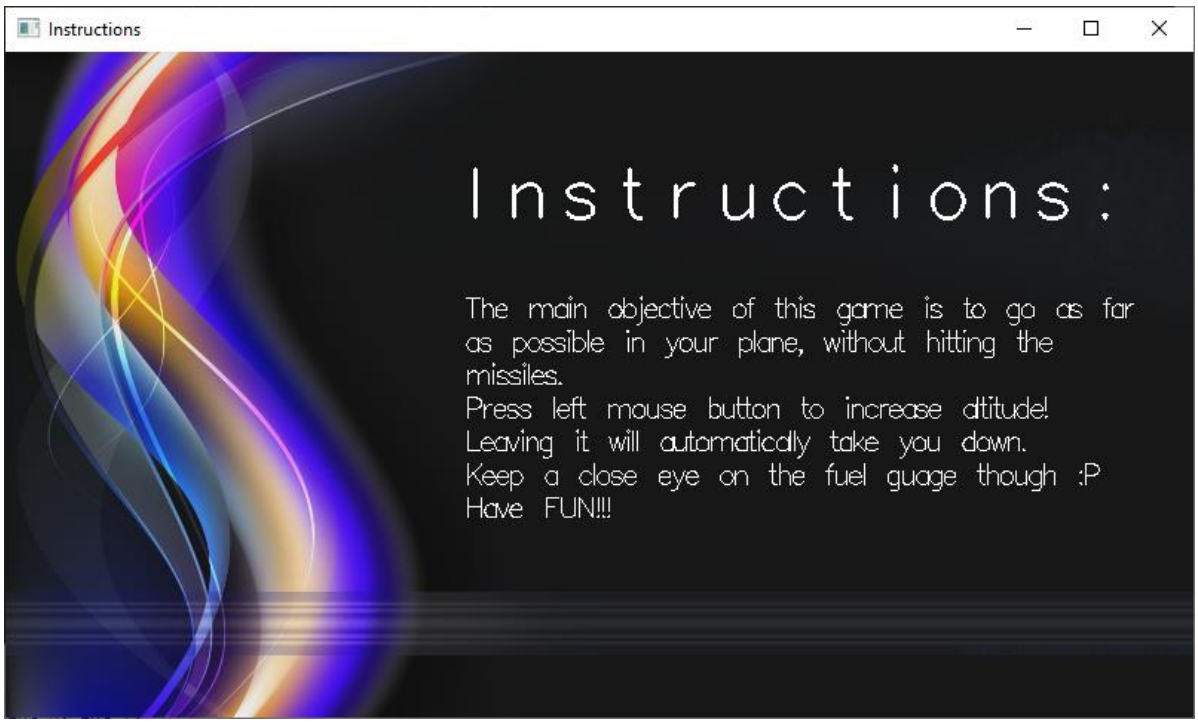

Fig 5.8: Instruction page

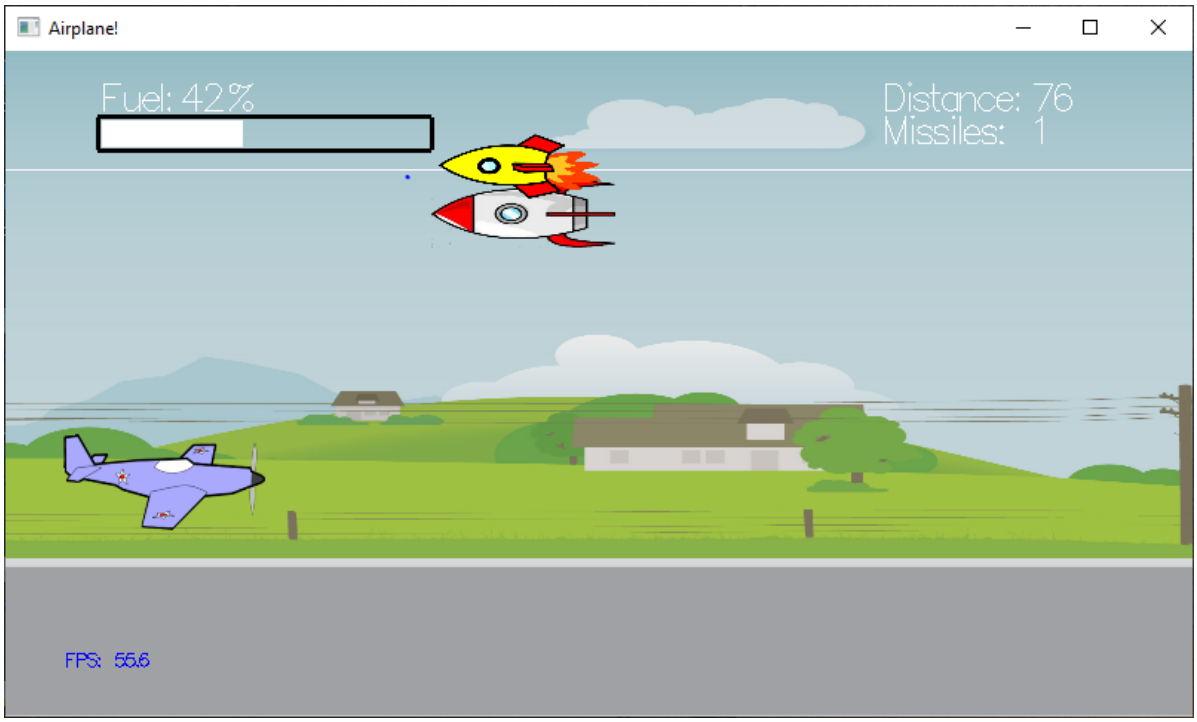

Fig 5.9: Actual game play

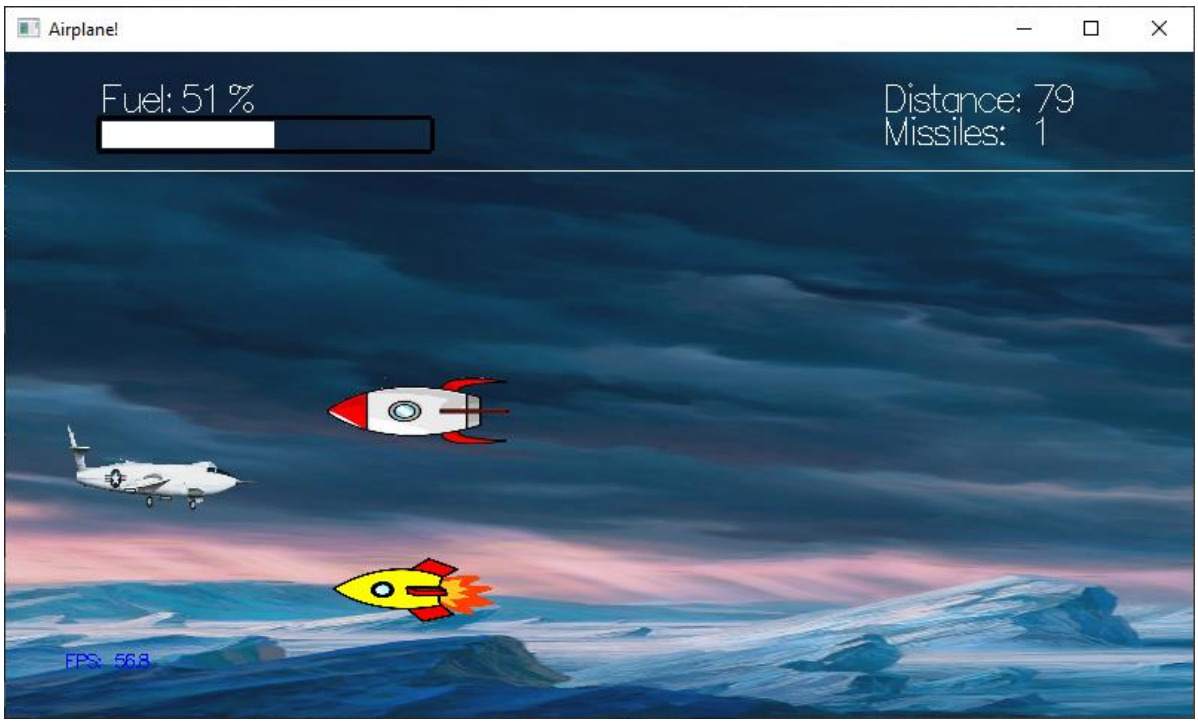

Fig 5.10: Actual game play with different scenery and plane

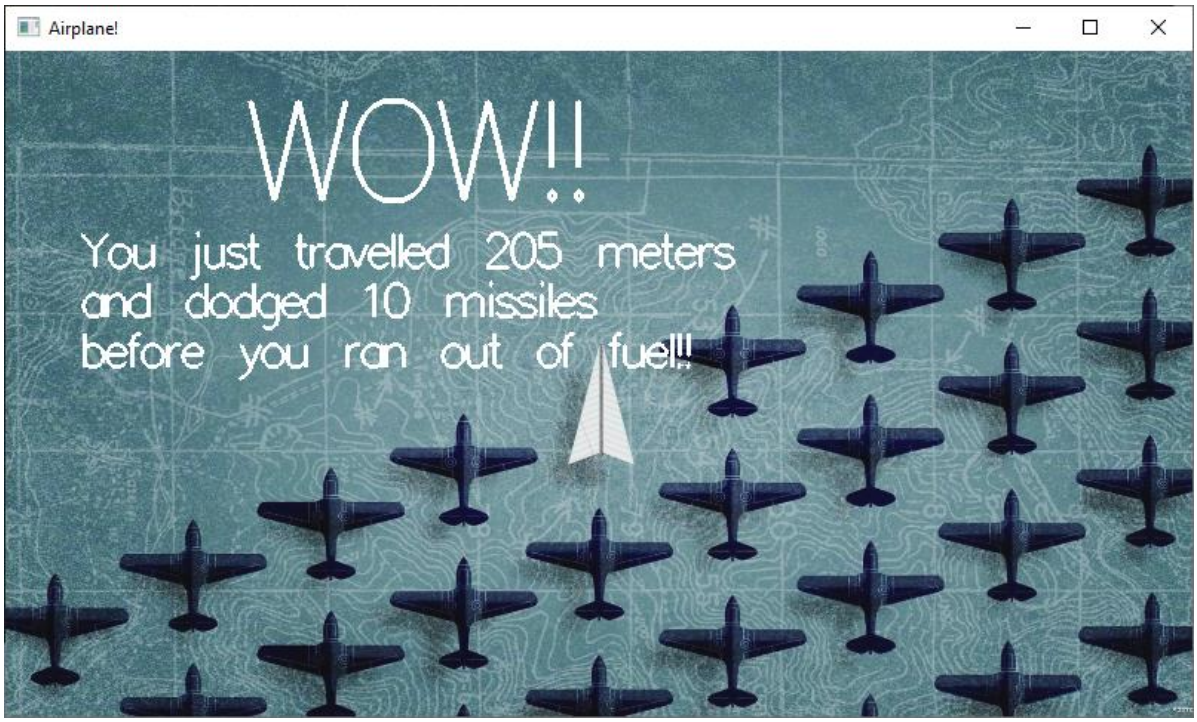

Fig 5.11: High Score page

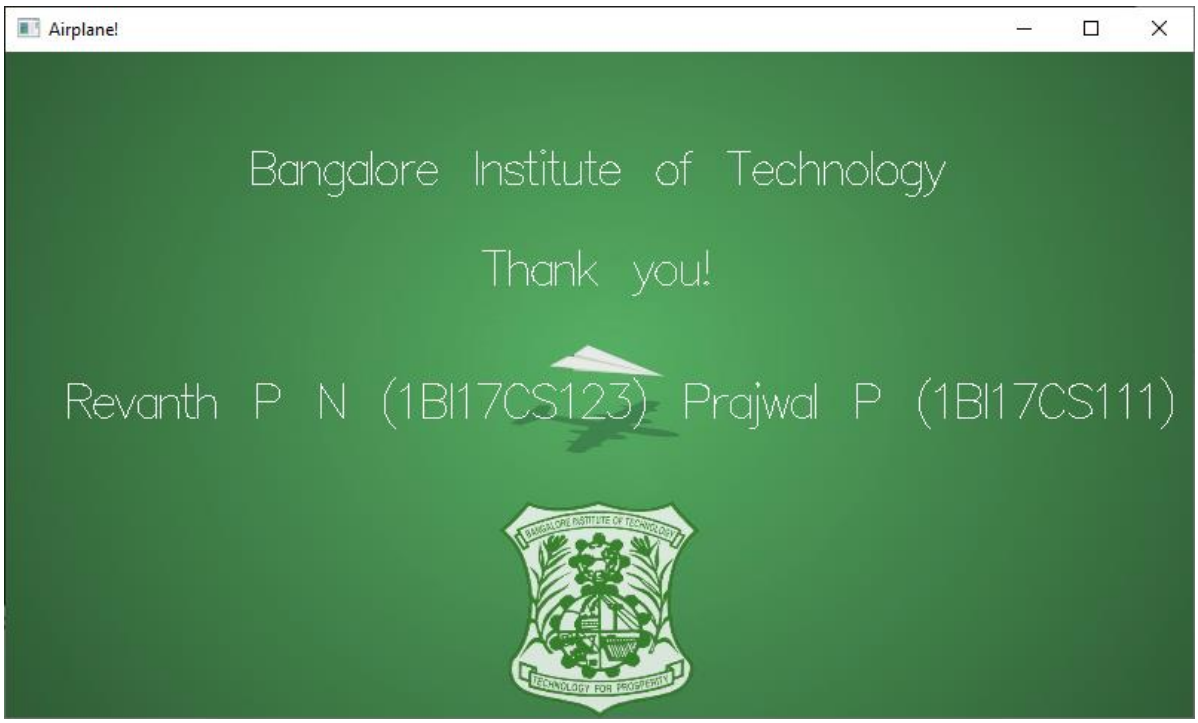

Fig 5.12: Credits page

Chapter-6

CONCLUSION

The purpose of the project is to learn OpenGL APIs and transforming objects in OpenGL window. It has enabled us to have a deeper understanding of how computer graphics is implementing.

Thus, in this project we have acquired a lot of knowledge about various techniques in OpenGL programming. We have explored many new concepts on the World Wide Web, such as texture mapping, etc.

We thus would like to emphasize the importance of this project to many other perspectives of Technical, mathematical, graphical and software concepts which we were unaware of.

FUTURE ENHANCEMENT

1. In further the same project can be enhanced by adding new levels in the game.
2. We can add audio while playing the game.
3. We could also include more planes and scenes.
4. We could also add friends and compare scores with them.

BIBLIOGRAPHY

[1] Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version, 3rd/4th Edition, Pearson Education, 2011

[2] Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL, 5th Edition, Pearson Education

[3] James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer Graphics with OpenGL, Pearson Education

[4] Macro Cantu: Mastering Delphi

Websites

[1] https://www.opengl.org/

[2] http://stackoverflow.com/questions/ opengl

[3] https://geeksforgeeks.org/