CHAPTER 1

INTRODUCTION

1.1 About Computer Graphics:

The term computer graphics has been used in a broad sense to describe almost everything on computers that is not text or sound. 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.

Today, computer graphics is widespread. Such imagery is found in and on television, newspapers, weather reports, and in a variety of medical investigations and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media such graphs are used to illustrate papers, reports and other presentation material.

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.

Using this editor you can draw and paint using the mouse. It can also perform a host of other functions like drawing lines, circles, and polygons and so on. Interactive picture construction techniques such as basic positioning methods, rubber-band methods, dragging and drawing are used. Block operations like cut, copy and paste are supported to edit large areas of the workspace simultaneously. It is user friendly and intuitive to use.

Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D Or 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly. Computers have become a powerful medium for the rapid and economical production of pictures. Graphics provide a so natural means of communicating with the computer that they have become widespread. Interactive graphics is the most important means of producing pictures since the invention of photography and television. We can make pictures of not only the real world objects but also of abstract objects such as mathematical surfaces on 4D and of data that have no inherent geometry. A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer and output devices.

There are different types of Graphics architectures which are:

1. **Display processors**: The earliest attempts to build special purpose graphics system were concerned primarily with relieving the general purpose computer from the task of refreshing the display continuously. These display processors had conventional architecture but included instructions to display primitives on the CRT.

2. **Pipeline architectures**: The availability of inexpensive solid state memory led to the universality of raster display. For computer graphics applications, the most important use of custom VLSI circuits has been in creating pipeline architectures.

3. **Graphics pipeline**: Each object comprises a set of graphical primitives. Each primitive comprises set of vertices.

4. **Vertex processing**: The assignment of vertex color scan as simple as program specifying a color or as complex as computation of a color from a physically realistic lightning model that incorporates the surface properties of the object and the characteristic light sources in the scene.

5. **Clipping and primitive assembly**: We must do clipping because of the limitations of that no imaging system can see the whole world at once. The human retina has a limited size corresponding to an approximately 90-degreed field of view.

1.2 History of Computer Graphics:

William fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeing. One of the first displays of computer animation was future world (1976), which included an animation of a human face and hand-produced by Carmel and Fred Parle at the university of Utah.

There are several international conferences and journals where the most significant results in computer graphics are published, among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on graphics journals.

1.3 About OpenGL:

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. The interface consists of over 250 different function calls which can be used to draw complex 3D scenes from simple primitives. 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, where it competes with direct 3D on Microsoft Windows platforms. OpenGL is managed by the non-profit technology consortium, the khronos group, Inc.

OpenGL serves two main purposes:

- 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.

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms.

OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

OpenGL Available Everywhere: Supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and Mac OS PC, no other graphics API operates on a wider range of hardware platforms and software environments.

OpenGL runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, Open Step, and BeOS; it also works with every major windowing system, including Win32, Mac OS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, FORTRAN, Python, Perl and Java and offers complete independence from network protocols and topologies.

The OpenGL interface: Our application will be designed to access OpenGL directly through functions in three libraries namely-

- GL
- GLU
- glut

1.4 Applications of Computer Graphics:

- **Video games:**
  These games requires human interaction with the User Interface for making nice visual result on the Video Device

- **Education Field:**
  In the learning process, acquiring knowledge and skills required for better career path CG is needed. So Computer generated models in economic, financial and physical systems are often used.
  We regularly love to have computerized model to understand any topic easily. Equipment, Physiological systems, physical systems are coded using CG.

- **Computer Aided Design (CAD):**
  Most of engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircrafts, Building, in almost all kinds of Industries (where designing is necessary)
  After making a full diagram, we can even see its animation (Operation and working of a Product)

- **Computer Arts:**
  If we are intelligent enough we can rock by making creative arts using these Graphics tools. For making these arts we generally use CAD packages, paint and Paint brush programs and in animation too
  Computer Arts Examples include Logo design (for companies, college, Industries, Institutions), Cartoon drawing, Product advertisements and many.

- **Simulation:**
  Using CG graphics reproduction or duplicating already existing thing will be done. For Instance, if we go for the Flight simulators, these computer generated images are very much needed for training pilots to understand easily (learning standard methods)

- **Entertainment:**
  When we talk about Entertainment, immediately movies and games get in to picture.CG are mostly used in music videos, motion pictures, cartoon animation films. For finding out tricks to be used in Games, for its interactivity we often use CG

- **Image Processing: (Medical)**
  In Medical field, concerning Image Processing CG is used to various technologies to interpret already existing pictures and is useful to modify Photographs and TV scans. Basically CG is used to improve the picture quality, and visualizing effects. Some of its applications includes ,Tomography ,.Ultrasonic medical scanners ,Picture enhancements.

- **User Interfaces (Graphical User Interfaces):**
  CG is effectively used to make Menus, Icons (Graphical Symbols),to make window manager(multiple windows).And Some of the Graphic packages includes PHIGS, Graphics Kernel System, Painting and drawing.

Some of the other applications are:-

- Scientific and Business Visualization
- Graphic Presentation
- Graphic Design
- Web Designing

- Computational Physics
- Information Visualization
- Display of Information
- Information Architecture
- Genetics, Molecular Biology, Neuroscience, Animation, Statistics
- Desktop Publishing.

1.5 Built-In Functions:

- **glColor3f (float, float, float) :-**
  This function will set the current drawing color.
- **gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top):-**
  Which defines a two dimensional viewing rectangle in the plane z=0.
- **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 ( ):-**
  Sets the current matrix to an identity matrix.
- **glMatrixMode (mode):-**
  Sets the current matrix mode, mode can be GL_MODELVIEW, GL_PROJECTION or GL_TEXTURE.
- **glutInit (int *argc, char**argv):-**
  Initializes GLUT, the arguments from main are passed in and can be used by the application.
- **glutInitDisplayMode (unsigned int mode):-**
  Request 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.
- **glutInitWindowSize (int width, int height):-**
  Specifies the initial position of the top-left corner of the window in pixels
- **glutCreateWindow (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.
- **glutMouseFunc(void *f(int button, int state, int x, int y):-**
  Register the mouse callback function f. The callback function returns the button, the state of button after the event and the position of the mouse relative to the top-left corner of the window.

- **Void glutKeyboardFunc(void(*func) (void)):-**
  This function is called every time when you press enter key to resume the game or when you press „b" or „B" key to go back to the initial screen or when you press esc key to exit from the application.

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

- **glutSpecialFunc(void(*func)(void)):-**
  This function is called when you press the special keys in the keyboard like arrow keys, function keys etc. In our program, the function is invoked when the up arrow or down arrow key is pressed for selecting the options in the main menu and when the left or right arrow key is pressed for moving the object(car) accordingly.
- **glut PostReDisplay ( ) :-**
  Which requests that the display callback be executed after the current callback returns.

- **Void MouseFunc (void (*func) void)):-**
  This function is invoked when mouse keys are pressed. This function is used as an alternative to the previous function i.e., it is used to move the object(car) to right or left in our program by clicking left and right button respectively.

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

1.6 Keys used:

1. Key Left-arrow : Moves the Aircraft in the left direction.

2. Key Right-arrow : Moves the Aircraft in the right direction.

3. Key Up-arrow : To shoot the laser-bullets from the Aircraft.

4. Key ESC: To display the game options / pauses the game.

Special Functions Used:

The functions performed by the keyboard are with a wide effect-

1. Key F2: To display OpenGL information.

2. Key 1: To calibrate the frames to 15 FPS.

3. Key 2: To calibrate the frames to 30 FPS.

4. Key 3: To calibrate the frames to 60 FPS.

5. Key 4: To calibrate the frames to maximum FPS

CHAPTER 2

LITERATURE SURVEY

- Computer graphics started with the display of data on hardcopy plotters and cathode ray tube (CRT) screens soon after the introduction of computers.

- Computer graphics today largely interactive, the user controls the contents, structure, and appearance of objects and of displayed images by using input devices, such as keyboard, mouse, or touch-sensitive panel on the screen. Graphics based user interfaces allow millions of new users to control simple, low-cost application programs, such as spreadsheets, word processors, and drawing programs.

- 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. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. 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, where it competes with Direct3D on Microsoft Windows platforms (see Direct3D vs. OpenGL). OpenGL is managed by the non-profit technology consortium, the Khronos Group.

- In the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. By the early 1990s, Silicon Graphics (SGI) was a leader in 3D graphics for workstations. SGI's competitors (including Sun Microsystems, Hewlett-Packard and IBM) were also able. In addition, SGI had a large number of software customers; by changing to the OpenGL API they planned to keep their customers locked onto SGI (and IBM) hardware for a few years while market support for OpenGL matured to bring to market 3D hardware, supported by extensions made to the PHIGS standard. In 1992, SGI led the creation of the OpenGL architectural review board (OpenGL ARB), the group of companies that would maintain.

- OpenGL specification took for years to come. On 17 December 1997, Microsoft and SGI initiated the Fahrenheit project, which was a joint effort with the goal of unifying the OpenGL and Direct3D interfaces (and adding a scene-graph API too). In 1998 Hewlett-Packard joined the project. It initially showed some promise of bringing order to the world of interactive 3D computer graphics APIs, but on account of financial constraints at SGI, strategic reasons at Microsoft, and general lack of industry support, it was abandoned in 1999.

- Many openGL functions are used for rendering and transformation purposes. Transformations functions like glRotate (), glTranslate (), glScaled () can be used.

- OpenGL provides a powerful but primitive set of rendering command, and all higher-level drawing must be done in terms of these commands. There are several libraries that allow you to simplify your programming tasks, including the following:

- OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections and rendering surfaces.

- OpenGL Utility Toolkit (GLUT) is a window-system-independent toolkit, written by Mark Kill guard, to hide the complexities of differing window APIs.

- To achieve the objective of the project, information related to the light sources is required with OpenGL we can manipulate the lighting and objects in a scene to create many different kinds of effects. It explains how to control the lighting in a scene, discusses the OpenGL conceptual model of lighting, and describes in detail how to set the numerous illumination parameters to achieve certain effects and this concept is being obtained.

- To demonstrate the transformation and lightening, effects, different polygons have to be used. Polygons are typically drawn by filling in all the pixels enclosed within the boundary, but we can also draw them as outlined polygons or simply as points at the vertices. This concept is obtained from.

- The properties of a light source like its material, diffuse, emissive, has to mention in the project. So to design the light source and the objects, programming guide of an OpenGL is used.

CHAPTER 3

SYSTEM REQUIREMENT SPECIFICATION

3.1 Hardware Requirements:

- Minimum hardware specification-

- Microprocessor: Intel® Pentium® CPU N3710@1.60GHz*4
- Main memory : 3.8GiB
- Hard Disk : 75GB
- Hard disk speed in RPM:5400 RPM
- Keyboard: QWERTY Keyboard
- Monitor : 1024 x 768 display resolution

3.2 Software Requirements:

- Minimum software specification-

- Operating system : WINDOWS 7
- Tool Used : OpenGL
- OPENGL Library
- X86
- X64(WOW)
- Graphics Driver
- C Language

CHAPTER 4

DESIGN

4.1 EXISTING SYSTEM:

Existing system for a graphics is the TC++. This system will support only the 2D graphics. 2D graphics package being designed should be easy to use and understand. It should provide various options such as free hand drawing, line drawing, polygon drawing, filled polygons, flood fill, translation, rotation, scaling, clipping etc. Even though these properties were supported, it was difficult to render 2D graphics cannot be very . Even the effects like dynamic background, dynamic lighting cannot be provided. So we go for CodeBlocks 20.03.

4.2 PROPOSED SYSTEM:

To achieve UI of the game, open GL software is proposed. It is software which provides a graphical interface. It is a interface between application program and graphics hardware.

The advantages are: -

1. Open GL is designed as a streamlined.

2. It's a hardware independent interface i.e., it can be implemented on many different hardware platforms.

3. With OpenGL we can draw a small set of geometric primitives such as points, lines and polygons etc.

4. It provides double buffering which is vital in providing transformations.

5. It is event driven software.

6. It provides call back function.

4.3 LOW LEVEL DESIGN:

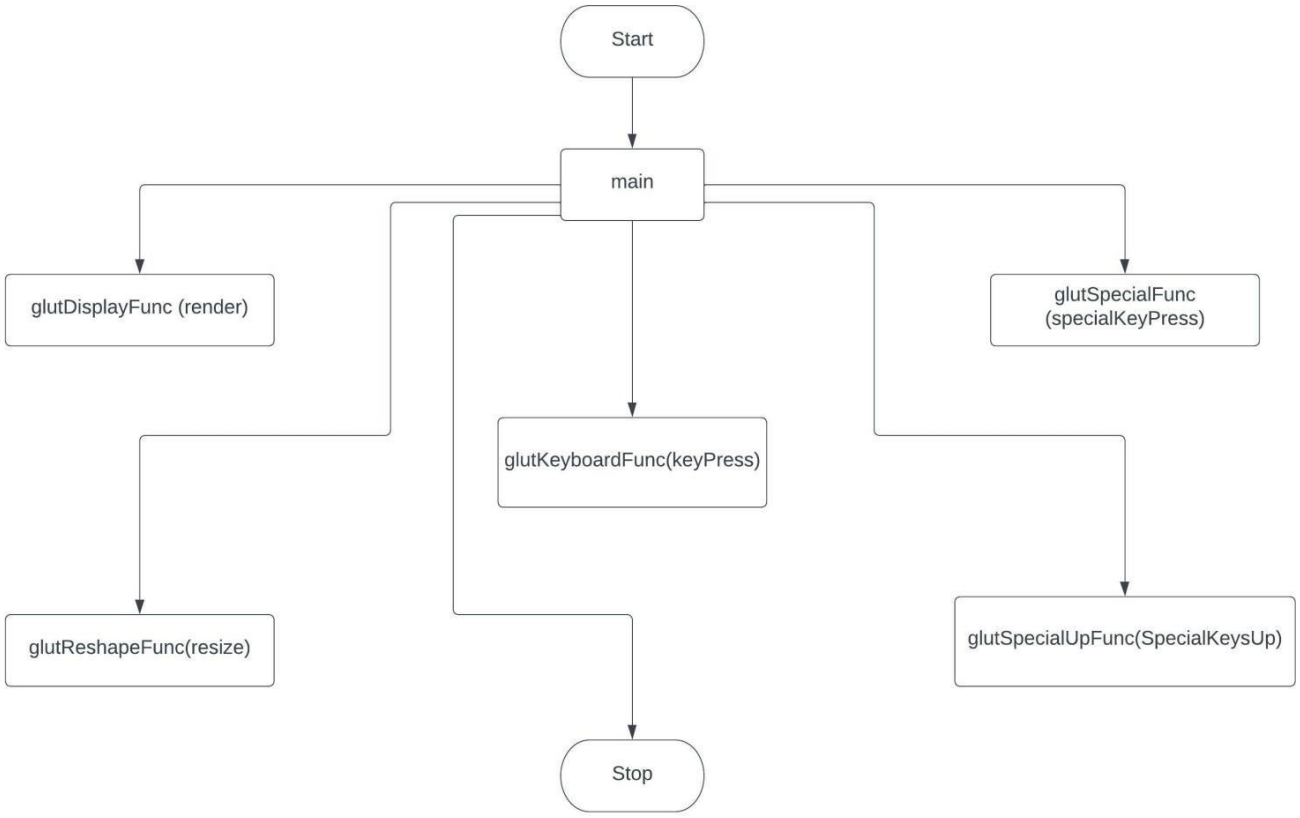

Fig 4.1 Design of the low level

4.4 User Defined Functions:

- **game_screen():**

This function initializes the starting/home screen of the game.

- **Shuttle():**

This function creates the aircraft using OpenGL polygon transformations .

- **timerfunc():**

This function starts a timer in the event loop that delays the event loop for delay miiliseconds.

- **MainLoop():**

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

- **circular_aliens():**

This function create the alien space ships.

- **new_wave_initialization():**

This function is used to initialize a new wave after one wave is completed.

- **Shuttle_bullet():**

This function create the laser-bullet for the aircraft.

- **main():**

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

- **PostRedisplay():**

It ensures that the display will be drawn only once each time the program goes through the event loop.

CHAPTER 5

IMPLEMENTATION

5.1 FUNCTIONS:

GL_LINES-

Treats each pair of vertices as an independent line segment.

Vertices 2n -1 and 2n define line n. N/2 lines are drawn.

GL_LINE_LOOP -

Draws a connected group of line segments from the first vertex to the last, then back to the first. Vertices n and n + 1 define line n. The last line, however, is defined by vertices N and N lines are drawn.

Basic Functions:-

glPushMatrix, glPopMatrix Function-

The glPushMatrix and glPopMatrix functions push and pop the current matrix stack.

SYNTAX: void glPushMatrix();

void glPopMatrix(void);

glBegin, glEnd Function-

The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives.

SYNTAX:void glBegin, glEnd(GLenum mode);

PARAMETERS: mode

The primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd. The following are accepted symbolic constants and their meanings:

Transformation Functions:-

glTranslate Function-

The glTranslated and glTranslatef functions multiply the current matrix by a translation matrix.

SYNTAX:void glTranslate(x, y, z);

PARAMETERS: x, y, z -The x, y, and z coordinates of a translation vector.

Funtions used to display:-

glMatrixMode Function-

The glMatrixMode function specifies which matrix is the current matrix.

SYNTAX: void glMatrixMode(GLenum mode);

PARAMETERS:

mode -The matrix stack that is the target for subsequent matrix operations. The mode parameter can assume one of three values:

Value	Meaning
GL_MODELVIEW	Applies subsequent matrix operations to the model view matrix stack.

glLoadIdentity Function-

The glLoadIdentity function replaces the current matrix with the identity matrix.

SYNTAX: void glLoadIdentity(void);

5.2 FUNCTIONS USED TO SET THE VIEWING VOLUME:

glOrtho-

This function defines orthographic viewing volume with all parameters measured from the centre of projection.

Multiply the current matrix by a perspective matrix.

SYNTAX: void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,

GLdouble near, GLdouble far)

PARAMETERES:

- left, right -Specify the coordinates for the left and right vertical clipping planes.
- bottom, top -Specify the coordinates for the bottom and top horizontal clipping planes.
- nearVal, farVal -Specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

5.3 CALL BACK FUNCTIONS:

glutDisplayFunc Function-

glutDisplayFunc sets the display callback for the current window.

SYNTAX: void glutDisplayFunc(void (*func)(void));

glutReshapeFunc Function-

glutReshapeFunc sets the reshape callback for the current window.

SYNTAX:void glutReshapeFunc(void (*func)(int width, int height));

5.4 MAIN FUNCTION:

glutInit Function-

glutInit is used to initialize the GLUT library.

SYNTAX:glutInit(int *argcp, char**argv);

PARAMETERS:

- argcp -A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argcs will be updated, because glutInit extracts any command line options intended for the GLUT library.
- Argv -The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.
- glutInit(&argc,argv);

glutInitDisplayMode Function-

glutInitDisplayMode sets the initial display mode.

SYNTAX:void glutInitDisplayMode(unsigned int mode);

PARAMETERS:

mode -Display mode, normally the bitwise OR-ing of GLUT display mode bit masks. See values below:

GLUT_RGB: An alias for GLUT_RGBA.

GLUT_DOUBLE: Bit mask to select a double buffered window. This overrides GLUT_SINGLE.

GLUT_DEPTH: Bit mask to select a window with a depth buffer.

CHAPTER 6

SNAPSHOTS

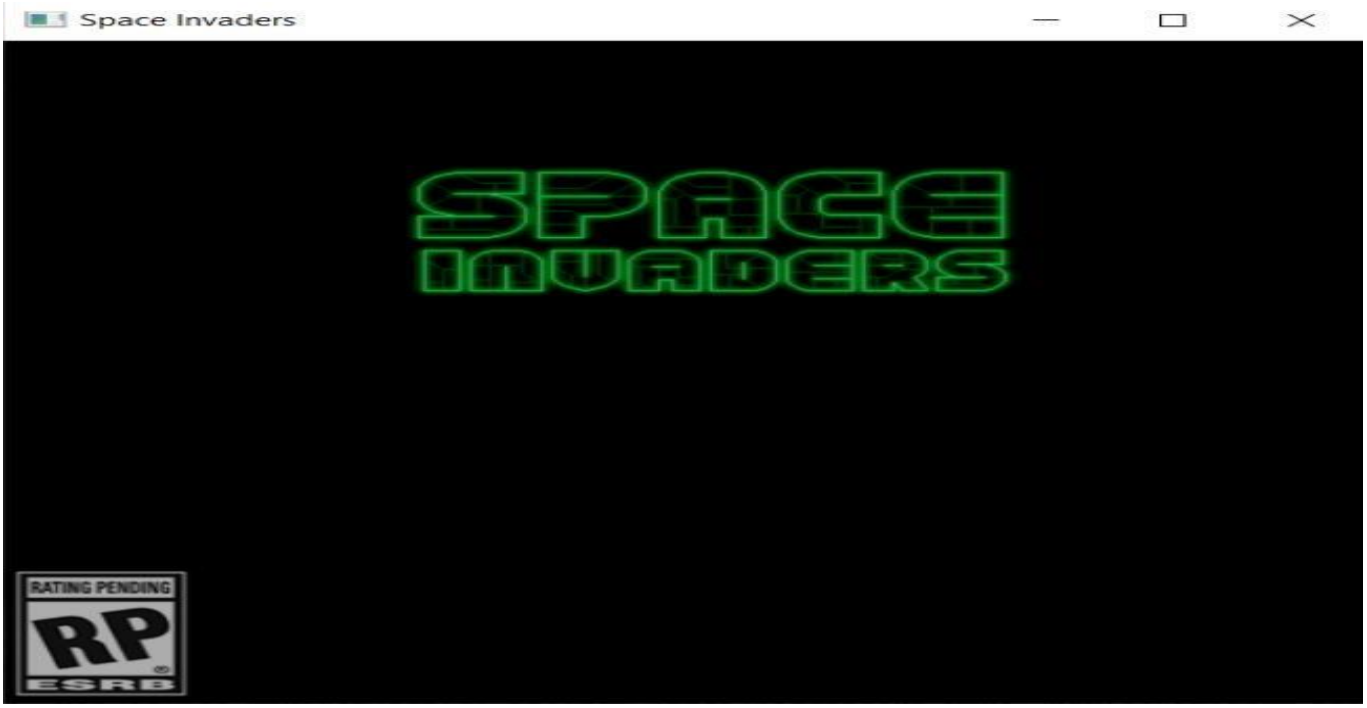

Figure6.1: Starting Screen of Space Invaders

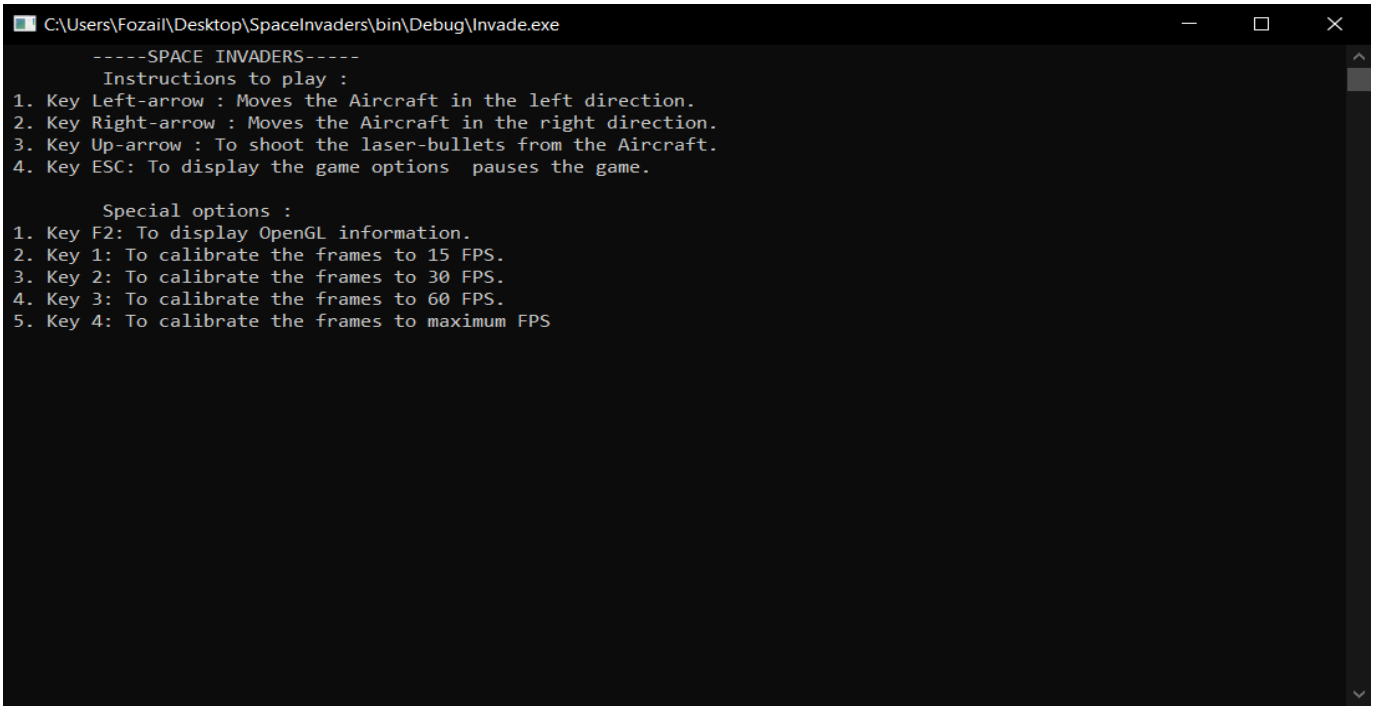

Figure6.2: Instruction to play

Figure6.3:Loading-Screen

Figure6.4: Options list

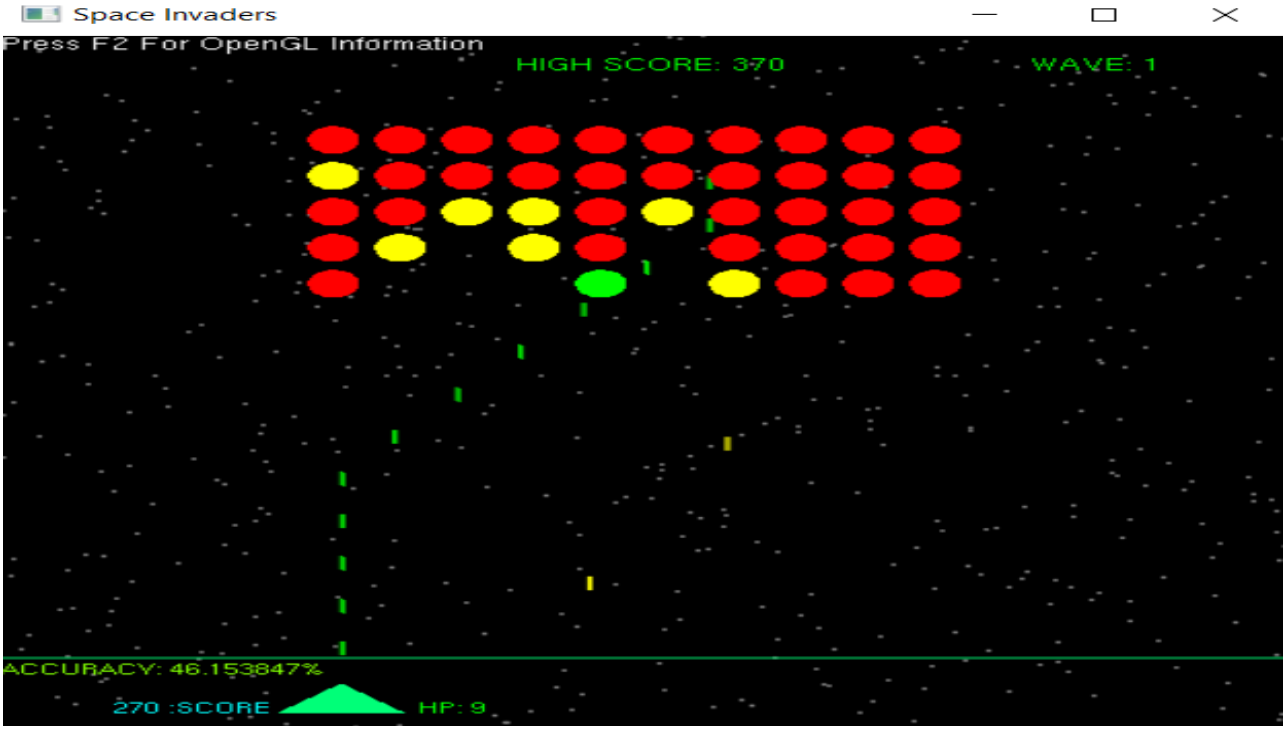

Figure6.5: Game view in solid mode

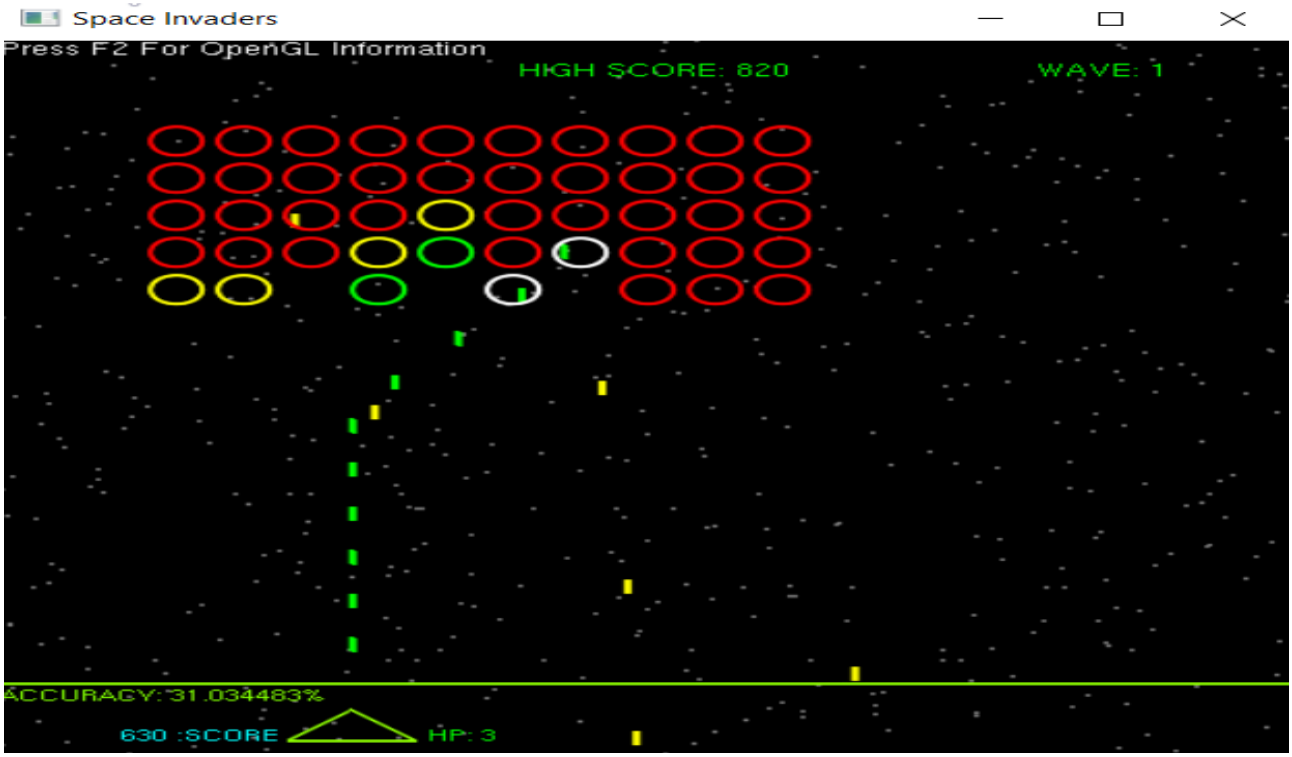

Figure6.6: Game view in hollow mode

Figure6.7: Wave completed

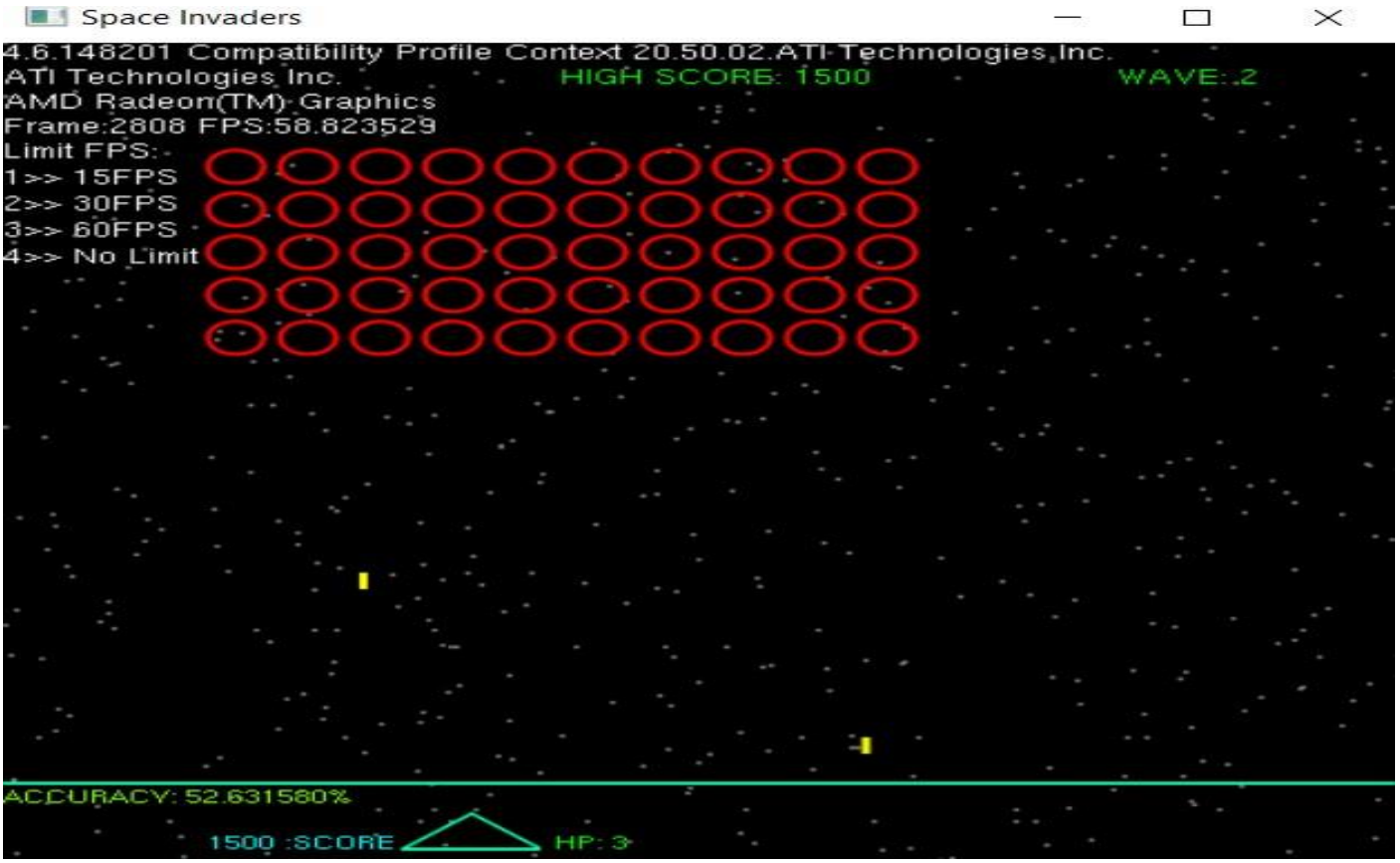

Figure6.8: OpenGL Information and FPS

APPENDIX

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<time.h>
#include<GL/freeglut.h>
//#include "load_textures.h" //FROM FILE
#include "textures.c"
#define num_textures 3
GLuint textureID[num_textures];
//unsigned char *data;        //FROM FILE
//unsigned int width[num_textures],height[num_textures]; //FROM FILE
void DrawTexturedRect(int x,int y,int w,int h,GLuint texture,unsigned int fading,unsigned int i,unsigned int ill_change);
unsigned int fade_color[num_textures],fade_swap[num_textures];
unsigned char display_screen=0; //Type Of Interface To Display
void opengl_info(void);
char gl_info_string[6][50];
void *font = GLUT_BITMAP_9_BY_15;
void *font1 = GLUT_BITMAP_HELVETICA_10;
void *font2 = GLUT_BITMAP_HELVETICA_12;
unsigned char vendor_info=0;
void smooth_input(void); //Smooth Input By Taking Advantage Of glutSpecialUpFunc
unsigned char flag_left_arrow=0,flag_right_arrow=0,flag_up_arrow=0; //Flags To Check If Keys Are Pressed
unsigned long int bullets_fired=0;
float shb_delay=0.1;
clock_t shb_time1=0;
double shb_timediff=0.0;
```

```
void display_string(int x,int y,void *font_to_use,char *string_to_display)
{
	glRasterPos2i(x,y);
	glutBitmapString(font_to_use,(unsigned char*)string_to_display);
}
void resize(int width,int height)
{
	glutReshapeWindow(500,500);
}
void render(void);
clock_t time1=0,time2=0,time3=0;
double timediff,fps_timediff,fps=60.0,delta_time=0.0166666666666667;
unsigned long int frame_count=0;
void splash_screen(void);
unsigned char splash_screen_to_display=0;
void main_menu_screen(void);
unsigned char menu_option=0,difficulty_option=0;
unsigned char difficulty_score_multiplier=10;
void options_screen(void);
unsigned char options_screen_option=0;
unsigned char dynamic_background=1; //STATIC 0 or DYNAMIC 1
unsigned char objects_theme=0;
void game_screen(void);
void game_hud(void);
unsigned long int game_score=0,game_high_score=0,wave_count=0;
unsigned char new_high_score_flag=0;
float accuracy=0.0;

unsigned char digit_count(unsigned long int num);
void paused_screen(void);
```

```
unsigned char paused_option=0;
void wave_completed_screen(void);
unsigned char wave_completed_screen_option=0;
void game_over_screen(void);
void keyPress(unsigned char key, int x, int y);
void specialKeyPress(int key, int x, int y);
void SpecialKeysUp(int key, int x, int y);
void star_background(unsigned char);
#define num_stars 501
struct stars
{
	float star_x_pos;
	float star_y_pos;
}star[num_stars];
void color_change(void);
int red=255,green=0,blue=0;
float cc_time_delay=0.01;
clock_t cc_time1=0;
double cc_timediff=0.0;
void shuttle(void);
float shuttle_x_pos=225.0,shuttle_y_pos=10.0;
int shuttle_hitpoints=3;
unsigned char shuttle_i_got_hit=0;
void shuttle_bullet(void);
#define max_bullets 50
struct shuttle_bullet
{
	float x_pos,y_pos;
	unsigned char allocated;
}sh_bullet[max_bullets];
```

```
float triangle_area(float x1,float y1,float x2,float y2,float x3,float y3);
unsigned char collision_with_shuttle(float x1,float y1,float x2,float y2,float x3,float y3,float x,float y);
float shuttle_area;
void circular_aliens(void);
#define num_cir_aliens 50
unsigned char ca_movement_type=1;
float alien_x_pos,alien_y_pos,r=10.0,angle;
struct circular_aliens
{
        float alien_initial_x_pos,alien_initial_y_pos,alien_min_x_pos,alien_max_x_pos;
        unsigned char alien_hitpoints,i_got_hit,alien_direction,go;
}cir_alien[num_cir_aliens];
void create_ca_alive_list(void);
unsigned char ca_alive_count;
int ca_alive[num_cir_aliens];
void ca_bullet_alloc(void);
unsigned char alien_that_will_shoot=0;
float ca_bullet_alloc_time_delay=0.5;
clock_t ca_time1=0;
double ca_timediff=0.0;
void cir_alien_bullet(void);
//USES max_bullets OF shuttle_bullets
struct cir_alien_bullet
{
        float x_pos,y_pos;
        unsigned char allocated;
}ca_bullet[max_bullets];
void new_wave_initialization(void)
{
        int i,j=0;
```

```
wave_count++;
wave_completed_screen_option=0;
red=255,green=0,blue=0;
shuttle_x_pos=225.0,shuttle_y_pos=10.0;
shuttle_i_got_hit=0;
ca_movement_type=rand()%3;
for(i=0;i<50;i++)
{
        cir_alien[i].alien_initial_x_pos=(60.0+((i%10)*26.0)); //Add x amount
        if(ca_movement_type==0 || ca_movement_type==2)
        {
                cir_alien[i].alien_min_x_pos=cir_alien[i].alien_initial_x_pos;
                cir_alien[i].alien_max_x_pos=cir_alien[i].alien_min_x_pos+146.0; //Decrease by 3*x the amount
        }
        else if(ca_movement_type==1)
        {
                cir_alien[i].alien_min_x_pos=60.0;
                cir_alien[i].alien_max_x_pos=440.0;
        }
        if(i>9 && i%10==0)j++;
        cir_alien[i].alien_initial_y_pos=(440.0-(j*26.0)); //Add x amount
        cir_alien[i].alien_hitpoints=3;
        cir_alien[i].i_got_hit=0;
        cir_alien[i].alien_direction=cir_alien[i].go=1;
}
for(i=0;i<max_bullets;i++)
{
        sh_bullet[i].allocated=0;
        ca_bullet[i].allocated=0;
}
```

```
}
void new_game_initialization(void)
{
	frame_count=0;
	vendor_info=0;
	options_screen_option=0;
	paused_option=0;
	game_score=0;
	new_high_score_flag=0;
	bullets_fired=0;
	accuracy=0.0;
	if(difficulty_option==0) //EASY
	{
		shuttle_hitpoints=10;
		difficulty_score_multiplier=10;
		ca_bullet_alloc_time_delay=0.5;
	}
	else if(difficulty_option==1) //MODERATE
	{
		shuttle_hitpoints=6;
		difficulty_score_multiplier=20;
		ca_bullet_alloc_time_delay=0.3;
	}
	else if(difficulty_option==2) //HARD
	{
		shuttle_hitpoints=3;
		difficulty_score_multiplier=30;
		ca_bullet_alloc_time_delay=0.2;
	}
	wave_count=0;
```

```
	new_wave_initialization();
}
int main(int argc,char **argv)
{	int i;
	for(i=0;i<num_textures;i++)
	{
		fade_color[i]=0;
		fade_swap[i]=1;
	}
	for(i=0;i<num_stars;i++)
	{
		star[i].star_x_pos=rand()%501;
		star[i].star_y_pos=(int)(i*(501.0/num_stars))%501;
	}
	new_game_initialization();
	shuttle_area=triangle_area(shuttle_x_pos,shuttle_y_pos,shuttle_x_pos+25.0,shuttle_y_pos+21.6506350946,shuttle_
x_pos+50.0,shuttle_y_pos);
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
	glutInitWindowSize(500,500);
	glutInitWindowPosition((glutGet(GLUT_SCREEN_WIDTH))/2-250,(glutGet(GLUT_SCREEN_HEIGHT))/2-250);
	glutCreateWindow("Space Invaders");
	gluOrtho2D(0.0,500.0,0.0,500.0);
	glutDisplayFunc(render);
	glutIdleFunc(render);
	glutReshapeFunc(resize);
	glutSpecialFunc(specialKeyPress);
	glutSpecialUpFunc(SpecialKeysUp);
	glutKeyboardFunc(keyPress);
```

```
printf("\t-----SPACE INVADERS------\n \t Instructions to play : \n 1. Key Left-arrow : Moves the Aircraft in the  left direction.\n 2. Key Right-arrow : Moves the Aircraft in the right direction.\n 3. Key Up-arrow : To shoot the laser-bullets from the Aircraft.\n 4. Key ESC: To display the game options pauses the game.");

printf("\n \n \t Special options : \n 1. Key F2: To display OpenGL information.\n 2. Key 1: To calibrate the frames to 15 FPS. \n 3. Key 2: To calibrate the frames to 30 FPS.\n 4. Key 3: To calibrate the frames to 60 FPS.\n 5. Key 4: To calibrate the frames to maximum FPS ");

glEnable(GL_POINT_SMOOTH);

glHint(GL_POINT_SMOOTH_HINT,GL_NICEST);

glEnable(GL_LINE_SMOOTH);

glHint(GL_LINE_SMOOTH_HINT,GL_NICEST);

glEnable(GL_POLYGON_SMOOTH);

glHint(GL_POLYGON_SMOOTH_HINT,GL_NICEST);

glEnable(GL_BLEND);

glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);

//glEnable(GL_MULTISAMPLE);


//LOAD TEXTURES (Note The Loading After Glut Initialization)

glGenTextures(num_textures,textureID);

//data=load_BMP_raw("nvidia.tex",&width[0],&height[0]);                                    //FROM FILE

//free(data);                                                                                    //FROM FILE

//printf("Width:%u                Height:%u                TextureID:%u\n",width[0],height[0],textureID[0]); //FROM FILE

glBindTexture(GL_TEXTURE_2D,textureID[0]);

glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,nvidia_tex.width,nvidia_tex.height,0,GL_RGB,GL_UNSIGNED_BYTE,nvidia_tex.pixel_data);     //EMBEDDED

//glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,width[0],height[0],0,GL_RGB,GL_UNSIGNED_BYTE,data); //FROM FILE


//data=load_BMP_raw("title.tex",&width[1],&height[1]);                                    //FROM FILE

//free(data);                                                                                    //FROM FILE

//printf("Width:%u                Height:%u                TextureID:%u\n",width[1],height[1],textureID[1]); //FROM FILE

glBindTexture(GL_TEXTURE_2D,textureID[1]);
```

```
	glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,title_tex.width,title_tex.height,0,GL_RGB,GL_UNSIGNED_BYTE
,title_tex.pixel_data);	//EMBEDDED

	//glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,width[1],height[1],0,GL_RGB,GL_UNSIGNED_BYTE,data);
//FROM FILE



	//data=load_BMP_raw("esrb.tex",&width[2],&height[2]);	//FROM
FILE

	//free(data);	//FROM FILE

	//printf("Width:%u	Height:%u	TextureID:%u\n",width[2],height[2],textureID[2]);
//FROM FILE

	glBindTexture(GL_TEXTURE_2D,textureID[2]);

	glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,esrb_tex.width,esrb_tex.height,0,GL_RGB,GL_UNSIGNED_BYT
E,esrb_tex.pixel_data);	//EMBEDDED

	//glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,width[2],height[2],0,GL_RGB,GL_UNSIGNED_BYTE,data);
//FROM FILE

//OPENGL INFORMATION

	strcpy(gl_info_string[0],(char*)glGetString(GL_VERSION));

	strcpy(gl_info_string[1],(char*)glGetString(GL_VENDOR));

	strcpy(gl_info_string[2],(char*)glGetString(GL_RENDERER));

	strcpy(gl_info_string[4],"Press F2 For OpenGL Information");

	glutMainLoop();

	return 0;

}

void opengl_info(void)

{

	glColor3ub(255,255,255);

	if(vendor_info)

	{

		display_string(0,490,font2,gl_info_string[0]); //GL_VERSION

		display_string(0,475,font2,gl_info_string[1]); //GL_VENDOR

		display_string(0,460,font2,gl_info_string[2]); //GL_RENDERER

		sprintf(gl_info_string[3],"Frame:%lu FPS:%f",frame_count,fps);
```

```
                display_string(0,445,font2,gl_info_string[3]); //FRAME and FPS
                sprintf(gl_info_string[5],"Limit FPS:\n1>> 15FPS\n2>> 30FPS\n3>> 60FPS\n4>> No Limit");
                display_string(0,430,font2,gl_info_string[5]);
        }
        else
        {
                display_string(0,490,font2,gl_info_string[4]); //MORE INFO
        }}
void smooth_input(void)
{
        if(flag_left_arrow==1)
        {
                shuttle_x_pos=shuttle_x_pos-((60.0/fps)*4.0);
                if(shuttle_x_pos<0.0)shuttle_x_pos=0.0;
        }
        if(flag_right_arrow==1)
        {
                shuttle_x_pos=shuttle_x_pos+((60.0/fps)*4.0);
                if(shuttle_x_pos>450.0)shuttle_x_pos=450.0;
        }
        shb_timediff=(double)(time2-shb_time1)/CLOCKS_PER_SEC;
        if(flag_up_arrow==1 && shb_timediff>=shb_delay)
        {
                shb_time1=clock();
                int i;
                for(i=0;i<max_bullets;i++)
                {
                        if(sh_bullet[i].allocated==0)
                        {
                                sh_bullet[i].x_pos=shuttle_x_pos+25.0;
```

```
                                sh_bullet[i].y_pos=shuttle_y_pos+21.6506350946;
                                sh_bullet[i].allocated=1;
                                bullets_fired++;
                                break;
                        }
                }
        }
}
void render(void)
{       time2=clock(); //Common For FRAME LIMITER and FPS CALCULATOR
        timediff=(double)(time2-time1)/CLOCKS_PER_SEC; //Common For FRAME LIMITER and FPS CALCULATOR
        if(timediff>=delta_time)
        {       time1=clock();
                glClear(GL_COLOR_BUFFER_BIT);
                if(display_screen==2)game_screen();
                else if(display_screen==0)splash_screen();
                else if(display_screen==1)main_menu_screen();
                else if(display_screen==3)options_screen();
                else if(display_screen==4)paused_screen();
                else if(display_screen==5)wave_completed_screen();
                else if(display_screen==6)game_over_screen();
                fps=(1.0/timediff);
                if(fps<10.0)fps=10.0; //FORCE 10.0 FPS
                glutSwapBuffers();
        }}
        void game_screen(void)
{       star_background(dynamic_background);
        shuttle_bullet();
        create_ca_alive_list();
        ca_bullet_alloc();
```

```
	cir_alien_bullet();
	circular_aliens();
	smooth_input();
	shuttle();
	game_hud();
	opengl_info();
	frame_count++;
}
void game_hud(void)
{
	char game_hud_string[5][25];
	if(game_score>game_high_score)
	{	new_high_score_flag=1;
		game_high_score=game_score;
	}
	sprintf(game_hud_string[0],"HIGH SCORE: %lu",game_high_score);
	glColor3ub(0,255,0);
	display_string(200,475,font2,game_hud_string[0]);
	sprintf(game_hud_string[1],"WAVE: %lu",wave_count);
	glColor3ub(0,255,0);
	display_string(400,475,font2,game_hud_string[1]);
	glColor3ub(red,green,blue);
	glBegin(GL_LINES);
	glVertex2f(0,50);
	glVertex2f(500,50);
	glEnd();
	if(bullets_fired==0)accuracy=0.0;
	else accuracy=(((wave_count*50.0*3.0)-(ca_alive_count*3.0))/bullets_fired)*100.0;
	sprintf(game_hud_string[2],"ACCURACY: %f%%",accuracy);
	glColor3ub(128,255,0);
```

```
        display_string(0,38,font1,game_hud_string[2]);
        unsigned char digit_size=(digit_count(game_score))*6;
        sprintf(game_hud_string[3],"%lu :SCORE",game_score);
        glColor3ub(0,255,255);
        display_string(shuttle_x_pos-45-digit_size,shuttle_y_pos,font1,game_hud_string[3]);
        sprintf(game_hud_string[4],"HP: %d",shuttle_hitpoints);
        if(shuttle_hitpoints>=3)glColor3ub(0,255,0);
        else if(shuttle_hitpoints==2)glColor3ub(255,255,0);
        else glColor3ub(255,0,0);
        display_string(shuttle_x_pos+55,shuttle_y_pos,font1,game_hud_string[4]);
}
void splash_screen(void)
{
                if(splash_screen_to_display==0)
                {
                        //DrawTexturedRect(0,200,width[0],height[0],textureID[0],1,0,1);              //FROM FILE
                        DrawTexturedRect(0,200,nvidia_tex.width,nvidia_tex.height,textureID[0],1,0,1);  //EMBEDDED
                }
                else if(splash_screen_to_display==1)
                {
                        //DrawTexturedRect(105,230,width[1],height[1],textureID[1],1,1,0);            //FROM FILE
                        //DrawTexturedRect(-10,5,width[2],height[2],textureID[2],1,2,1);              //FROM FILE
                        DrawTexturedRect(105,230,title_tex.width,title_tex.height,textureID[1],1,1,0); //EMBEDDED
                        DrawTexturedRect(-10,5,esrb_tex.width,esrb_tex.height,textureID[2],1,2,1);      //EMBEDDED
                }
                else if(splash_screen_to_display==2)
                {
//DUMMY LOADING PROGRESS BAR
                        char splash_string[3][15];
                        static float splash_x=0;
```

```
                glColor3ub(0,136,24);
                glLineWidth(3.0);
                glBegin(GL_LINES);
                glVertex2f(0,5);
                glColor3ub(0,255,0);
                glVertex2f(splash_x,5);
                glEnd();
                sprintf(splash_string[0],"Developed by \nFozail Ahmed \n 1CE19CS031\n");
                display_string(205,250,font,splash_string[0]);
                sprintf(splash_string[1],"\n\n\nDeepak Jadon\n 1CE19CS027");
                display_string(205,250,font,splash_string[1]);
                sprintf(splash_string[2],"%.2f%%",((splash_x/500.0)*100));
                display_string(430,10,font,splash_string[2]);
                splash_x=splash_x+((60.0/fps)*1.8);
                if(splash_x>499.0)
                {
                        glLineWidth(1.0);
                        display_screen=1;
                        glDeleteTextures(num_textures,textureID);
                }
        }
void DrawTexturedRect(int x,int y,int w,int h,GLuint texture,unsigned int fading,unsigned int i,unsigned int ill_change)
{
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D,texture);
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        if(fading==1)
        {       glColor3ub(fade_color[i],fade_color[i],fade_color[i]);
                if(fade_swap[i]==1)
```

```
                {       fade_color[i]++;
                        if(fade_color[i]==255)fade_swap[i]=!fade_swap[i];
                }
                else
                {       fade_color[i]--;
                        fade_color[i]--;
                        if(fade_color[i]==1 && ill_change==1)splash_screen_to_display++;
                }
        else glColor3ub(255,255,255);
        glBegin(GL_QUADS);
   glTexCoord2i(0,0); glVertex2i(x,y);
   glTexCoord2i(1,0); glVertex2i(x+w,y);
   glTexCoord2i(1,1); glVertex2i(x+w,y+h);
   glTexCoord2i(0,1); glVertex2i(x,y+h);
   glEnd();
glDisable(GL_TEXTURE_2D);
}

void main_menu_screen(void)
{       char menu_string[9][30];
        sprintf(menu_string[0],"HIGH SCORE : %lu",game_high_score);
        strcpy(menu_string[1],"New Game");
        strcpy(menu_string[2],"Difficulty:");
        strcpy(menu_string[3],"Easy");
        strcpy(menu_string[4],"Moderate");
        strcpy(menu_string[5],"Hard");
        strcpy(menu_string[6],"Options");
        strcpy(menu_string[7],"Quit");
        strcpy(menu_string[8],"v1.0");
        star_background(dynamic_background);
```

```
        if(game_high_score>0)
        {       glColor3ub(0,255,0);
                display_string(200,475,font2,menu_string[0]);
        }
        if(menu_option==0)glColor3ub(255,255,255);
        else glColor3ub(255,0,0);
        display_string(200,250,font,menu_string[1]);
        if(menu_option==1)glColor3ub(255,255,255);
        else glColor3ub(255,0,0);
        display_string(200,235,font,menu_string[2]);
        if(difficulty_option==0)
        {       glColor3ub(0,255,0);
                display_string(305,235,font,menu_string[3]);
        }
        else if(difficulty_option==1)
        {
                glColor3ub(255,255,0);
                display_string(305,235,font,menu_string[4]);
        }
        else if(difficulty_option==2)
        {       glColor3ub(255,0,0);
                display_string(305,235,font,menu_string[5]);
        }
    if(menu_option==2)glColor3ub(255,255,255);
    else glColor3ub(255,0,0);
    display_string(200,220,font,menu_string[6]);
    if(menu_option==3)glColor3ub(255,255,255);
    else glColor3ub(255,0,0);
    display_string(200,205,font,menu_string[7]);
    glColor3ub(255,0,0);
```

```
    display_string(455,10,font,menu_string[8]);
}
void options_screen(void)
{       char options_string[8][25];
        strcpy(options_string[0],"OPTIONS");
        strcpy(options_string[1],"DISPLAY");
        strcpy(options_string[2],"Dynamic Background: ");
        strcpy(options_string[3],"ON");
        strcpy(options_string[4],"OFF");
        strcpy(options_string[5],"Objects Theme:");
        strcpy(options_string[6],"SOLID POLYGONS");
        strcpy(options_string[7],"HOLLOW BRIGHT");
        star_background(dynamic_background);
        glColor3ub(255,0,0);
        display_string(210,310,font,options_string[0]);
        display_string(135,295,font,options_string[1]);
        if(options_screen_option==0)glColor3ub(255,255,255);
        else glColor3ub(255,0,0);
        display_string(135,280,font,options_string[2]);
        if(dynamic_background==1)
        {       glColor3ub(0,255,0); //Green Color
                display_string(315,280,font,options_string[3]);
        }
        else if(dynamic_background==0) //OFF
        {       glColor3ub(255,0,0); //Red Color
                display_string(315,280,font,options_string[4]);
        }
        if(options_screen_option==1)glColor3ub(255,255,255);
        else glColor3ub(255,0,0);
        display_string(135,265,font,options_string[5]);
```

```
	if(objects_theme==0) //SOLID POLYGONS
	{	glColor3ub(128,255,0);
		display_string(315,265,font,options_string[6]);
	}
	else if(objects_theme==1) //HOLLOW BRIGHT
	{	glColor3ub(255,255,0);
		display_string(315,265,font,options_string[7]);
	}}
void paused_screen(void)
{	char paused_string[3][50];
	strcpy(paused_string[0],"GAME PAUSED");
	strcpy(paused_string[1],"Resume");
	strcpy(paused_string[2],"Quit To Main Menu");
	star_background(dynamic_background);
	glColor3ub(255,0,0);
	display_string(200,250,font,paused_string[0]);
	if(paused_option==0)glColor3ub(255,255,255);
	else glColor3ub(255,0,0);
	display_string(200,235,font,paused_string[1]);
	if(paused_option==1)glColor3ub(255,255,255);
	else glColor3ub(255,0,0);
	display_string(200,220,font,paused_string[2]);
}
void wave_completed_screen(void)
{	char wave_completed_string[5][25];
	strcpy(wave_completed_string[0],"WAVE COMPLETED!");
	strcpy(wave_completed_string[3],"Next Wave >>>");
	strcpy(wave_completed_string[4],"Quit To Main Menu");
	star_background(dynamic_background);
	glColor3ub(0,255,0);
```

```
	display_string(180,300,font,wave_completed_string[0]);
	glColor3ub(238,130,238);
	if(new_high_score_flag==1)sprintf(wave_completed_string[1],"Score: %lu *NEW HIGH SCORE*",game_high_score);
	else sprintf(wave_completed_string[1],"Score: %lu",game_score);
	display_string(180,270,font,wave_completed_string[1]);
	glColor3ub(0,255,255);
	sprintf(wave_completed_string[2],"Accuracy: %f%%",accuracy);
	display_string(180,240,font,wave_completed_string[2]);
	if(wave_completed_screen_option==0)glColor3ub(255,255,255);
	else glColor3ub(255,0,0);
	display_string(180,190,font,wave_completed_string[3]);
	if(wave_completed_screen_option==1)glColor3ub(255,255,255);
	else glColor3ub(255,0,0);
	display_string(180,175,font,wave_completed_string[4]);
}
void game_over_screen(void)
{	char game_over_string[4][25];
	strcpy(game_over_string[0],"GAME OVER!");
	strcpy(game_over_string[3],"<<< Back To Main Menu");
	star_background(dynamic_background);
	glColor3ub(255,0,0);
	display_string(180,300,font,game_over_string[0]);
	glColor3ub(238,130,238);
	if(new_high_score_flag==1)sprintf(game_over_string[1],"Score: %lu *NEW HIGH SCORE*",game_high_score);
	else sprintf(game_over_string[1],"Score: %lu",game_score);
	display_string(180,270,font,game_over_string[1]);
	glColor3ub(0,255,255);
	sprintf(game_over_string[2],"Accuracy: %f%%",accuracy);
	display_string(180,240,font,game_over_string[2]);
```

```
            glColor3ub(255,255,255);
            display_string(135,190,font,game_over_string[3]);
}
void keyPress(unsigned char key, int x, int y){
            if(display_screen==2)
            {switch(key){
                        case '1':
           delta_time=0.0666666666666667; //For 15fps
           break;
       case '2':
            delta_time=0.0333333333333333; //For 30fps
           break;
       case '3':
            delta_time=0.0166666666666667; //For 60fps
            break;
       case '4':
            delta_time=0.0; //No Frame Limit,Although glutSwapBuffers waits for vSync
            break;
            case 27 :
          display_screen=4;  //Switch To Paused Screen (Pause Game)      }}
            else if(display_screen==0)
            {           if((splash_screen_to_display==0 || splash_screen_to_display==1) && key==27)
                        {           splash_screen_to_display++;
                        }}
            else if(display_screen==1)
   {switch(key){
                        case 13: //Press Enter Key To Select Option
                                    if(menu_option==0){new_game_initialization();display_screen=2;} //New Game -> Game Screen
                                    else if(menu_option==2)display_screen=3; //Options -> Options Screen
                                    else if(menu_option==3)exit(0); //Quit
```

```
        }}
        else if(display_screen==3)
{switch(key)
        {
        case 27:
                display_screen=1;
        }
}
        else if(display_screen==4)
{
        switch(key)
        {
        case 13:
                if(paused_option==0)display_screen=2; //Paused Screen -> Game Screen (Resume Game)
                else if(paused_option==1)display_screen=1; //Quit To Main Menu
        }    }
        else if(display_screen==5)
{switch(key)
        {       case 13:
                if(wave_completed_screen_option==0)
                {                       new_wave_initialization();
                        display_screen=2; //Start New Wave
                }
                else
                        display_screen=1; //Quit To Main Menu}}
        else if(display_screen==6)
{
        switch(key)
        {
        case 13:
```

```
                display_screen=1; //Back To Main Menu
 }}
void specialKeyPress(int key, int x, int y)
{       if(display_screen==2)
        {       if(GLUT_KEY_LEFT==key)flag_left_arrow=1;
                else if(GLUT_KEY_RIGHT==key)flag_right_arrow=1;
                else if(GLUT_KEY_UP==key)flag_up_arrow=1;
        else if(GLUT_KEY_F2==key)vendor_info=!vendor_info; //Negate 0->1 (or) 1->0
        }
        else if(display_screen==1){
        switch(key){
                case GLUT_KEY_UP:
                menu_option=(menu_option-1)%4;
                        if(menu_option==255)menu_option=3; //Unsigned char,(0-1)=255
                        break;
                case GLUT_KEY_DOWN:
                        menu_option=(menu_option+1)%4;
                        break;
                case GLUT_KEY_LEFT:
                        if(menu_option==1){             {
                                difficulty_option=(difficulty_option-1)%3;
                                if(difficulty_option==255)difficulty_option=2;
                        }break;
                case GLUT_KEY_RIGHT:
                        if(menu_option==1)
                        {       difficulty_option=(difficulty_option+1)%3;
                        }break;}}
        else if(display_screen==3)
        {switch(key)    {
                case GLUT_KEY_UP:
```

```
            options_screen_option=!options_screen_option;
            break;
            case GLUT_KEY_DOWN:
            options_screen_option=!options_screen_option;
            break;
            case GLUT_KEY_LEFT:
            if(options_screen_option==0)dynamic_background=!dynamic_background; //ON or OFF
                        else if(options_screen_option==1)objects_theme=!objects_theme;
                        break;
            case GLUT_KEY_RIGHT:
                        if(options_screen_option==0)dynamic_background=!dynamic_background; //ON or OFF
                        else if(options_screen_option==1)objects_theme=!objects_theme;
                        break;
            }}
else if(display_screen==4){
            switch(key){
            case GLUT_KEY_UP:
                        paused_option=(paused_option-1)%2;
                        if(paused_option==255)paused_option=1; //Unsigned char,(0-1)=255
                        break;
            case GLUT_KEY_DOWN:
                        paused_option=(paused_option+1)%2;
                        break;}}
else if(display_screen==5)
{switch(key){
case GLUT_KEY_UP:
                        wave_completed_screen_option=!wave_completed_screen_option;
                        break;
            case GLUT_KEY_DOWN:
                        wave_completed_screen_option=!wave_completed_screen_option;
```

```
                        break;}}
void SpecialKeysUp(int key, int x, int y)
{       if(GLUT_KEY_LEFT==key)flag_left_arrow=0;
        else if(GLUT_KEY_RIGHT==key)flag_right_arrow=0;
        else if(GLUT_KEY_UP==key)flag_up_arrow=0;
}void star_background(unsigned char option)
{       if(option==0)
        {int i;
                glColor3ub(128,128,128); //White Color
                glPointSize(2.0);
                glBegin(GL_POINTS);
                for(i=0;i<num_stars;i++)
                {
        glVertex2f(star[i].star_x_pos,star[i].star_y_pos); //Draw Stars
                }
                glEnd();
        }
        else if(option==1)
        {       int i=0;
                glColor3ub(128,128,128); //White Color
                glPointSize(2.0);
                glBegin(GL_POINTS);
                while(i<num_stars)
                {glVertex2f(star[i].star_x_pos,star[i].star_y_pos); //Draw Stars
                        star[i].star_y_pos=star[i].star_y_pos-((60.0/fps)*1.0); //Decrease y_pos
                        if(star[i].star_y_pos<0)star[i].star_y_pos=500; // -1 -> 500
                        else if(star[i].star_y_pos==0)star[i].star_x_pos=rand()%501; //Create New x_pos at y_pos=0
                        i++;
                }glEnd();}}
void color_change(void)
```

```
{       cc_timediff=(double)(time2-cc_time1)/CLOCKS_PER_SEC;
        if(cc_timediff>=cc_time_delay)
        {               cc_time1=clock();
        if(red==255 && green<255 && blue==0)green++;
                else if(red>0 && green==255 && blue==0)red--;
                else if(red==0 && green==255 && blue<255)blue++;
                else if(red==0 && green>0 && blue==255)green--;
                else if(red<255 && green==0 && blue==255)red++;
                else if(red==255 && green==0 && blue>0)blue--;
        }}
void shuttle(void)
        int i;
        for(i=0;i<max_bullets;i++)
        {
        if(ca_bullet[i].allocated==1&&ca_bullet[i].y_pos<50.0&&collision_with_shuttle(shuttle_x_pos,shuttle_y_pos,shuttl
e_x_pos+25.0,shuttle_y_pos+21.6506350946,shuttle_x_pos+50.0,shuttle_y_pos,ca_bullet[i].x_pos,ca_bullet[i].y_pos))
                {       ca_bullet[i].allocated=0;
                        shuttle_hitpoints=shuttle_hitpoints-1;
                        if(shuttle_hitpoints<1)display_screen=6; //GAME OVER
                        shuttle_i_got_hit=shuttle_i_got_hit+5;
                }}
        color_change();
        if(shuttle_i_got_hit>0){
                glColor3ub(255,255,255);
                shuttle_i_got_hit--;
        }
        else glColor3ub(red,green,blue);
        if(objects_theme==0)glBegin(GL_TRIANGLES);
        else if(objects_theme==1){glLineWidth(2.0);glBegin(GL_LINE_LOOP);}
        glVertex2f(shuttle_x_pos,shuttle_y_pos);
```

```
        glVertex2f(shuttle_x_pos+25.0,shuttle_y_pos+21.6506350946);
        glVertex2f(shuttle_x_pos+50.0,shuttle_y_pos);
        glEnd();
}
void shuttle_bullet(void)
{       int i;
        glColor3ub(0,255,0);
        for(i=0;i<max_bullets;i++)
        {       if(sh_bullet[i].allocated==1)
                {
                sh_bullet[i].y_pos=sh_bullet[i].y_pos+((60.0/fps)*5.0);
                if(sh_bullet[i].y_pos<500.0){
                                glBegin(GL_LINES);
                                glVertex2f(sh_bullet[i].x_pos,sh_bullet[i].y_pos+10);
                                glVertex2f(sh_bullet[i].x_pos,sh_bullet[i].y_pos);
                                glVertex2f(sh_bullet[i].x_pos+1,sh_bullet[i].y_pos);
                                glVertex2f(sh_bullet[i].x_pos+1,sh_bullet[i].y_pos+10);
                                glEnd();
                        }
                        else sh_bullet[i].allocated=0;
                }}}
float triangle_area(float x1,float y1,float x2,float y2,float x3,float y3)
{       return fabs((x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2))/2.0);
}
unsigned char collision_with_shuttle(float x1,float y1,float x2,float y2,float x3,float y3,float x,float y)
{       float area1,area2,area3,area_sum,area_diff;
        area1=triangle_area(x,y,x2,y2,x3,y3);
        area2=triangle_area(x1,y1,x,y,x3,y3);
        area3=triangle_area(x1,y1,x2,y2,x,y);
        area_sum=area1+area2+area3;
```

```
        area_diff=shuttle_area-area_sum;

        return ((area_diff)>-1.0 && (area_diff)<1.0);

}

void circular_aliens(void)

{       int i,j;

  for(i=0;i<num_cir_aliens;i++) //Scan Through All Aliens

    {   if(cir_alien[i].alien_hitpoints>0) //Only Aliens With HP>0

        {

        for(j=0;j<max_bullets;j++) //Scan Through All Bullets

                {
if(sh_bullet[j].allocated==1&&(pow(cir_alien[i].alien_initial_x_possh_bullet[j].x_pos,2)+pow(cir_alien[i].alien_initial_y_po s-
sh_bullet[j].y_pos,2))<=pow(r,2)) //Thank You Pythagoras :)

                        {

                cir_alien[i].alien_hitpoints=cir_alien[i].alien_hitpoints-1;
        cir_alien[i].i_got_hit=cir_alien[i].i_got_hit+5;

        sh_bullet[j].allocated=0;

        game_score=game_score+difficulty_score_multiplier;

           } }

if(cir_alien[i].i_got_hit>0){

                        glColor3ub(255,255,255);

                        cir_alien[i].i_got_hit--;               }

                else if(cir_alien[i].alien_hitpoints==3)glColor3ub(255,0,0);

                else if(cir_alien[i].alien_hitpoints==2)glColor3ub(255,255,0);

                else if(cir_alien[i].alien_hitpoints==1)glColor3ub(0,255,0);

                if(objects_theme==0)glBegin(GL_POLYGON); //FILLED POLYGONS

                else if(objects_theme==1){glPointSize(2.0);glBegin(GL_POINTS);}

        for(angle=0.0;angle<=6.28318531;angle=angle+0.1) //360 degrees

                {

                alien_x_pos=(r*cosf(angle))+cir_alien[i].alien_initial_x_pos;//calculate the x component

                alien_y_pos=(r*sinf(angle))+cir_alien[i].alien_initial_y_pos;//calculate the y component

                glVertex2f(alien_x_pos,alien_y_pos);//output vertex
```

```
                }
                glEnd();
        }}
  if(ca_movement_type==0 || ca_movement_type==1)
  {
for(i=0;i<num_cir_aliens;i++)
    {
        if(cir_alien[i].alien_initial_x_pos<=cir_alien[i].alien_max_x_pos && cir_alien[i].go==1) //Food Is On The Right:D
        {
        if(cir_alien[i].alien_direction==1)cir_alien[i].alien_initial_x_pos=cir_alien[i].alien_initial_x_pos+((60.0/fps)*0.8);
                if(cir_alien[i].alien_initial_x_pos>=cir_alien[i].alien_max_x_pos)
                {       cir_alien[i].alien_initial_x_pos=cir_alien[i].alien_max_x_pos;
                        cir_alien[i].go=0; //Throw Food To The Left
                        cir_alien[i].alien_direction=!cir_alien[i].alien_direction; //Hamster Changes Direction
                        cir_alien[i].alien_initial_y_pos=cir_alien[i].alien_initial_y_pos-((60.0/fps)*5.0);
                        if(cir_alien[i].alien_initial_y_pos<=50.0)display_screen=6; }}
        if(cir_alien[i].alien_initial_x_pos>=cir_alien[i].alien_min_x_pos && cir_alien[i].go==0) //Food Is On The Left :D
        {if(cir_alien[i].alien_direction==0)cir_alien[i].alien_initial_x_pos=cir_alien[i].alien_initial_x_pos-((60.0/fps)*0.8);
                if(cir_alien[i].alien_initial_x_pos<=cir_alien[i].alien_min_x_pos{
                        cir_alien[i].alien_initial_x_pos=cir_alien[i].alien_min_x_pos;
                        cir_alien[i].go=1; //Throw Food To The Right
                        cir_alien[i].alien_direction=!cir_alien[i].alien_direction; //Hamster Changes Direction
                                cir_alien[i].alien_initial_y_pos=cir_alien[i].alien_initial_y_pos-((60.0/fps)*5.0);
        if(cir_alien[i].alien_initial_y_pos<=50.0)display_screen=6; //Invasion Barrier Reached
  }}}
  if(ca_movement_type==2)
  {     for(i=0;i<num_cir_aliens;i++)
    {
        if(cir_alien[i].alien_initial_x_pos<=cir_alien[i].alien_max_x_pos && cir_alien[i].go==1) //Food Is On The Right:D
        {
```

```
        if(cir_alien[i].alien_direction==1)cir_alien[i].alien_initial_x_pos=cir_alien[i].alien_initial_x_pos+((60.0/fps)*0.8);
//Go Get The Food Buddy!
                if(cir_alien[i].alien_initial_x_pos>=cir_alien[i].alien_max_x_pos)
                        {cir_alien[i].alien_initial_x_pos=cir_alien[i].alien_max_x_pos;
                        cir_alien[i].go=0; //Throw Food To The Left
                        cir_alien[i].alien_direction=!cir_alien[i].alien_direction; //Hamster Changes Direction
                        if(i%2==0)
                        {cir_alien[i].alien_initial_y_pos=cir_alien[i].alien_initial_y_pos-((60.0/fps)*10.0);
                                if(cir_alien[i].alien_initial_y_pos<=50.0)display_screen=6; //Invasion Barrier Reached
                        }}}
        if(cir_alien[i].alien_initial_x_pos>=cir_alien[i].alien_min_x_pos && cir_alien[i].go==0) //Food Is On The Left :D
        {if(cir_alien[i].alien_direction==0)cir_alien[i].alien_initial_x_pos=cir_alien[i].alien_initial_x_pos-((60.0/fps)*0.8);
                if(cir_alien[i].alien_initial_x_pos<=cir_alien[i].alien_min_x_pos)
                {       cir_alien[i].alien_initial_x_pos=cir_alien[i].alien_min_x_pos;
                        cir_alien[i].go=1; //Throw Food To The Right
                        cir_alien[i].alien_direction=!cir_alien[i].alien_direction; //Hamster Changes Direction
                        if(i%2!=0)
                        {cir_alien[i].alien_initial_y_pos=cir_alien[i].alien_initial_y_pos-((60.0/fps)*5.0);
                                if(cir_alien[i].alien_initial_y_pos<=50.0)display_screen=6;
                        }}}}}}
void create_ca_alive_list(void)
{       int i,j;
        for(i=0,j=0,ca_alive_count=0;i<num_cir_aliens;i++) {
        if(cir_alien[i].alien_hitpoints>0){
                        ca_alive[j]=i;
                        j++;
                        ca_alive_count++;}}
        if(ca_alive_count==0)display_screen=5; //WAVE COMPLETED :)
}void ca_bullet_alloc(void){
        ca_timediff=(double)(time2-ca_time1);
```

```
        if(ca_timediff>=ca_bullet_alloc_time_delay && ca_alive_count>0)
        {ca_time1=clock();
alien_that_will_shoot=ca_alive[rand()%ca_alive_count];if(ca_bullet[alien_that_will_shoot].allocated==0)
{                       ca_bullet[alien_that_will_shoot].x_pos=cir_alien[alien_that_will_shoot].alien_initial_x_pos;
                        ca_bullet[alien_that_will_shoot].y_pos=cir_alien[alien_that_will_shoot].alien_initial_y_pos-r;
                        ca_bullet[alien_that_will_shoot].allocated=1;
                }}}
void cir_alien_bullet(void){
        int i;
        glColor3ub(255,255,0);
        for(i=0;i<max_bullets;i++)
        {
        if(ca_bullet[i].allocated==1)
        {ca_bullet[i].y_pos=ca_bullet[i].y_pos-((60.0/fps)*5.0);
                        if(ca_bullet[i].y_pos>0.0)
                        {       glBegin(GL_LINES);
                                glVertex2f(ca_bullet[i].x_pos,ca_bullet[i].y_pos);
                                glVertex2f(ca_bullet[i].x_pos,ca_bullet[i].y_pos-10);
                                glVertex2f(ca_bullet[i].x_pos-1,ca_bullet[i].y_pos);
                                glVertex2f(ca_bullet[i].x_pos-1,ca_bullet[i].y_pos-10);
                                glEnd();
                        }
                        else ca_bullet[i].allocated=0;
                }}}
unsigned char digit_count(unsigned long int num)
{       unsigned char count=1;
        while(num>9)
        {       num=num/10;
                count++}
        return count;}
```

CONCLUSION AND FUTURE SCOPE

CONCLUSION:

The project has been successfully completed but further it could have been enhanced. We learnt a lot of new things and built-in functions while doing the project, which can prove very useful in the software field. By the user"s point of view the OpenGL software is very easy to use and also it is most widely used application in interaction.

We have successfully created Space Invaders, a game using OpenGL . We believe that our game is a unique take on the old arcade shooters of the 70"s and incorporates a fun and challenging shooting mechanic. We faced many challenges during the process but we also had a lot of fun creating the game. Even though almost all of our issues were resolved, if we get a chance to revisit the project again, we would make the game a lot more challenging and try to improve the player"s experience with the game.

FUTURE SCOPE:

- Providing better user experience by implementing 3D models of aircraft and space ships.
- Providing different weapons and aircraft design for more interactive and enjoyable gaming experience.
- Providing more number of levels and stages, newer dynamic background for better visualization .
- Implementing multi-player gaming system to enjoy the game with friends.

REFERENCES

[1] OpenGL Programming Guide (Addis on-Wesley Publishing Company).

[2] The OpenGL Utility Toolkit(GLUT) Programming Interface - API Version 3 By MARK J.KILGARD.

[3] Interactive computer graphics - A top down approach by using OpenGL by EDWARD ANGEL.

[4] Computer Graphics with OpenGL 4th Edition by -DONALD D.HEARN , M.PAULINE BAKER,WARREN CARITHERS.

[5] http://www.opengl.org/registry/

[6] www.openglforum.com

[7] www.w3schools.com

[8]www.hackerrank.com

DECLARATION

We student of 6th semester BE, Computer Science and Engineering College hereby declare that project work entitled “SPACE INVADERS” has been carried out by us at City Engineering College, Bangalore and submitted in partial fulfillment of the course requirement for the award of the degree of Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belgaum, during the academic year 2021-2022.

We also declare that, to the best of our knowledge and belief, the work reported here does not form the part of dissertation on the basis of which a degree or award was conferred on an earlier occasion on this by any other student.

Date:

Place: Bangalore

DEEPAK JADON (1CE19CS027)

FOZAIL AHMED (1CE19CS031)