**CHAPTER 1**

**INTRODUCTION**

**Computer graphics** are graphics created using computers and more generally, the representation and manipulation of image data by a computer. The term computer graphics has been used in a broad sense to describe "almost everything on computers that is not text or sound".

The development of computer graphics has made computers easier to interact with, better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized animation, movies and the video game industry.

**The various applications of computer graphics are**

* Graphs and charts
* Computer-Aided design
* Virtual-Reality environment
* Data Visualization
* Education and Training
* Computer Art
* Entertainment
* Image Processing
* Graphical User interfaces

**Graphs and Charts:**

An early application for computer graphics is the display of simple data graphs, usually plotted on a character printer. Data plotting is still one of the most common graphics applications, but today one can easily generate graphs showing highly complex data relationships for printed reports or for presentations using 35 mm slides, transparencies, or animated videos. Graphs and charts are commonly used to summarize financial, statistical, mathematical, scientific, engineering, and economic data for research reports, managerial summaries, consumer information bulletins, and other types of publications.

**Computer Aided Design**:

A major use of computer graphics is in design processes—particularly for engineering and architectural systems, although most products are now computer designed. Generally referred to as CAD, computer-aided design, or CADD, computer-aided drafting and design, these methods are now routinely used in the design of buildings, automobiles, aircraft, watercraft, spacecraft, computers, textiles, home appliances, and a multitude of other products. The manufacturing process is also tied in to the computer description of designed objects so that the fabrication of a product can be automated, using methods that are referred to as CAM, computer-aided manufacturing.

**Virtual Reality Environment**:

It is a recent application of computer graphics which is used to create virtual-reality environments in which a user can interact with the objects in a three-dimensional scene. Specialized hardware devices provide three-dimensional viewing effects and allow the user to “pick up” objects in a scene. Animations in virtual-reality environments are often used to train heavy equipment operators or to analyze the effectiveness of various cabin configurations and control placements. This allows the designer to explore various positions of the bucket or backhoe that might obstruct the operator’s view, which can then be taken into account in the overall tractor design.

**Data Visualization**:

Producing graphical representations for scientific, engineering, and medical data sets and processes is another fairly new application of computer graphics, which is generally referred to as scientific visualization. The term business visualization is used in connection with data sets related to commerce, industry, and other nonscientific areas. Numerical computer simulations, for example, frequently produce data files containing thousands and even millions of values. Similarly, satellite cameras and other recording sources are amassing large data files faster than they can be interpreted. Other visualization techniques include contour plots, renderings for constant-value surfaces or other spatial regions, and specially designed shapes that are used to represent different data types.

**Education and Training**:

Computer-generated models of physical, financial, political, social, economic, and other systems are often used as educational aids. Models of physical processes, physiological functions, population trends, or equipment, such as the color-coded diagram in for some training applications, special hardware systems are designed. Examples of such specialized systems are the simulators for practice sessions or training of ship captains, aircraft pilots, heavy-equipment operators, and air traffic-control personnel. Some simulators have no video screens; a flight simulator with only a control panel for instrument flying. But most simulators provide screens for visual displays of the external environment with multiple panels is mounted in front of the simulator.

**Entertainment:**

Television productions, motion pictures, and music videos routinely use computer-graphics methods. Sometimes graphics images are combined with live actors and scenes, and sometimes the films are completely generated using computer-rendering and animation techniques. Many TV series regularly employ computer-graphics methods to produce special effects, such as the scene in Figure from the television series Deep Space Nine. Some television programs also use animation techniques to combine computer-generated figures of people, animals, or cartoon characters with the live actors in a scene or to transform an actor’s face into another shape. And many programs employ computer graphics to generate buildings, terrain features, or other backgrounds for a scene.

**Computer Art**:

Both fine art and commercial art make use of computer-graphics methods. Artists now have available a variety of computer methods and tools, including specialized hardware, commercial software packages (such as Lumena), symbolic mathematics programs (such as Mathematica), CAD packages, desktop publishing software, and animation systems that provide facilities for designing object shapes and specifying object motions. Example: use of a paintbrush program that allows an artist to “paint” pictures on the screen of a video monitor. A paintbrush system, with a Wacom cordless, pressure-sensitive stylus, was used to produce the electronic painting. The stylus translates changing hand pressure into variable line widths, brush sizes, and color gradations.

**Image Processing**:

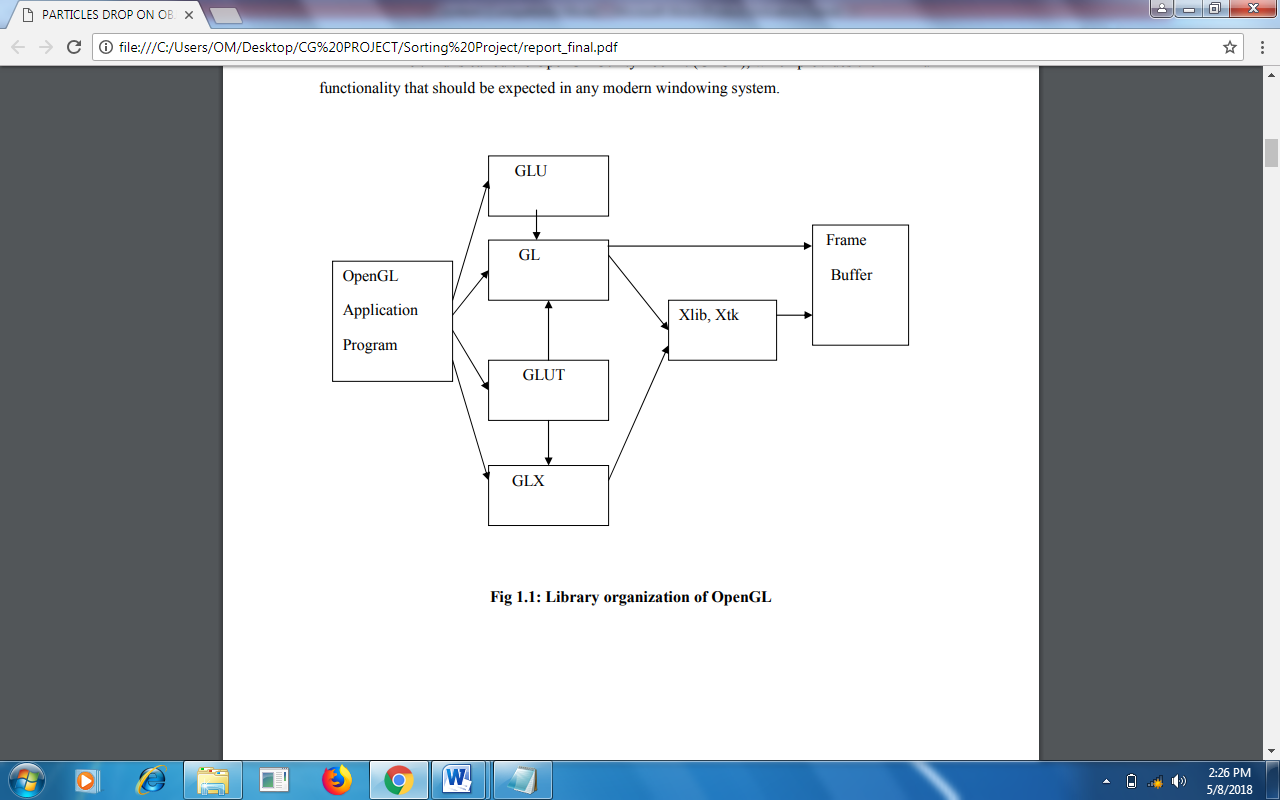
The modification or interpretation of existing pictures, such as photographs and TV scans, is called image processing. In computer graphics, a computer is used to create a picture. Image-processing techniques, on the other hand, are used to improve picture quality, analyze images, or recognize visual patterns for robotics applications. However, image-processing methods are often used in computer graphics, and computer-graphics methods are frequently applied in image processing. Typically, a photograph or other picture is digitized into an image file before image-processing methods are employed. Then digital methods can be used to rearrange picture parts, to enhance color separations, or to improve the quality of shading, OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce [2D](http://en.wikipedia.org/wiki/2D_computer_graphics) 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.](http://en.wikipedia.org/wiki/Silicon_Graphics) (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation .

**OpenGL**

OpenGL has become a widely accepted standard for developing graphics application. Most of our applications will be designed to access OpenGL directly through functions in three libraries. Functions in main GL library 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 begin with the letters ‘glu’.

The third is called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.



**Fig 1.1: Library organization of OpenGL**

* 1. **Overview of the project**

This project is a simple game as described above using openGL. The game involves shooting of arrows one after the other in order to hit the balls which are moving at a constant speed in the right end.

**The various concepts used in this application are as follows:**

**Menus:**

OpenGL supports simple cascading pop-up menus. In this application simple pop-up menus along with the submenu are used to provide a simple user interface. The menu will pop up by clicking the right mouse button.

**Blending:**

The project uses blending to make objects appear transparent.

**Transformations:**

A transformation is a function that takes a point or vector and maps that point or vector into another point or vector. In this application the transformation is used to make the plane to rotate in all 3 axes.

**Bitmap Character:**

OpenGL supports two kinds of texts: bitmap character and stroke character. Bitmap characters are basically 2D font without thickness. In this application bitmap character is used to display the texts.

**The main features of the project are as follows**

* The application is user friendly, thereby it can be used by children and who do not know computer in depth too.
* As provision for future development.
* It has many options of keyboard functions.
* It has a good visual effect.
  1. **Aim of the project**

This project is a simple game as described above using openGL. The game involves shooting of arrows one after the other in order to hit the balls which are moving at a constant speed in the right end. This game has been designed in a simple and lucid manner so that further developments can be made, and run on many platforms with a few changes in the code.

User has to shoot the balls by pressing the key 'f' on the keyboard..

**CHAPTER 2**

**REQUIREMENT SPECIFICATION**

A software requirement definition is an abstract description of the services which the system should provide, and the constraints under which the system must operate. It should only specify the external behavior of the system.

**2.1 Functional Requirements**

In software engineering, a **functional requirement** defines a function of a software system or its component. A function is described as a set of inputs, the behavior, and outputs (see also software). Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define *what* a system is supposed to accomplish. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases.

**The various methods used in this project are as follows:-**

* **Display**

The module draws the output on the screen and the functions in it.

* **Menu**

This module specifies the action corresponding to menu entry.

* **Keyboard**

The module specifies the action corresponding to the keyboard.

**2.2 Non-functional requirements:**

These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process and standards. Non-functional requirements often apply to the system as a whole.

**Non-Functional Requirements are as follows:-**

**2.2.1 Dependability:**

The dependability of a computer system is a property of the system that equates to its trustworthiness. Trustworthiness essentially means the degree of user confidence that the system will operate as they expect and that the system will not ‘fail’ in normal use.

**2.2.2 Availability:**

The ability of the system is to deliver services when requested. There is no error in the program while executing the program.

**2.2.3 Reliability:**

The ability of the system to deliver services as specified. The program is compatible with all types of operating system without any failure.

**2.2.4 Safety:**

The ability of the system is to operate without catastrophic failure. This program is user friendly and it will never affect the system.

**2.2.5 Security:**

The ability of the system is to protect itself against accidental or deliberate intrusion.

**2.3 Details of the software**

Here, the coding of our project is done in Microsoft Visual C++ which is a commercial integrated development environment (IDE) with OpenGL (Open Graphics Library) which is a standard specification to produce 2D and 3D computer graphics. We use, the OpenGL Utility Toolkit called GLUT which is a library of utilities for OpenGL programs.

**2.3.1 Microsoft Visual C++**

Microsoft Visual C++ is a commercial [integrated development environment](http://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) product engineered by [Microsoft](http://en.wikipedia.org/wiki/Microsoft) for the [C](http://en.wikipedia.org/wiki/C_(programming_language)), [C++](http://en.wikipedia.org/wiki/C%2B%2B) and [C++/CLI](http://en.wikipedia.org/wiki/C%2B%2B/CLI) [programming languages](http://en.wikipedia.org/wiki/Programming_language). It has tools for [developing](http://en.wikipedia.org/wiki/Software_development_process) and [debugging](http://en.wikipedia.org/wiki/Debugging) C++ code, especially code written for the [Microsoft Windows](http://en.wikipedia.org/wiki/Microsoft_Windows)[API](http://en.wikipedia.org/wiki/Application_programming_interface), OpenGL API, the [DirectX](http://en.wikipedia.org/wiki/DirectX)[API](http://en.wikipedia.org/wiki/API) and the [Microsoft .NET](http://en.wikipedia.org/wiki/Microsoft_.NET) Framework.

**2.3.2 OpenGL and GLUT**

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, [cross-platform](http://en.wikipedia.org/wiki/Cross-platform) [API](http://en.wikipedia.org/wiki/Application_programming_interface) for writing applications that produce [2D](http://en.wikipedia.org/wiki/2D_computer_graphics) and [3D computer graphics](http://en.wikipedia.org/wiki/3D_computer_graphics), describing a set of functions and the precise behaviors that they must perform. From this specification, hardware vendors create implementations - libraries of functions created to match the functions stated in the OpenGL specification, making use of hardware acceleration where possible. Hardware vendors have to meet specific tests to be able to qualify their implementation as an OpenGL implementation.

GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API)for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms.

**2.4 Software Requirements**

* OPERATING SYSTEM **:** Windows 98, Windows XP, Windows Vista,

Windows 7/8/10

* FRONT END **:** Microsoft Visual Studio 2010 and above
* CODING LANGUAGE **:** C++

**2.5 Hardware Requirements**

* SYSTEM : Pentium IV 2.4 GHz or above
* HARD DISK : 40 GB, 80 GB, 160 GB or above
* MONITOR : 15 VGA color
* RAM : 1 GB or above

**CHAPTER 3**

**SOFTWARE DESIGN**

This project has been created using the OpenGL interface along with the GLUT (OpenGL Library Toolkit), using the platform Eclipse as the compiling tool. This game has been designed in a simple and lucid manner so that further developments can be made, and run on many platforms with a few changes in the code.

We begin our game by providing a simple menu option upon right click of the mouse to display 2 levels of the game which the user can choose, and a third option to exit the game.

Upon selecting the first level, a red ball is continuously produced and moves upwards at the right side of the screen. Hitting the ‘f’ key on the keyboard produces an arrow, which the user must aim to hit the balls with. On successful contact with the ball, the game registers it as a ‘HIT’ and increments a counter variable. A count of 3 (i.e. 3 hits) takes the player to the next level of the game.

Level 2 increases the difficulty level by producing2 colours of balls-a blue one and a red one. While the red is as before produced continuously and moves from bottom to top, the blue one moves from top to bottom at irregular intervals to confuse the player. The player needs to aim correctly and target only the red ones. Player needs to again score a minimum of 3 ‘hits’ to be a Winner.

Level 2 can be directly entered in the beginning itself (without playing level 1). We provide this option so that the player can choose his difficulty level.

The underlying logic of the game is Collision detection. The bounds of the ball are detected with the bounds, or the continuously changing position of the arrow head. 2 variables ‘pos’ (for the arrowhead) and ‘up’ (for the ball) are kept tab of, and are updated with their changing values.

If the given position vector does intersect with the current value of the ‘up’, we infer that a collision has been detected and register it as a hit, and increment ‘counter’. Otherwise the function returns a NULL.

Once a ‘hit’ takes place, we call a Sleep() timer ( as buffers are too fast to see the collision in our game), show the ball burst ( without particle effect or engines) and reset the ball to its initial position and the cycle starts all over again.

Careful observations of these values have been made while developing the game.

To model the objects as 3D entities we also use the effects of lighting in our game via 4 different mechanisms:

* **AMBIENT** -light that comes from all directions equally and is scattered in all directions equally by the polygon.
* **DIFFUSE** -light that comes from a point source and hits surfaces with an intensity that depends on whether they face towards the light or away from it. SPECULAR -light is reflected more in the manner of a mirror where most of the light bounces off in a particular direction defined by the surface shape.
* **EMISSION** -the light is actually emitted by the polygon -equally in all directions.

Texture is yet another feature we have implemented to show the material properties of the object, in whichever small way we could.

**CHAPTER 4**

**IMPLEMENTATION**

To implement the Current system we have used different functions of our project which are as follows:

* 1. **USER DEFINED FUNCTIONS**
* **Choose():**

This function gives a switch case to choose from the different menu options. Choosing case 2 (Level 1) calls the function Display1() and choosing case 3 (Level 2) calls Display2(). Choosing quit case 1 will exit from the game at any point in the game.

* **Display():**

Sets the background colour and clears the color buffer bit, and depth buffer bit (for hidden surface removal and Z buffer test), and it is called from the main().

* **Display1():**

It plays the Level 1 of the game. It initially sets the red ball to a position given by ‘up’ variable, and using the Translatef() function it displaces only the y-coordinate of the ball upwards.

glutSolidSphere() renders a 3D sphere with radius, slices and stack as parameters.

While the ‘f’ key is pressed, the function creates an arrow head, and associates a variable ‘pos’ with it, to translate the arrow towards the right in a single direction. This variable is incremented continually every time and called with the Translatef() to redraw the arrow at new positions.

If the condition for bounds satisfy, that means collision has occurred, and counter1 is incremented to register a hit. The flag ‘bang’ is set to 1, so that when encountered during the next iteration the following changes can take place: position of sphere is reset to 0, bang is reset to 0 (to prepare for next hit), and position of arrow is reset.

The counter1 value is checked every iteration. Once it has reached a value of 3, Display2() is called to play Level 2 of the game.

‘up’ variable is continually incremented to keep the ball moving upwards for a large number of iterations by calling glutPostRedisplay() every time. It marks the normal plane of current window as needing to be redisplayed with the same specifications.

* **Display2():**

It enters the Level 2 of the game. It initially sets the red ball to a position given by ‘up’ variable, and using the Translatef() function it displaces only the y-coordinate of the ball upwards.

Also, it sets the blue balls position and using the Translatef() function it displaces only the y-coordinate of the ball downwards.

glutSolidSphere() renders a 3D sphere with radius, slices and stack as parameters.

While the ‘f’ key is pressed, the function creates an arrow head, and associates a variable ‘pos’ with it, to translate the arrow towards the right in a single direction. This variable is incremented continually every time and called with the Translatef() to redraw the arrow at new positions.

If the condition for bounds satisfy, that means collision has occurred, and counter1 is incremented to register a hit. The flag ‘bang’ is set to 1, so that when encountered during the next iteration ,the following changes can take place: position of sphere is reset to 0, bang is reset to 0 (to prepare for next hit), position of arrow is reset.

‘up’ variable is continually incremented to keep the ball moving upwards for a large number of iterations by calling glutPostRedisplay() every time. It marks the normal plane of current window as needing to be redisplayed with the same specifications.

The counter2 value is checked upon every iteration. Once it has reached a value of 3, myHit() is called to display that the player is a winner.

* **myHit():**

This function is primarily to display text on screen. glBlendFunc() defines the operation of blending when it is enabled. We also set the antialiasing for lines and set the line width to prepare to draw our text as Stroke Characters.

* **DrawHit():**

It draws text in Stroke Character font. A stroke font is a 3D font. As opposed to bitmap fonts these can be rotated, scaled, and translated. The GLUT\_STROKE\_ROMAN font is used here. The text can be placed anywhere on the screen using Translatef() and scaled to any size as needed using Scalef().

* **Instructions():**

It creates a separate instruction page where the instructions are displayed on another window using Stroke Character font, and translated to appropriate positions on the screen.

* **Draw\_instruct():**

It draws text in Stroke Character font. A stroke font is a 3D font. As opposed to bitmap fonts these can be rotated, scaled, and translated. The GLUT\_STROKE\_ROMAN font is used here. The text can be placed anywhere on the screen using Translatef() and scaled to any size as needed using Scalef().

* **Keyboard():**

Keyboard is a keyboard call-back function which is used to make our program interactive. It makes the shoot flag = 1 in our program every time key ‘f’ is pressed, by recognizing it as an ASCII character.

* **Main():**

The main function performs the required initializations and starts the event processing loop. All the functions in GLUT have the prefix *glut*, and those which perform some kind of initialization have the prefix *glutInit*.

glutInit(int \*argc, char \*\*argv) -parameters are pointers to the *unmodified*argc and argv variables from the main function.

We establish the window's position by using the function *glutInitWindowPosition*.

We choose the window size using the function *glutInitWindowSize*.

We define the display mode using the function *glutInitDisplayMode*. GLUT\_RGB -selects a RGBA window. This is the default color mode. GLUT\_SINGLE –selects a single buffer window.

Each window can be created with *glutCreateWindow*. The returns value of *glutCreateWindow*is the window identifier.

glutDisplayFunc() passes the name of the function to be called when the window needs to be redrawn.

glutKeyboardFunc-is notify the windows system which function(s) will perform the required processing when a key is pressed. This function is to register a call-back for keyboard events that occur when you press a key.

Creating a menu: glutCreateMenu creates a menu table on a default right click of mouse. glutAddMenuEntry adds a menu entry to this menu created.

When we are ready to get in the application event processing loop we enter glutMainLoop. It gets the application in a never ending loop, always waiting for the next event to process.

* **Init():**

Sets the background color for the game and enables light source to provide the following lighting effects:

* AMBIENT -light that comes from all directions equally and is scattered in all directions equally by the polygon.
* DIFFUSE -light that comes from a point source and hits surfaces with an intensity that depends on whether they face towards the light or away from it.
* SPECULAR -light is reflected more in the manner of a mirror where most of the light bounces off in a particular direction defined by the surface shape.
* EMISSION -the light is actually emitted by the polygon -equally in all directions

It also sets the Material for the object and enables this option to provide different surface textures to our objects.

**4.2 BUILT IN FUNCTIONS**

* **void glClear (GLbitfield*mask*);**
* ***mask*** *–* Bitwise OR of masks that indicate the buffers to be cleared. The four masks are GL\_COLOR\_BUFFER\_BIT, GL\_DEPTH\_BUFFER\_BIT, GL\_ACCUM\_BUFFER\_BIT and GL\_STENCIL\_BUFFER\_BIT.
* It clears buffers to preset values.
* **Void glClearColor (GLclampf*red*, GLclampf*green*, GLclampf*blue,* GLclampf*alpha*);**
* *red, green, blue, alpha* – specify the red, green, blue and alpha values used when the color buffers are cleared. The initial values are all 0.
* It specifies clear values for the color buffers.
* **void glColor3f (GLfloat*red*, GLfloat*green,* GLfloat*blue*);**
* *red, green, blue –* specify new red, green, and blue values for the current color.
* It sets the current color.
* **glutCreateWindow (char \**name*);**
* *name –* ASCII character string for use as window name.
* It creates a top-level window.
* **Void glutDisplayFunc (void (*\*func*) (void));**
* *func* – the new display callback function.
* It sets the display callback for the current window.
* **Void glutInitWindowSize (int*width,* int*height*);**
* width– width in pixels.
* height – height in pixels.
* It is used to set the initial window size.
* **Void glutMainLoop (void);**
* It enters the GLUT event processing group. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.
* **Void glutPostRedisplay (void);**
* It marks the current window as needing to be redisplayed.
* **void display (void);**
* It contains the function definition for display callback.
* **Void glutIdleFunc (void (\**func*) (void));**
* It sets the global idle callback.

# CHAPTER 5

# TESTING

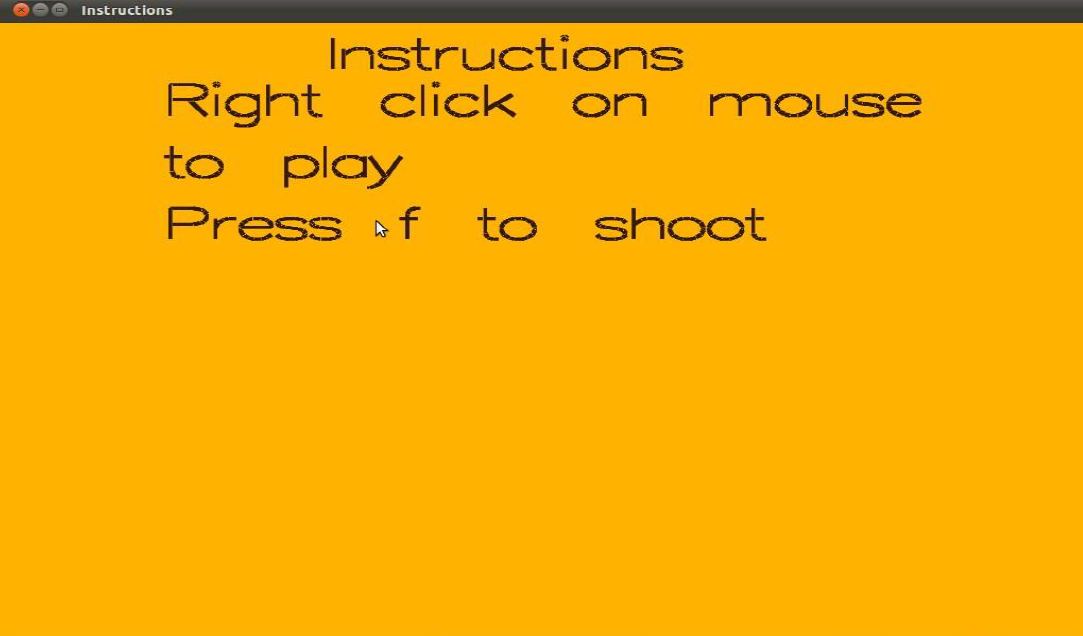
Testing has been conducted as tabulated below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Functions with parameters under test** | **Expected result** | **Actual result** | **Comments** |
| 1 | All the output statement must be in there position mentioned | The object and particle drop should appear at the window | The object and particle appeared. | PASS |
| 2 | Choose function | This function gives a switch case to choose from the different menu options. | Choosing case 2 (Level 1) calls the function Display1() and choosing case 3 (Level 2) calls Display2(). Choosing quit case 1 will exit from the game at any point in the game. | PASS |
| 4 | Instructions Function | It creates a separate instruction page where the instructions are displayed on another window | The keys work properly and instructions are shown. | PASS |
| 5 | Display Function | This function has count of arrows present and the score occurred for Level-1. | The arrow count present and score updated. | PASS |
| 6 | Display2 Function | This function has count of arrows present and the score occurred for Level-2. | The arrow count present and score updated. | PASS |
| 7 | Keyboard Function | ‘f’ key needs to be pressed to shoot. | The arrow shoots on pressing ‘f’ key. | PASS |

**CHAPTER 6**

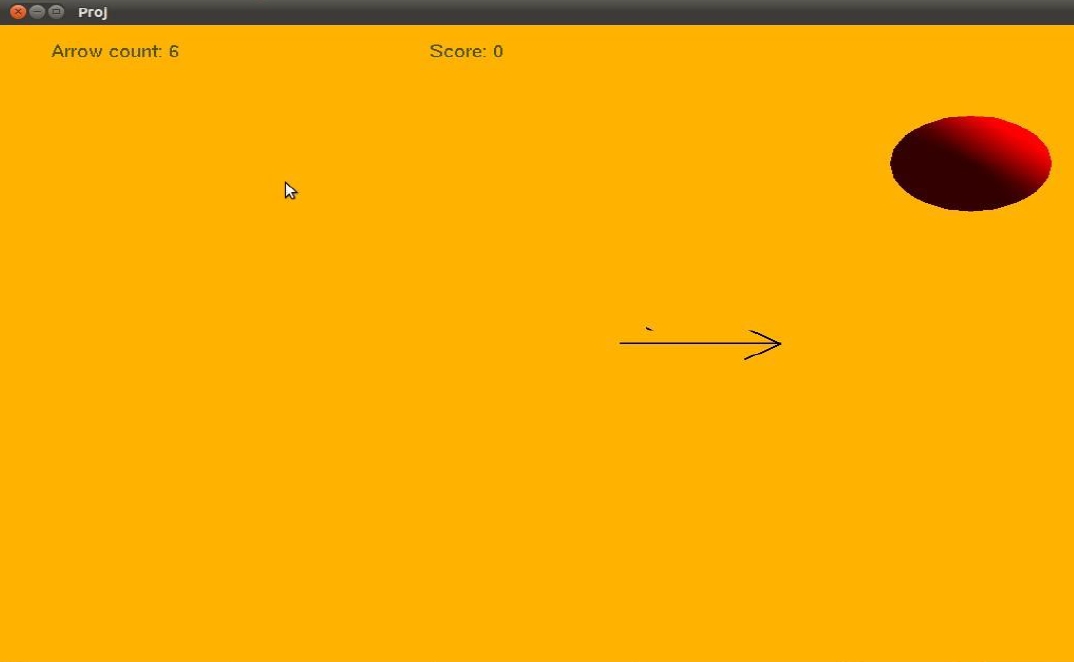
**SNAPSHOTS**

* 1. **Instructions Page**



**Figure 6.1:Instructions Page**

* 1. **Game Round-1**



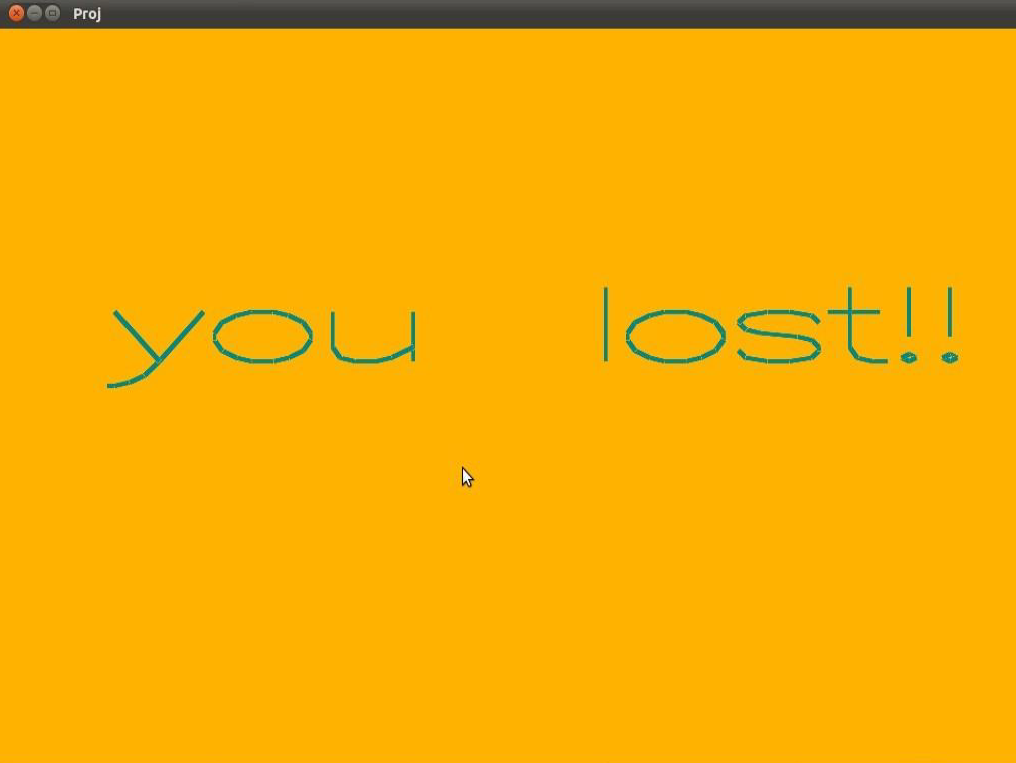
**Figure 6.2: Game Round-1 Page**

* 1. **Game Round-2**



**Figure 6.3: Game Round-2 Page**

* 1. **Game Lost**



**Figure 6.4: Game lost Page**

* 1. **Yee…Won The Game**



**Figure 6.5: Game Won Page**

**CHAPTER 7**

**CONCLUSION**

The development of the project is not an easy process as it involves lot of challenges in different stages of software analysis, design, coding and testing.

Having understood the requirements properly and implementing the solutions as per the expectation as brought to the closure of the project. We have tried our best to make this project very realistic, so that the user does not face any trouble when switching over from any real life graphics project to this highly useful one.

The development of computer graphics has made computers easier to interact with and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry.

We started with modest aim with no prior experience in any programming projects as this, but ended up in learning many things, fine tuning the programming skills and getting into the real world of software development with an exposure to corporate environment. During the development of any software of significant utility, we are faced with the trade-off between speed of execution and amount of memory consumed. This is simple interactive application. It is extremely user friendly and has the features, which makes simple graphics project. It has an open source code and no security features has been included. The user is free to alter the code for feature enhancement.

Checking and verification of all possible types of the functions are taken care. Care was taken to avoid bugs. Bugs may be reported to creator as the need may be .

So, we conclude on note that we are looking forward to develop more such projects with an appetite to learn more in computer graphics.

**BIBILOGRAPHY**

**Reference Books**

[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, 2008

[3] James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer graphics

with OpenGL: pearson education

**Websites**

* www.opengl.org
* www.google.co.in(OpenGLprojects)
* [www.wikipedia.com](http://www.wikipedia.com)