CHAPTER 1

INTRODUCTION

1.1 OPENGL

OpenGL is the abbreviation for Open Graphics Library. It is a software interface for graphics hardware. This interface consists of several hundred functions that allow you, a graphics programmer, to specify the objects and operations needed to produce high-quality color images of two-dimensional and three-dimensional objects. Many of these functions are actually simple variations of each other, so in reality there are about 120 substantially different functions. The main purpose of OpenGL is to render two-dimensional and three-dimensional objects into the frame buffer. These objects are defined as sequences of vertices (that define geometric objects) or pixels (that define images). OpenGL performs several processes on this data to convert it to pixels to form the final desired image in the frame buffer.

1.2 HISTORY

As a result, SGI released the OpenGL standard in the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. Software developers wrote custom interfaces and drivers for each piece of hardware. This was expensive and resulted in much duplication of effort. By the early 1990s, Silicon Graphics (SGI) was a leader in 3D graphics for workstations. Their IRIS GL API was considered the state of the art and became the de facto industry standard, overshadowing the open standards-based PHIGS. This was because IRIS GL was considered easier to use, and because it supported immediate mode rendering. By contrast, PHIGS was considered difficult to use and outdated in terms of functionality.

SGI's competitors (including Sun Microsystems, Hewlett-Packard and IBM) were also able to bring to market 3D hardware, supported by extensions made to the PHIGS standard. This in turn caused SGI market share to weaken as more 3D graphics hardware suppliers entered the market. In an effort to influence the market, SGI decided to turn the Iris GL API into an open standard.
SGI considered that the Iris GL API itself wasn't suitable for opening due tolicensing and patent issues. Also, the Iris GL had API functions that were not relevant to 3D graphics. For example, it included a windowing, keyboard and mouse API, in part because it was developed before the X

Window System and Sun's NEWS systems were developed.

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. Meanwhile, SGI would continue to try to maintain their customers tied to SGI hardware by developing the advanced and proprietary Iris Inventor and Iris Performer programming APIs.

1.3 FEATURES OF OPENGL

• **Industry standard**

An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL Is the only truly open, vendor-neutral, multiplatform graphics standard.

• Stable

OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

• Reliable and portable

All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.

• Evolving

Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers.

• **Scalable**

OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

• **Easy to use**

OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.

• Well-documented

Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

1.4 LIBRARIES

Most of our applications will be designed to access OpenGL directly through functions in three libraries.

They are:

• GL – Graphics Library

Functions in the main GL (or OpenGL in Windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in Windows).

• **GLU – Graphics Utility Library**

This library uses only GL functions but contain code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begins with the letters glu.

• GLUT – OpenGL Utility Toolkit

To interface with the window system and to get input from external devices into our programs we need at least one more library. For the X window System, this library is called GLX, for Windows, it is wgl, and for the Macintosh, it is agl. Rather than using a different library for each system, we use a readily available library called the OpenGL Utility Toolkit (GLUT), which provides minimum functionality that should be expected in any modern windowing system.

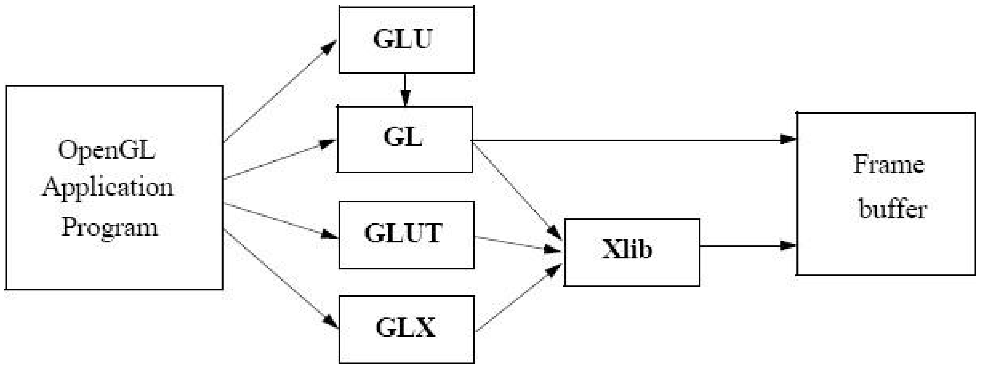


Fig: 1.5 Library Organization

The above figure shows the organization of the libraries for an X Window
System environment.

#include&lt;GL/glut.h&gt; is sufficient to read in glut.h, gl.h and glu.h.

1.5 GRAPHICS FUNCTIONS

Our basic model of a graphics package is a black box, a term that engineers use to denote a system whose properties are described only by its inputs and outputs; we may know nothing about its internal workings. OpenGL can be classified into seven major groups:

• Primitive function

The primitive functions define the low-level objects or atomic entities that our system can display. Depending on the API, the primitives can include points, lines, polygons, pixels, text, and various types of curves and surfaces.

• **Attribute functions**

If primitives are the what of an API – the primitive objects that can be displayed- then attributes are the how. That is, the attributes govern the way the primitive appears on the display. Attribute functions allow us to perform operations ranging from choosing the color with which we display a line segment, to picking a pattern with which to fill inside of a polygon.

• Viewing functions

The viewing functions allow us to specify various views, although APIs differ in the degree of flexibility, they provide in choosing a view.

• Transformation functions

One of the characteristics of a good API is that it provides the user with a set of transformations functions such as rotation, translation and scaling.

• Input functions

For interactive applications, an API must provide a set of input functions, to allow users to deal with the diverse forms of input that characterize modern graphics systems. We need functions to deal with devices such as keyboards, mice and data tablets.

• Control functions

These functions enable us to communicate with the window system, to initialize our programs, and to deal with any errors that take place during the execution of our programs.

• Query functions

If we are to write device independent programs, we should expect the implementation of the API to take care of the differences between devices, such as how many colors are supported or the size of the display. Such information of the particular implementation should be provided through aset of query functions.

CHAPTER 2

REQUIREMENT ANALYSIS

2.1 DEFINITION

A software requirements specification (SRS) is a description of a software system to be developed. Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers on how the software product should function (in a market driven project, these roles may be played by the marketing and development divisions). Software requirements specification is a rigorous assessment of requirements before the more specific system design stages, and its goal is to reduce later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules. The SRS is divided into functional and non-functional requirements.

2.2 NON-FUNCTIONAL REQUIREMENTS

In systems engineering and requirements engineering, a non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. They are contrasted with functional requirements that define specific behaviour or functions. The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture, because they are usually architecturally significant requirements.

2.3 FUNCTIONAL REQUIREMENTS

The functional requirements are the requirements which are needed to develop the software application. The functional requirement are broadly classified into 2 categories, they are,

1. Hardware Requirements
2. Software Requirements

2.3.1 HARDWARE REQUIREMENTS

- Computer components with Graphics enabled
- Mouse, Keyboard and computer accessories
- 500 Mb space in the HDD
- LCD Screen

2.3.2 SOFTWARE REQUIREMENT

- Operating System: Mac OS/ Ubuntu / Windows 7 or higher
- Programming language: C
- Graphics library Used: OpenGL
- IDE Used: VS CODE
- Compiler Used: GCC

CHAPTER 3

DESIGN

3.1 ALGORITHM

An algorithm is a step-by-step instruction to execute the program. The algorithm of our project is shown below.

STEP 1: Initialize the graphics library.
STEP 2: Display the starting page.
STEP 3: Take the user input.

```
if input == A
slide2();
else if input == S
slide3();
else if input == D
slide4();
else if input == F
slide5();
else if input == G
slide6();
else if input == H
slide7();
else
exit();
end if
```

Fig 3.1 Algorithm

3.2 FLOW CHART

Flowchart is a graphical representation of a computer program in relation to its sequence of Functions . The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields. Fig.3.2 gives a visual representation of the sequence of steps and decisions needed to perform a process in the proposed project.

In the fig 3.2 each rectangular box represents a scene. The first scene is the content display page which includes name of the project, college name and etc. Next scene is the description page which gives look of the bank of the project. On clicking the Key A from Keyboard first scenario of the project is where the bank is can be seen.. Then by clicking Key S next page that shows us that a person entering the bank with a piggy bank. On clicking the Key D inside bank scene can be seen where the receptionist is happy by seeing that person. On clicking the Key F the person keeps the piggy bank on the receptionist table and breaks it with the hammer. On clicking on the Key G we can see that the person turns out to the robber and the robber points the gun towards receptionist. On clicking on the Key H we can observe that the robber have looted the entire bank and is running away with the money.

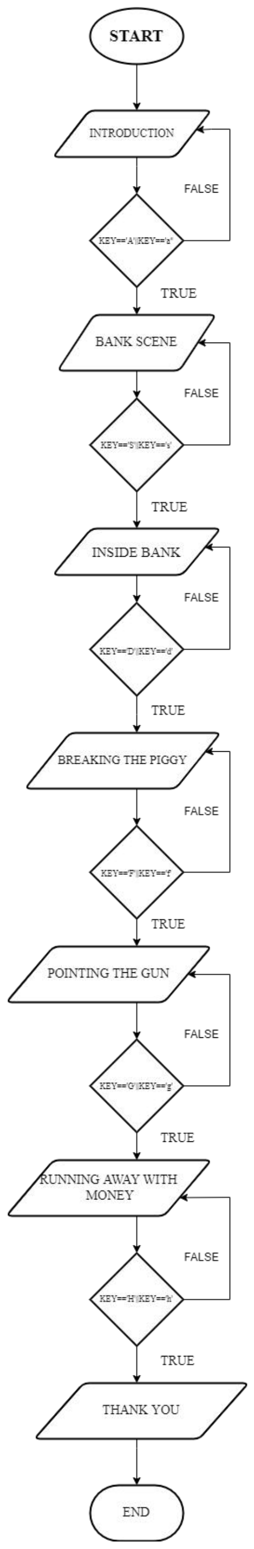


Fig 3.2 : FLOW CHART

CHAPTER 4
IMPLEMENTATION

4.1 OVERVIEW

This project is a demonstration of "**INNOCENT ROBBER**". We have taken the help of built-in functions present in the header file. To provide functionality to our project we have written sub functions. These functions provide us the efficient way to design the project. In this chapter we are describing the functionality of our project using these functions. Keyboard interactions are provided where, when a Enter button is pressed, menu displays and we can select options from menu displayed.

4.2 BUILT-IN FUNCTIONS

• void glColor3f(GLfloat red, GLfloat green, GLfloat blue): This function sets present RGB colours. Different colour is given to object using the colours parameters such as red, green, blue. The maximum and minimum values of the floating-point types are 1.0 and 0.0 respectively.

• void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far): Defines the orthographic volume with all the parameters measured from the centre of projection of the plane.

• void LoadIdentity(): Sets the current transformation matrix to an identity matrix.

• void glClear(GL_COLOR_BUFFER_BIT): Clears all buffer whose bits are set in mask. The mask is formed by the logical OR of values defined in gl.h GL_COLOR_BUFFER_BIT refers to colour buffers.

• void glBegin( ) and void glEnd( ): The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives. Syntax of glBegin is as follows: glBegin(GLenum mode);

• void glFlush(void): The glFlush function forces execution of OpenGL function in finite time. This function has no parameters. This function does not return a value.

• void MatrixMode(GL_PROJECTION): Projection matrixes are stored inOpenGL projection mode. So to set the projection transformation matrix. That mode is invoked through the statement, glMatrixMode(GL_PROJECTION);

• void glutDisplayFunc(): Sets the display call back for the current window.

• void glutPostRedisplay(): Mark the normal plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display call back will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display call back opportunity generates only a single redisplay call back. GlutPostRedisplay may be called within a window's display or overlay display call back to remark that windows for redisplay.

• void glutInit(int *argc, char **argv): The glutInit will initialize the GLUT library, the arguments form main are passed in and can used by the application. This will negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to user if GLUT cannot be properly initialized.

• void glutInitDisplayMode(unsigned int mode): The glutInitDisplayMode sets the initial display mode. It requests a display with the properties in mode. The value of mode is determined by the logical OR of option including the colour model(GLUT_RGB), buffering (GLUT_DOUBLE) and (GLUT_DEPTH).

• void glutInitWindowSize(int width, int height): This function specifies the initial height and width of the width in pixels.

• void glutInitWindowPosition(int x, int y): This function specifies the initial position of the top-left corner of the windows in pixels.

• void glutCreateWindow(char *title): The glutCreateWindow create a window on the display. The string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

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

• void glutKeyboardFunc(void(*func)(unsigned char key,int x,int y)): This function is used to tell the windows system which function we want to process the normal key even

4.3 USER-DEFINED FUNCTIONS

- **key():** This function is used to add the keyboard interactions to the project
- **intro():**In this function the introduction of the project is displayed
- **slide1():**This function is used to draw the outer bank region and bank.
- **robber():**Using this function we have written the code for robber.
- **door():**Using this function the code for door is written.
- **insideBank():**Using this function the inside of the bank code is written.
- **star():**Using this function code for the scenario of breaking the piggy bank is written.
- **Hammer():**using this function the code for hammer is written.
- **insideBank2:**Using this function the code for inside the bank in written.
- **Last():**In this function robber running away with money is displayed.

4.4 CODE SNIPPETS

```
void key(unsigned char key,int x,int y)
{
        if(key=='A' || key=='a')
        {
                slide=2;
                glutPostRedisplay();
        }
        if(key=='S' || key=='s')
        {
                slide=3;
                glutPostRedisplay();
        }
        if(key=='D' || key=='d')
        {
                slide=4;
                glutPostRedisplay();
        }
        if(key=='F'|| key=='f')
        {
                slide=5;
                glutPostRedisplay();
        }
        if(key=='G'|| key=='g')
        {
                slide=6;
                glutPostRedisplay();
        }
        if(key=='H'|| key=='h')
        {
                slide=7;
```

```
                    glutPostRedisplay();
            }


    }
```

Fig 4.4.1: Code Snippet of Keyboard Function

This Code Snippet is used to provide Keyboard Function where we will provide different keys for different Slides.

```
void outpu(char *s1)
{
int k1;
for(k1=0;k1<strlen(s1);k1++)
{
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,s1[k1]);
}
}
```

Fig 4.4.2: Code Snippet of Bitmap Character

This function is used to print the string on the display window. The string should be written within the output() function and axis can be mentioned to display the String.

```
void display()
{

        if(slide==2)
        {
                slide1();
                glutTimerFunc(50 ,timer0 , 0);
                glutPostRedisplay();
        }
        if(slide==3)
        {
                bank();
                glutTimerFunc(1000 , timer , 0);
                glutPostRedisplay();
        }
        if(slide==4){
                insideBank();
                glutTimerFunc(500 , timer1 , 0);
                glutPostRedisplay();
        }
        if(slide==5){
                hammer();
                glutTimerFunc(500 , timer2 , 0);
                glutPostRedisplay();
        }
        if(slide==6){
                insideBank2();
                glutTimerFunc(500 , timer3 , 0);
                glutPostRedisplay();
        }
```

```
                if(slide==7){
                        last();
                        glutTimerFunc(2000 ,timer4 , 0);
                        glutTimerFunc(500 ,timer5 , 0);
                        glutPostRedisplay();
                }


        }
```

Fig 4.4.3: Code Snippet of Display Function

The display Function is used to call the different slides based on the user Input. The object will be drawn when we call display Function.

```
void door()
{
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glColor3f(0.53,0.72,0.70);
        glBegin(GL_POLYGON);
        glPushMatrix();
        glVertex2f(50+d,22);
        glVertex2f(50+d,73.5);
        glVertex2f(58,73.5);
        glVertex2i(58,22);
        glPopMatrix();
        glEnd();
        robber();
        glColor3f(0.53,0.72,0.70);
        glBegin(GL_POLYGON);
        glPushMatrix();
        glVertex2f(42,22);
        glVertex2f(42,73.5);
        glVertex2f(50-d,73.5);
        glVertex2i(50-d ,22);
        glTranslatef(15,0,0);
        glPopMatrix();
        glEnd();
}
```

Fig 4.4.4: Code Snippet for door

This function will display the door , along with we have called the robber function in which the robber code is written.

```
void star()
{
        glClearColor(0,0,0,0.0);
        glClear(GL_COLOR_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glPushMatrix();
        glTranslatef(50, 49.9 , 0);
        glScalef(1,1,0);
        glColor3f(1,0,0);
        glutSolidSphere(24, 80 , 40);
        glPopMatrix();
        //Outer triangle
        glColor3f(1,0,0);
        glBegin(GL_POLYGON);
        glVertex2f(15,65);
        glVertex2f(80,10);
        glVertex2f(65,65);
        glEnd();
        glColor3f(1,0,0);
        glBegin(GL_POLYGON);
        glVertex2f(30,85);
        glVertex2f(40,10);
        glVertex2f(65,55);
        glEnd();
        glColor3f(1,0,0);
        glBegin(GL_POLYGON);
        glVertex2f(90,55);
        glVertex2f(56,35);
        glVertex2f(48,72);
        glEnd();
```

```
glColor3f(1,0,0);
glBegin(GL_POLYGON);
glVertex2i(50,92);
glVertex2i(57,72);
glVertex2i(43,72);
glEnd();
glColor3f(1,0,0);
glBegin(GL_POLYGON);
glVertex2i(70,86);
glVertex2i(54,68);
glVertex2i(68,52);
glEnd();
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glPushMatrix();
glTranslatef(50, 49.9 , 0);
glScalef(1,1,0);
glColor3f(.98,0.91,.11);
glutSolidSphere(22, 80 , 40);
glPopMatrix();
//innertriangle
glColor3f(.98,0.91,.11);
glBegin(GL_POLYGON);
glVertex2f(69.5,85);
glVertex2f(56,68);
glVertex2f(67,52);
glEnd();
glColor3f(.98,0.91,.11);
glBegin(GL_POLYGON);
glVertex2i(50,90);
glVertex2i(56,68);
```

```
          glVertex2i(44,68);
          glEnd();
          glColor3f(.98,0.91,.11);
          glBegin(GL_POLYGON);
          glVertex2f(89.2,54.8);
          glVertex2f(46,68);
          glVertex2f(57,40);
          glEnd();
          glColor3f(.98,0.91,.11);
          glBegin(GL_POLYGON);
          glVertex2f(30.8,83.5);
          glVertex2f(40.4,12.5);
          glVertex2f(65,55);
          glEnd();
          glColor3f(.98,0.91,.11);
          glBegin(GL_POLYGON);
          glVertex2f(15.2,66.5);
          glVertex2f(79.5,11);
          glVertex2f(65,65);
          glEnd();
     glColor3ub(139,151,165);
          glBegin(GL_POLYGON);
          glVertex2f(35,55);
          glVertex2f(35,63);
          glVertex2f(63,63);
          glVertex2f(63,55);
          glEnd();
          glColor3ub(191,136,42);
          glBegin(GL_POLYGON);
          glVertex2f(38,53);
          glVertex2f(38,55);
```

```
glVertex2f(60,55);
glVertex2f(60,53);
glEnd();
glColor3ub(191,136,42);
glBegin(GL_POLYGON);
glVertex2f(60,53);
glVertex2f(60,55);
glVertex2f(63,55);
//glVertex2f(60,53);
glEnd();
glColor3ub(139,151,165);
glBegin(GL_POLYGON);
glVertex2f(57,40);
glVertex2f(55,53);
glVertex2f(60,53);
glVertex2f(62,40);
glEnd();
glColor3ub(148,131,123);
glBegin(GL_POLYGON);
glVertex2f(57.5,40.5);
glVertex2f(55.5,52.5);
glVertex2f(59.5,52.5);
glVertex2f(61.5,40.5);
glEnd();
glColor3ub(148,131,123);
glBegin(GL_POLYGON);
glVertex2f(52,50);
glVertex2f(51,53);
glVertex2f(55,53);
glVertex2f(55.5,50);
glEnd();
```

```
//strips
glColor3ub(0,0,0);
glBegin(GL_POLYGON);
glVertex2f(50,60);
glVertex2f(50,63);
glVertex2f(50.5,63);
glVertex2f(50.5,60);
glEnd();
glColor3ub(0,0,0);
glBegin(GL_POLYGON);
glVertex2f(51,60);
glVertex2f(51,63);
glVertex2f(51.5,63);
glVertex2f(51.5,60);
glEnd();
glColor3ub(0,0,0);
glBegin(GL_POLYGON);
glVertex2f(52,60);
glVertex2f(52,63);
glVertex2f(52.5,63);
glVertex2f(52.5,60);
glEnd();
glColor3ub(0,0,0);
glBegin(GL_POLYGON);
glVertex2f(53,60);
glVertex2f(53,63);
glVertex2f(53.5,63);
glVertex2f(53.5,60);
glEnd();
glColor3ub(0,0,0);
glBegin(GL_POLYGON);
```

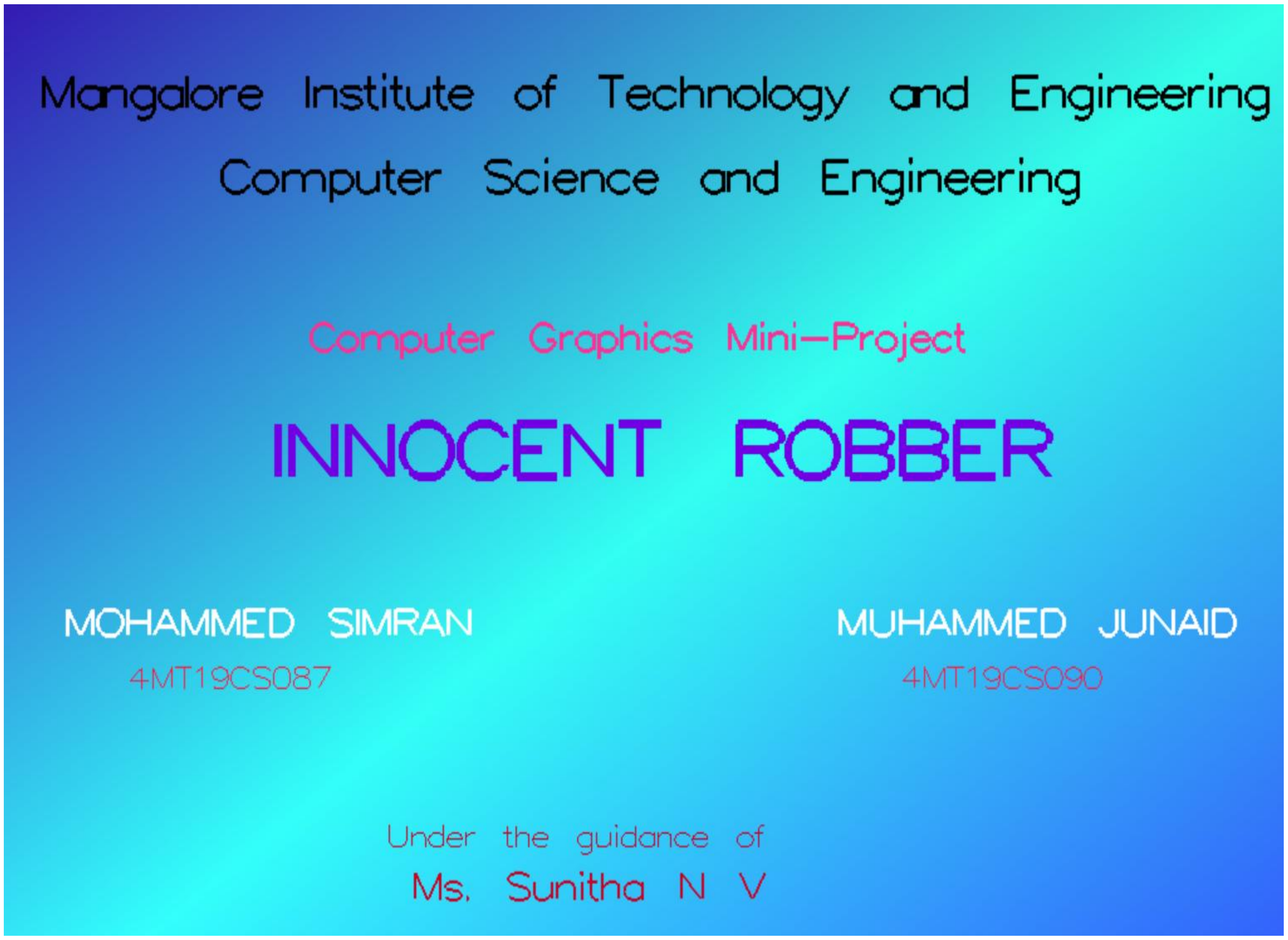
```
            glVertex2f(54,60);
            glVertex2f(54,63);
            glVertex2f(54.5,63);
            glVertex2f(54.5,60);
            glEnd();
            glColor3ub(0,0,0);
            glBegin(GL_POLYGON);
            glVertex2f(55,60);
            glVertex2f(55,63);
            glVertex2f(55.5,63);
            glVertex2f(55.5,60);
            glEnd();
            glColor3ub(0,0,0);
            glBegin(GL_POLYGON);
            glVertex2f(56,60);
            glVertex2f(56,63);
            glVertex2f(56.5,63);
            glVertex2f(56.5,60);
            glEnd();
            glColor3ub(0,0,0);
            glBegin(GL_POLYGON);
            glVertex2f(57,60);
            glVertex2f(57,63);
            glVertex2f(57.5,63);
            glVertex2f(57.5,60);
            glEnd();
    }
```

Fig 4.4.5: Code Snippet for Hammer function.

In this function we have written code for breaking the piggy bank with hammer, after breaking the piggy bank a gun appears in the hand of the robber.

CHAPTER 5

RESULTS AND SNAPSHOTS



Mangalore Institute of Technology and Engineering

Computer Science and Engineering

Computer Graphics Mini-Project

INNOCENT ROBBER

MOHAMMED SIMRAN

4MT19CS087

MUHAMMED JUNAID

4MT19CS090

Under the guidance of

Ms. Sunitha N V

Fig 5.1.1: Introduction page

Fig 5.1.1 is about the introduction page where the college name and the students name and lecturers name is displayed.

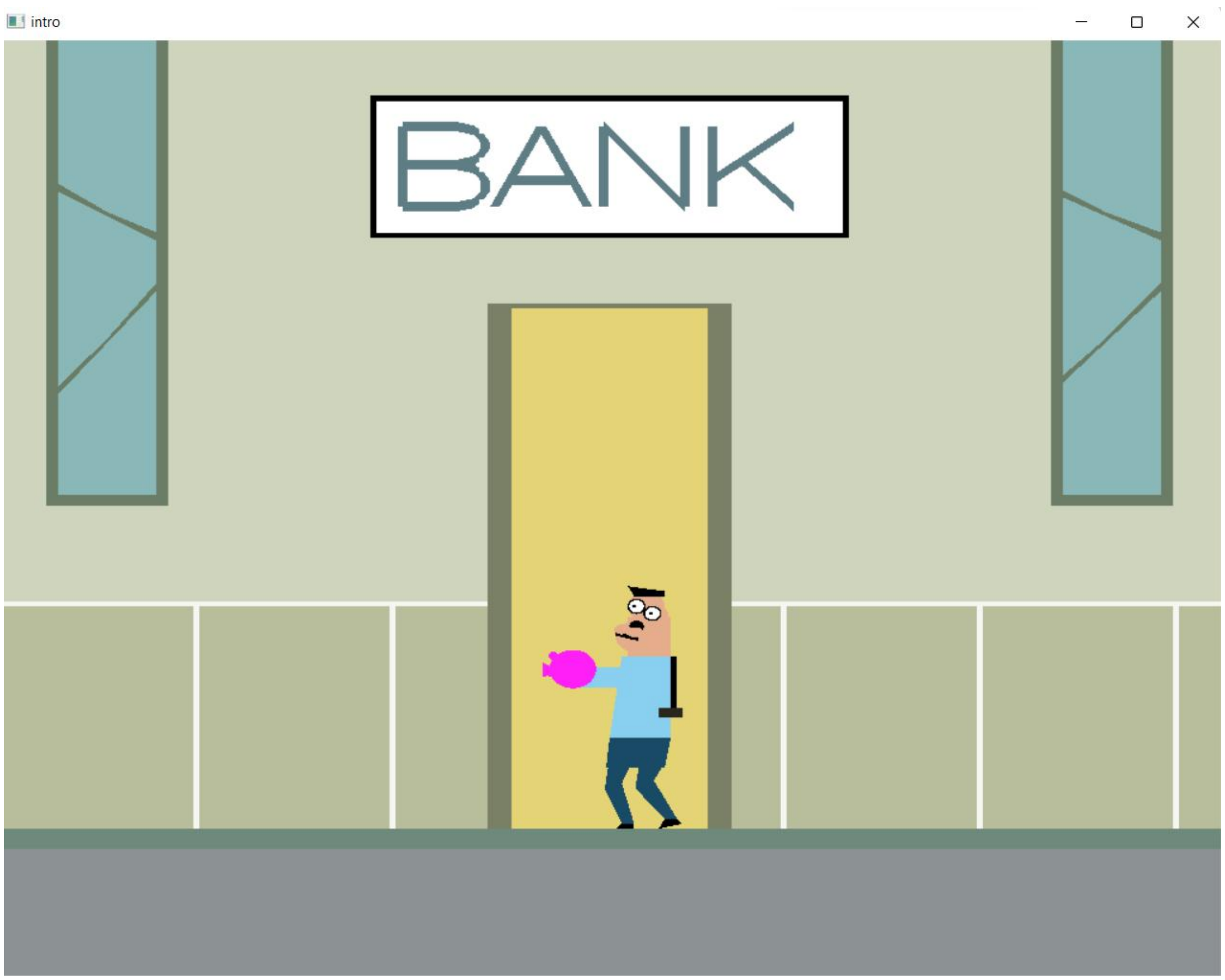


Fig 5.1.2 :Robber entering Bank with a Piggy Bank

In the above figure 5.1.2 it is displayed that the man(robber) is entering the bank with a piggy bank.

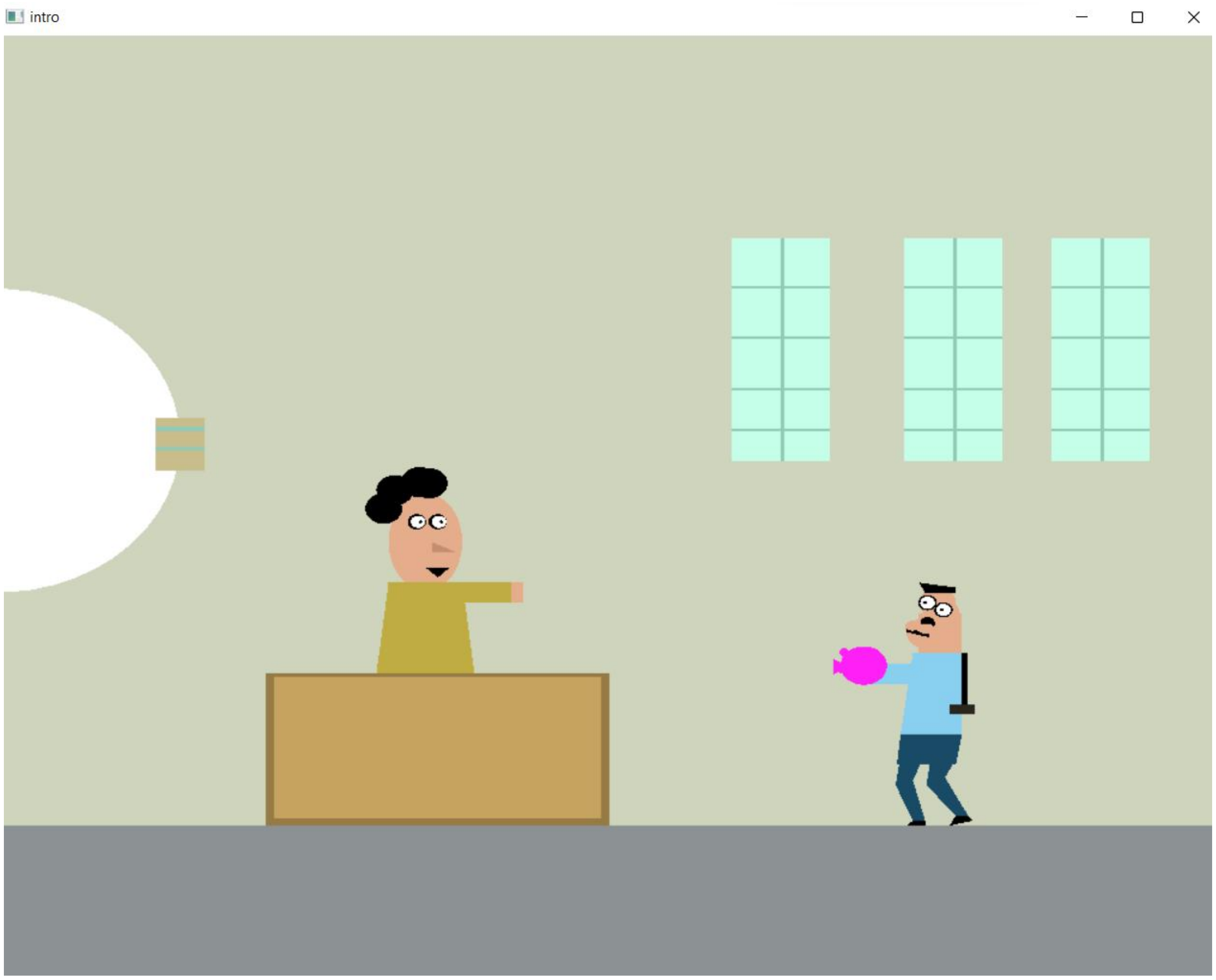
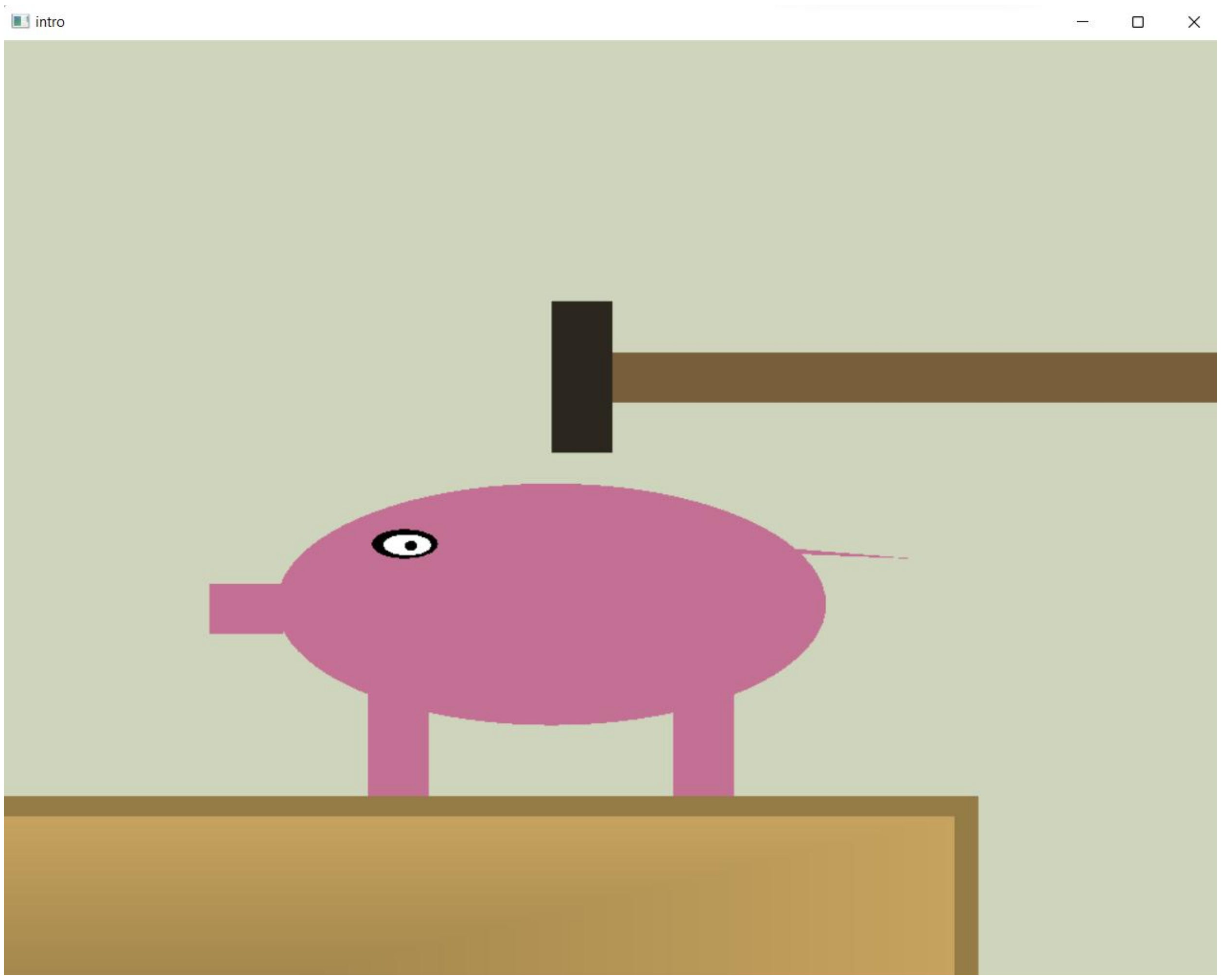


Fig 5.1.3 :Inside The Bank Scenario

In fig5.1.3 the inside view of the bank is displayed where a receptionist is happy about the nab bringing the piggy bank she receptionist thinks the man(robber) has come to deposit the money.



5.1.4: The Robber Breaks The Piggy Bank with The Hammer

In the fig5.1.4 it is displayed that the piggy bank is broken by a hammer.

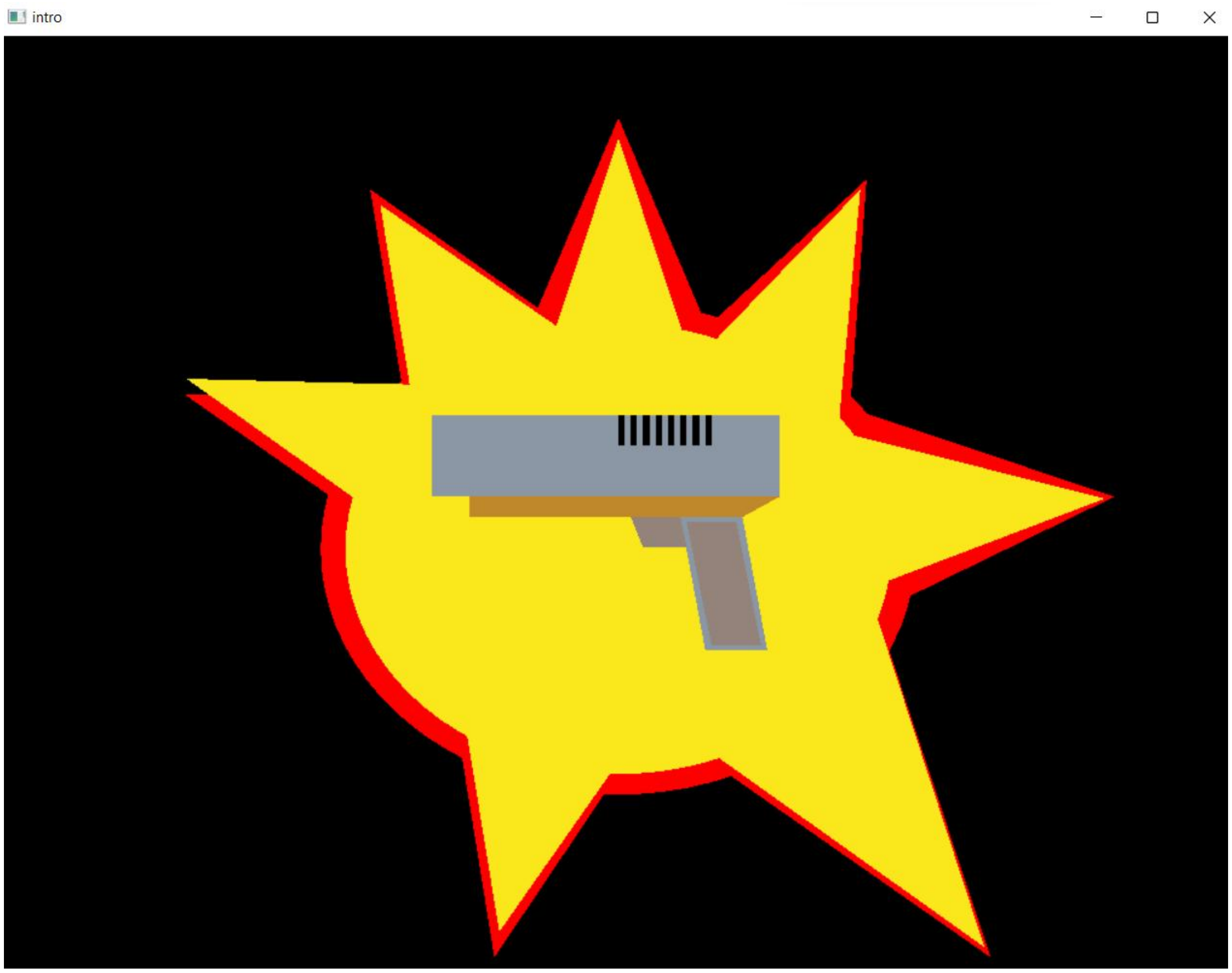
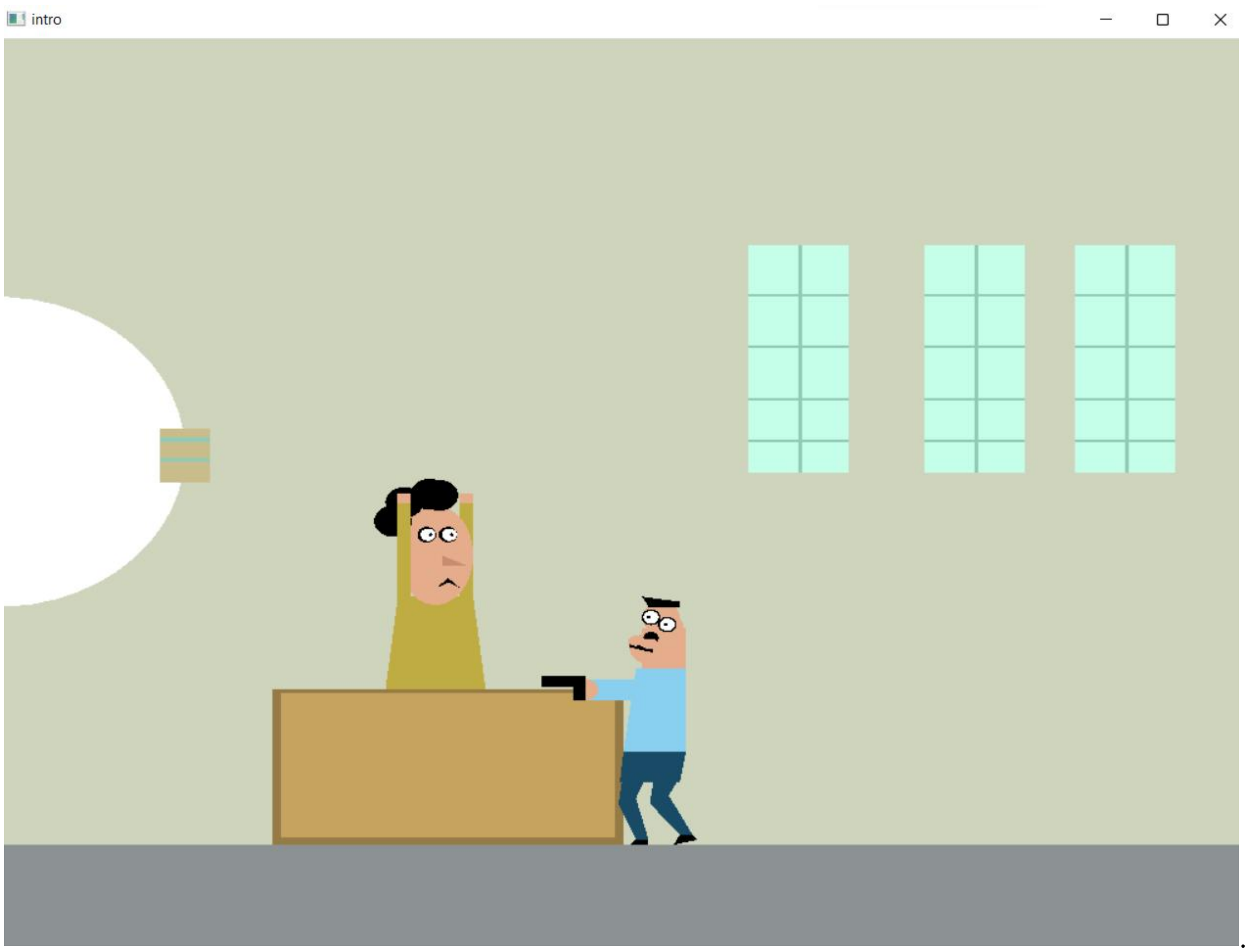


Fig 5.1.5: After Breaking The Piggy Bank A gun Appears From It

In the above fig 5.1.5 after breaking the hammer a gun appears out of piggy bank.



5.1.6: The Robber Points The Gun To Receptionist .

Fig 5.1.6 shows that the man is a robber and the robber I pointing the gun to receptionist .

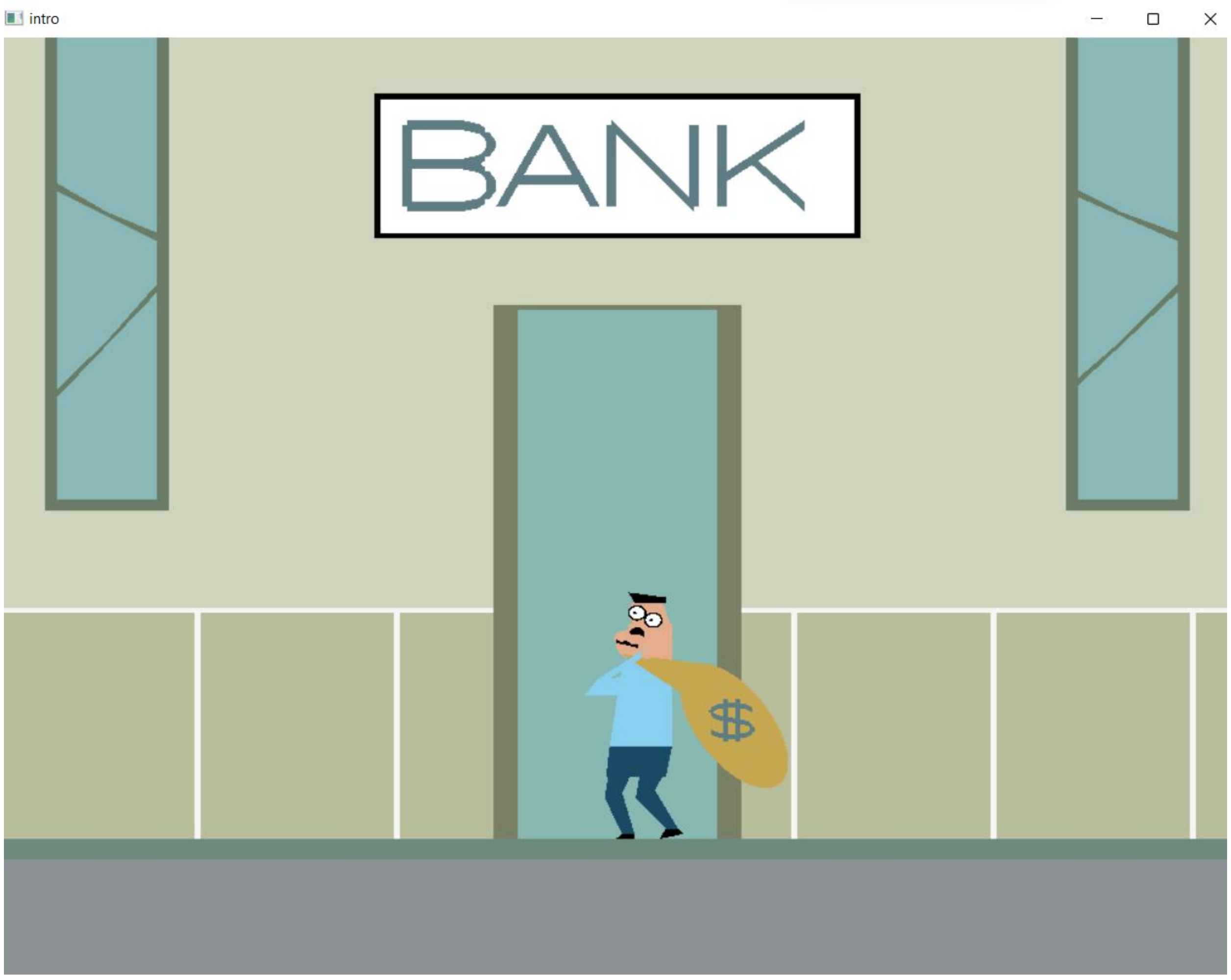


Fig 5.1.7: Robber Running Away After Looting The Bank.

In the above fig 5.1.7 we can see that the robber is running away with the money that he looted from the bank.

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENTS

6.1 CONCLUSION

This project demonstrates the example of a bank robbery using computer graphics and different functions of OpenGL.. The overall implementation of the code is done in WINDOWS and the programming language used is C with OpenGL, which is mainly used to design the Computer graphics project.

6.2 FUTURE ENHANCEMENTS

This project shows the simulation how the robbery is done even of heavy security. Better visualization can be given using OpneGL graphics library. The proposed project uses OpenGL's graphic functions in 2D. As an improvement one can make use of 3D graphics with lighting and camera options for the same demonstration. Compact and sophisticated code can be used.

REFERENCES

1) Donald Hearn and 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,2011