Bangladesh University of Business and Technology (BUBT)

Department of Computer Science and Engineering

Project Report on

Road Cross Alert Project

CSE 342-Computer Graphics Lab

Submitted To

Md. Abu Bakkar Siddique

Lecturer

Department of CSE

BUBT

Submitted By

NAME	ID
Md SahrialAlam	17182103334
Md Ikbal Hosen	17182103317
Md Kairul Anam Mubin	17182103286

ACKNOWLEDGEMENTS

We take this occasion to thank God, almighty for blessing us with His grace and taking our endeavour to a successful culmination. We extend our sincere and heartfelt thanks to our esteemed project adviser **Md. Abu Bakkar Siddique**, Lecturer, Department of CSE, BUBT for his invaluable guidance during the course of this project work. We extend my sincere thanks to him for his continuously helped throughout the project and without his guidance, this project would have been an uphill task.

Last but not the least, we would like to thank friends for the support and encouragement they have given us during the course of our work.

MdSahrialAlam
MdIkbal Hosen
Md Kairul Anam Mubin

ABSTRACT

Computer graphics is a powerful medium for presentation and design. In early days of its usage, it has been used mainly for presentation. Then it was started to use computer graphics in design development stage. Even more, nowadays you can get an inspiration from it. With tracing the change, we can see a centripetal movement of usage from the fringe to the core of the design field. This paper will describe how this change occurred with what kind of effort

DEDICATION

We would like to dedicate this software to
our parents & teachers.

INDEX

CHAPTER 1

INTRODUCTION

MS-Paint program comes pre-installed on Microsoft's 'Windows' series of operating systems. It is the default graphics program for windows systems. It's mainly used for basic image editing purposes. This project aims to simulate the working of this MS-Paint program. For creating the GUI and implementing various functionalities, OpenGL graphics library has been used in C language. The project has been implemented in Microsoft Visual Studio Professional Edition 2008 and 2013 which uses C and/or C++ as the language tool.

OpenGL provide various viewing function that helps us to develop various views of single object, and the way in which it appears on screen. Orthographic projection is the default view. OpenGL also provide various transformation functions with the help of these functions user can render its object at the desired location on the screen. In OpenGL we obtain viewing and modeling functionality through a small set of transformation functions. We can even rotate the object along desired locations and with the desired angle on the screen.

OpenGL provides a set of commands to render a three dimensional scene. OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed GLUT library.

GLUT is a complete API written by Mark Kilgard which allows us to create windows and render the 2D or 3D scenes. It exists for several platforms, that means that a program which uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.

The various features implemented in this paint program are-

Open

Open: This option can be used to open any previously saved image or drawing by the user.

Save

Save: This option can be used by the user to save any drawing.

Clear

Clear: This option clears the screen completely.

2D-gasket

2D-Gasket: This option can be used to draw the 2D Sierpinski Gasket.

2D-House

2D-House: This option can be used to draw 2D house of any color.

3D-gasket

3D-Gasket: This option can be used to draw the 3D Sierpinski Gasket.

Pencil: This tool can be used for free-hand drawing.

Line: This tool can be used to draw any straight line.

Triangle: This tool can be used to draw any type of outlined triangle.

Rectangle: This tool can be used to draw rectangle of any dimension.

Polygon: This tool can be used to draw polygon of any kind.

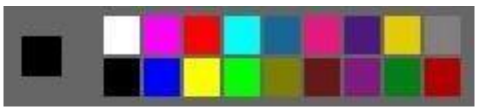
Circle: This tool can be used to draw a circle of specified radius.

Spray: This tool can be used for spray paint.

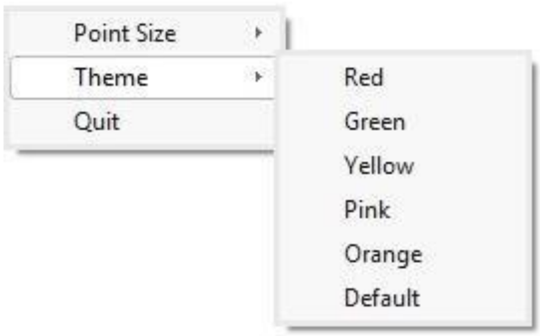
Eraser: This tool can be used to erase any drawing or a part of it.

Fill: This tool can be used to fill any closed object with selected color.

Brush: This tool can be used to draw or fill color with free hand.



Color Palette: This option can be used to select any particular color.



Menu: This menu appears on right click of the mouse and can be used to change the point size, Theme or to quit the application.

CHAPTER 2

REQUIREMENTS ANALYSIS

Software Requirements

Operating system like Windows XP, Windows 7 and Windows 8 is the platform required to develop 2D & 3D graphics applications.

A Visual C++ compiler is required for compiling the source code to make the executable file which can then be directly executed.

A built-in graphics library like glut, glut32 & header file like glut.h and DLL i.e. dynamic link libraries like glut, glut32 are required for creating the 3D layout.

Hardware requirements

The hardware requirements are very minimal and the software can be made to run on most of the machines.

Processor: Above x86

Processor speed: 500 MHz and above

RAM: 64 MB or above storage space 4GB and more

Monitor Resolution: A color monitor with a minimum resolution of 640*480

Platform

The package is implemented using Microsoft visual C++ under the windows programming environment. OpenGL and associated toolkits are used for the package development.

CHAPTER 3

DESIGN

Header files and their descriptions

#include<GL/glut.h> #include<stdio.h> #include<math.h> #include<windows.h>

- **GL/glut.h**: This header files provides an interface with application programs that are installed on the system.

- **stdio.h**: Input & Output operations can also be performed in C using C standard input, output library.

- **math.h**: contains constant definitions and external subordinate declarations for the math subroutine library.

- **Windows.h**: handles the output window creation.

Description of OpenGL functions

Glut library functions used are:

main(): Execution of any program always starts with main function irrespective of where it is written in a program.

glutSwapBuffe rs(void): Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monito r, rather than immediately after glutSwapBuffers is called.

glutPostRedisplay(): marks the plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window’s display cal back wil be cal ed to redisplay the

window’s normal plane. Multiple cal s to glutPostRedisplay before the next display cal back opportunity generates only a single redisplay callback.

glutInit(&argc,char ** argv): glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

glutInitDisplayMode(unsigned int mode): The initial display mode is used when creating toplevel windows, sub windows, and overlays to determine the OpenGL display mode for the to-becreated window or overlay.

GLUT_RGB specifies the structure of each pixel. GLUT_DEPTH is a buffer to hold the depth of each pixel.

glutCreateWindow(char *title): The glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name.

glutReshapeFunc():glutReshapeFunc sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width and height parameters of the callback specify the new window size in pixels. Before the callback, the current window is set to the window that has been reshaped.

glutMainLoop():glutMainLoop enters the GLUT event processing loop. 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.

glutBitmapCharacter(font,c): Without using any display lists, glutBitmapCharacter renders the character ‘c’ in the named bitmap font.

glPointSize(size): glPointSize specifies the rasterized diameter of points.

glFlush(): Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

glRasterPos3f(x , y, z): This function is used to set the cursor at the x, y, z location.
glBegin(enum) and glEnd(): glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives. glBegin accepts a single argument that specifies in which often ways the vertices are interpreted.

glColor3f(R,G,B): This function is used to pick a color of specified R,G,B value.

glClear(GLbitfield): glClear sets the bit plane area of the window to values previously selected by glClearColor, glClearIndex, glClearDepth, glClearStencil, and glClearAccum. Multiple color buffers can be cleared simultaneously by selecting more than one buffer at a time using glDrawBuffer.

glVertex2f(x,y): This function is used to draw a vertex at location x,y.

glEnable(GLenum): This function is used to enable the various functionalities like enabling the light, enabling the Z buffer.

glDisable(GLenum): This function is used to disable the various functionalities like enabling the light, enabling the Z buffer.

glRectf(x0,y0,x1,y1): This function is used to draw rectangle using the two diagonal points.

glClearColor(R,G,B,A):glClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range 0 to1 .

glMaterialfv(GLenum,GLenum,GLfloat *): glMaterial assigns values to material parameters. There are two matched sets of material parameters. One, the front-facing set, is used to shade points, lines, bitmaps, and all polygons (when two-sided lighting is disabled), or just front-facing polygons (when two-sided lighting is enabled).

glutSolidTeapot(size): glutSolidTeapot and glutWireTeapot render a solid or wireframe teapot respectively. Both surface normals and texture coordinates for the teapot are generated. The teapot is generated with OpenGL evaluators

glutSolidCube(size): glutSolidCube and glutWireCube render a solid or wireframe cube respectively. The cube is centered at the modeling coordinate’s origin with sides of length size.

glutDisplayFunc(void (*func)()): glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and (if no overlay display callback is registered) the layer in use is set to the normal plane. The display callback is called with no parameters.

glutKeyboardFunc(void (*func)(unsigned char key, int x, int y)): glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character

glutMouseFunc(void (*func)(int button, int state, int x, int y)): glutMouseFunc sets the mouse callback for the current window. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The button parameter is one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON.

glutMotionFunc(void (*func)(int x, int y)): glutMotionFunc set the motion callback for the current window. The motion callback for a window is called when the mouse moves within the window while one or more mouse buttons are pressed.

glutPassiveMotionFunc(void(*func)(int x, int y)): glutPassiveMotionFunc set the passive motion callback for the current window. The passive motion callback for a window is called when the mouse moves within the window while no mouse buttons are pressed.

glMatrixMode(GLenum) mode: Specifies which matrix stack is the target for subsequent matrix operations. Values accepted are: GL_MODELVIEW, GL_PROJECTION.The initial value is GL_MODELVIEW. Additionally, if the ARB tension extension is supported, GL_COLOR is also accepted.

gluPerspective(fovy, aspect, near, far): gluPerspective specifies a viewing frustum into the world coordinate system. In general, the aspect ratio in gluPerspective should match the aspect ratio of the associated viewport. For example, aspect = 2.0means the viewer's angle of view is twice as wide in x as it is in y.

glOrtho(left, right, top, bottom, Znear, Zfar): glOrtho describes a transformation that produces a parallel projection. The current matrix is multiplied by this matrix and the result replaces the current matrix.

Description of User-defined functions

Various user-defined functions used are:

void drawstring(float x, float y, float z, char *string): Used to display text.

void drawpoint(int x, int y): Used to draw point.

void paint(int x, int y): Used for PAINT BRUSH function.

void eraser(int x, int y): Used for ERASER function.

void reset(): It resets the variables in which vertices are stored, after a polygon is drawn.

void draw_pixel(GLfloat x, GLfloat y): It's used to draw the points of a circle.

void draw_circle(GLfloat p, GLfloat q, GLfloat r): It's used to draw a CIRCLE using MIDPOINT CIRCLE DRAWING algorithm

void draw_circle1(GLfloat p, GLfloat q, GLfloat r): It's used to draw the circle option on the tool bar.

void edgedetect(float x1, float y1, float x2, float y2, int *le, int *re): It's used to detect edges of the polygon to be filled.

void scanfill(GLint num1, GLint num2): It's used to FILL a Polygon using SCAN LINE ALGORITHM.

void detect_point(GLint num1, GLint num2, int x, int y): It's used to detect which POLYGON TO BE FILLED.

void display(void): It's user-defined DISPLAY function.

void keys(unsigned char key, int x, int y): It’s used to get FILENAME from the KEYBOARD.

void divide_trinagle(GLfloat *a, GLfloat *b, GLfloat *c, int k): It’s used for drawing the 2D Sierpinski gasket.

void tetra(GLfloat *a, GLfloat * b, GLfloat * c, GLfloat * d): It’s used in drawing 3D Sierpinsk i gasket.

void divide_tetra(GLfloat * a, GLfloat * b, GLfloat * c, GLfloat * d, int m): It’s used in drawing 3D Sierpinski gasket.

void myMouse(int btn, int state, int x, int y): It’s used to handle all the mouse events. void myReshape(GLsizei w, GLsizei h): It’s user defined RESHAPE FUNCTION. void point_size(int id): Used to change the pixel size.

CHAPTER 4

IMPLEMENTATION

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#include <stdlib.h>
#include <math.h>

GLint keyl,keyr,flag=0;
float  counter=600.0, cnt=-150.0,r1=0.0,g1=1.0,b1=0.0,bc=-260.0;
float r2=0.0,g2=1.0,b2=1.0,r=1.0,g=0.0,b=0.0;
int c=1,d=1;

void  road();
void  grass();
void  grass2();
void  line1();
void  line2();
void  line3();
void  line4();
void  car();
void  truck();
void  bus();
void  sq();
void  text();
void  tc();
void  light();
void  light2();
void *currentfont;


void setFont(void *font)
{
        currentfont=font;
}

void drawstring(float x,float y,float z,char *string)
{
        char *ct;
        glRasterPos3f(x,y,z); //glRasterPos — specify the raster position for pixel operations

        for(ct=string;*ct!='\0';ct++)
        {           glColor3f(0.0,0.0,0.0);
                    glutBitmapCharacter(currentfont,*ct); //glutBitmapCharacter renders a bitmap
character using OpenGL
        }
}

void initOpenGl()
{
    glClearColor(0.2,0.6,0.99,0); //Background Color Blue r g b a
    //glClearColor(0.0,0.0,0.0,0); //Background Color
    glMatrixMode(GL_PROJECTION); //glMatrixMode — specify which matrix is the current
matrix
    glLoadIdentity();
    gluOrtho2D(0,700,0,500); //gluOrtho2D — define a 2D orthographic projection matrix
,left,right,bottom,top
    glMatrixMode(GL_MODELVIEW);
//    The modelview matrix defines how your objects are transformed (meaning translation,rotation
and scaling) in your world coordinate frame

//The projection matrix defines the properties of the camera that views the objects in the world
coordinate frame. Here you typically set the zoom factor, aspect ratio and the near and far clipping
planes.
}

void text()
{
   setFont(GLUT_BITMAP_HELVETICA_18);
   glColor3f(1,1,1);
   drawstring(160,130,0.0,"BUBT BUS");
   glColor3f(1,1,1);
}



void tc()
{//tower
   glLoadIdentity();
   glColor3f(0.0,0.0,1.0);
   glBegin(GL_POLYGON);
   glVertex2f(310,190);
   glVertex2f(310,390);
   glVertex2f(340,390);
   glVertex2f(340,190);
   glEnd();

}

void light()
{
   glLoadIdentity(); //glLoadIdentity replaces the current matrix with the identity matrix
   glColor3f(1.0,0.0,0.0);
   glBegin(GL_POLYGON);
   glVertex2f(315,330);
   glVertex2f(315,370);
   glVertex2f(335,370);
   glVertex2f(335,330);
   glEnd();
}
```

```
void light2()
{
   glLoadIdentity();
   glColor3f(0.0,1.0,0.0);
   glBegin(GL_POLYGON);
   glVertex2f(315,330);
   glVertex2f(315,370);
   glVertex2f(335,370);
   glVertex2f(335,330);
   glEnd();
}

void light3()
{
   glLoadIdentity();
   glColor3f(1.0,0.0,0.0);
   glBegin(GL_POLYGON);
   glVertex2f(315,280);
   glVertex2f(315,320);
   glVertex2f(335,320);
   glVertex2f(335,280);
   glEnd();
}

void light4()
{
   glLoadIdentity();
   glColor3f(0.0,1.0,0.0);
   glBegin(GL_POLYGON);
   glVertex2f(315,280);
   glVertex2f(315,320);
   glVertex2f(335,320);
   glVertex2f(335,280);
   glEnd();
}

void window(int w1,int w2)
```

```
{
    glColor3f(0.0,0.0,0.0);
    glBegin(GL_POLYGON);
    glVertex2f(w1,160);//glVertex commands are used within glBegin/glEnd pairs to specify point,
line, and polygon vertices
    glVertex2f(w1,185);
    glVertex2f(w2,185);
    glVertex2f(w2,160);
    glEnd();
}

void buswindow(int w1,int w2)
{
    glColor3f(0.0,0.0,0.0);
    glBegin(GL_POLYGON);
    glVertex2f(w1,180);
    glVertex2f(w1,205);
    glVertex2f(w2,205);
    glVertex2f(w2,180);
    glEnd();
}



void wheel(int x,int y)
{
   float th;
   glBegin(GL_POLYGON);
   glColor3f(0,0,0);
   for(int i=0;i<360;i++)
   {
       th=i*(3.1416/180);
       glVertex2f(x+20*cos(th),y+20*sin(th));
   }

   glEnd();

}
```

```
void road()
{
    glLoadIdentity();
    //1
   // glLoadIdentity — replace the current matrix with the identity matrix
    glColor3f(0.5,0.5,0.5);//gray black road
    glBegin(GL_POLYGON);
    glVertex2f(0,95);
    glVertex2f(0,260);
    glVertex2f(800,260);
    glVertex2f(800,95);
    glEnd();
}

void grass()
{
    //2
    glLoadIdentity();
    glColor3f(0.0,0.5,0.1);
    glBegin(GL_POLYGON);
    glVertex2f(0,0);
    glVertex2f(0,95);
    glVertex2f(800,95);
    glVertex2f(800,0);
    glEnd();
}

void grass2()
{
    //3
    glLoadIdentity();
    glColor3f(0.0,0.5,0.1);
    glBegin(GL_POLYGON);
    glVertex2f(0,260);
    glVertex2f(0,300);
    glVertex2f(800,300);
    glVertex2f(800,260);
    glEnd();
}

void line1()
{
    //4
    glLoadIdentity();
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(0,190);
    glVertex2f(150,190);
    glEnd();

}

void line2()
{
    //5
    glLoadIdentity();
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(200,190);
    glVertex2f(300,190);
    glEnd();

}

void line3()
{
    //6
    glLoadIdentity();
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(350,190);
    glVertex2f(500,190);
    glEnd();
```

```
}

void line4()
{ //7
   glLoadIdentity();
   //glLoadIdentity() function ensures that each time when we enter the projection mode, the
matrix will be reset to identity matrix, so that the new viewing parameters are not combined with
the previous one
   glColor3f(1.0,1.0,1.0);
   glBegin(GL_LINE_LOOP);
   glVertex2f(550,190);
   glVertex2f(700,190);
   glEnd();

}

void zebra_crossing()
{
   glLoadIdentity();
   glColor3f(1.0,1.0,1.0);
   glBegin(GL_POLYGON);
   glVertex2f(200,95);
   glVertex2f(200,260);
   glVertex2f(250,260);
   glVertex2f(250,95);
   glEnd();

}


void zebra_crossing1()
{
   glLoadIdentity();
   glColor3f(1.0,1.0,1.0);
   glBegin(GL_POLYGON);
   glVertex2f(400,95);
   glVertex2f(400,260);
```

```
    glVertex2f(450,260);
    glVertex2f(450,95);
    glEnd();

}



void zebra_line1()
{
    //4
    glLoadIdentity();
    glColor3f(0.5,0.5,0.5);
    glBegin(GL_POLYGON);
    glVertex2f(200,230);
    glVertex2f(250,230);
    glVertex2f(250,225);
    glVertex2f(200,225);
    glEnd();

}
void zebra_line2()
{
    //4
    glLoadIdentity();
    glColor3f(0.5,0.5,0.5);
    glBegin(GL_POLYGON);
    glVertex2f(200,200);
    glVertex2f(250,200);
    glVertex2f(250,195);
    glVertex2f(200,195);
    glEnd();

}
void zebra_line3()
{
    //4
    glLoadIdentity();
    glColor3f(0.5,0.5,0.5);
    glBegin(GL_POLYGON);
    glVertex2f(200,170);
    glVertex2f(250,170);
    glVertex2f(250,165);
    glVertex2f(200,165);
    glEnd();

}
void zebra_line4()
{
    //4
    glLoadIdentity();
    glColor3f(0.5,0.5,0.5);
    glBegin(GL_POLYGON);
    glVertex2f(200,140);
    glVertex2f(250,140);
    glVertex2f(250,135);
    glVertex2f(200,135);
    glEnd();

}
void zebra_line5()
{
    //4
    glLoadIdentity();
    glColor3f(0.5,0.5,0.5);
    glBegin(GL_POLYGON);
    glVertex2f(200,115);
    glVertex2f(250,115);
    glVertex2f(250,110);
    glVertex2f(200,110);
    glEnd();

}
```

```
void zebra_line6()
{
   //4
   glLoadIdentity();
   glColor3f(0.5,0.5,0.5);
   glBegin(GL_POLYGON);
   glVertex2f(400,230);
   glVertex2f(450,230);
   glVertex2f(450,225);
   glVertex2f(400,225);
   glEnd();

}
void zebra_line7()
{
   //4
   glLoadIdentity();
   glColor3f(0.5,0.5,0.5);
   glBegin(GL_POLYGON);
   glVertex2f(400,200);
   glVertex2f(450,200);
   glVertex2f(450,195);
   glVertex2f(400,195);
   glEnd();

}
void zebra_line8()
{
   //4
   glLoadIdentity();
   glColor3f(0.5,0.5,0.5);
   glBegin(GL_POLYGON);
   glVertex2f(400,170);
   glVertex2f(450,170);
   glVertex2f(450,165);
   glVertex2f(400,165);
   glEnd();
```

```
}
void zebra_line9()
{
    //4
    glLoadIdentity();
    glColor3f(0.5,0.5,0.5);
    glBegin(GL_POLYGON);
    glVertex2f(400,140);
    glVertex2f(450,140);
    glVertex2f(450,135);
    glVertex2f(400,135);
    glEnd();

}
void zebra_line10()
{
    //4
    glLoadIdentity();
    glColor3f(0.5,0.5,0.5);
    glBegin(GL_POLYGON);
    glVertex2f(400,115);
    glVertex2f(450,115);
    glVertex2f(450,110);
    glVertex2f(400,110);
    glEnd();

}

void boy(float a, float b){

 glBegin(GL_POLYGON);//head
 glColor3f(1, 1, 1);



 glVertex2f(20 + a, 230 + b);//head 1st co-ordinate(-x+y)
 glVertex2f(20.3 + a, 231 + b);
```

```
glVertex2f(20.7+ a, 232 + b);
glVertex2f(21+ a, 233 + b);
glVertex2f(21.3 + a, 234 + b);
glVertex2f(21.7 + a, 235 + b);
glVertex2f(22.3+ a, 236 + b);
glVertex2f(22.7 + a, 236.5 + b);
glVertex2f(23 + a, 237 + b);
glVertex2f(23.5 + a, 237.3 + b);
glVertex2f(24 + a, 237.7 + b);
glVertex2f(24.5 + a, 238+ b);
glVertex2f(25 + a, 238.3 + b);
glVertex2f(26 + a, 238.7 + b);
glVertex2f(27 + a, 239 + b);
glVertex2f(28 + a, 239.3 + b);
glVertex2f(29 + a, 239.7 + b);
glVertex2f(30 + a, 240 + b);

glVertex2f(31 + a, 239.7 + b);//head 2nd co-ordinate(+x+y)
glVertex2f(32 + a, 239.5 + b);
glVertex2f(33 + a, 239.3 + b);
glVertex2f(33.5 + a, 239 + b);
glVertex2f(34 + a, 238.7 + b);
glVertex2f(34.5 + a, 238.3+ b);
glVertex2f(34.7 + a, 238 + b);
glVertex2f(34.9 + a, 237.7 + b);
glVertex2f(35 + a, 237.5 + b);
glVertex2f(35.2 + a, 237.4 + b);
glVertex2f(35.3 + a, 237.3 + b);
glVertex2f(35.5 + a, 237.1 + b);
glVertex2f(35.7 + a, 237 + b);
glVertex2f(35.9 + a, 236.9 + b);
glVertex2f(36 + a, 236.7 + b);
glVertex2f(36.3 + a, 236.3 + b);
glVertex2f(36.7+ a, 236 + b);
glVertex2f(37 + a, 235.7 + b);
glVertex2f(37.3 + a, 235.3+ b);
glVertex2f(37.7 + a, 235 + b);
```

```
glVertex2f(38 + a, 234.5 + b);
glVertex2f(38.3 + a, 234+ b);
glVertex2f(38.7 + a, 233.5 + b);
glVertex2f(39 + a, 233 + b);
glVertex2f(39.3 + a, 232+ b);
glVertex2f(39.7 + a, 231 + b);
glVertex2f(40 + a, 230+ b);


glVertex2f(39.7 + a, 229 + b);//head 3rd co-ordinate(+x-y)
glVertex2f(39.3 + a, 228 + b);
glVertex2f(39 + a, 227 + b);
glVertex2f(38.7 + a, 226.5 + b);
glVertex2f(38.3 + a, 226 + b);
glVertex2f(38 + a, 225.5 + b);
glVertex2f(37.7 + a, 225 + b);
glVertex2f(37.3 + a, 224.5 + b);
glVertex2f(37 + a, 224 + b);
glVertex2f(36.7 + a, 223.7 + b);
glVertex2f(36 + a, 223.3 + b);
glVertex2f(35.7 + a, 223  + b);
glVertex2f(35.3 + a, 222.7 + b);
glVertex2f(35 + a, 222.3 + b);
glVertex2f(34.5 + a, 222 + b);
glVertex2f(34 + a, 221.7 + b);
glVertex2f(33.5 + a, 221.3 + b);
glVertex2f(33 + a, 221 + b);
glVertex2f(32 + a, 220.7 + b);
glVertex2f(31 + a, 220.3 + b);
glVertex2f(30 + a, 220 + b);

glVertex2f(29 + a, 220.3 + b);//head 4th co-ordinate(-x-y)
glVertex2f(28 + a, 220.7 + b);
glVertex2f(27.5 + a, 221 + b);
glVertex2f(27 + a, 221.3 + b);
glVertex2f(26.5 + a, 221.7 + b);
glVertex2f(26 + a, 222 + b);
```

```
glVertex2f(25.5 + a, 222.3 + b);
glVertex2f(25 + a, 222.7 + b);
glVertex2f(24.5 + a, 223 + b);
glVertex2f(24 + a, 223.3 + b);
glVertex2f(23.7 + a, 223.5 + b);
glVertex2f(23.3 + a, 224 + b);
glVertex2f(23 + a, 224.5 + b);
glVertex2f(22.7 + a, 225 + b);
glVertex2f(22.3 + a, 225.5 + b);
glVertex2f(22 + a, 226 + b);
glVertex2f(21.7 + a, 226.5 + b);
glVertex2f(21.3 + a, 227 + b);
glVertex2f(21 + a, 227.5 + b);
glVertex2f(20.7 + a, 228 + b);
glVertex2f(20.3 + a, 229 + b);
glVertex2f(20 + a, 230 + b);
glEnd();

glBegin(GL_POLYGON);//left eye
glColor3f(0, 0, 0);
glVertex2f(25 + a, 228 + b);
glVertex2f(23 + a, 230 + b);
glVertex2f(25 + a, 232 + b);
glVertex2f(27 + a, 230 + b);
glEnd();

glBegin(GL_POLYGON);//right eye
glColor3f(0, 0, 0);
glVertex2f(35 + a, 228 + b);
glVertex2f(33 + a, 230 + b);
glVertex2f(35 + a, 232 + b);
glVertex2f(37 + a, 230 + b);
glEnd();

glBegin(GL_LINE_STRIP);//mouth
glColor3f(0, 0, 0);
glVertex2f(28 + a, 228 + b);
```

```
glVertex2f(30 + a, 225 + b);
glVertex2f(32 + a, 228 + b);
glEnd();


glBegin(GL_POLYGON);//body
glColor3f(.73, .36, .36);
glVertex2f(18 + a, 215 + b);
glVertex2f(25 + a, 220 + b);
glVertex2f(35 + a, 220 + b);
glVertex2f(42 + a, 215 + b);
glVertex2f(42 + a, 180 + b);
glVertex2f(18 + a, 180 + b);
glEnd();

glBegin(GL_POLYGON);//left leg
glColor3f(1, 1, 1);
glVertex2f(20 + a, 180 + b);
glVertex2f(25 + a, 180 + b);
glVertex2f(25 + a, 160 + b);
glVertex2f(20 + a, 160 + b);
glEnd();

glBegin(GL_POLYGON);//right leg
glColor3f(1, 1, 1);
glVertex2f(35 + a, 180 + b);
glVertex2f(40 + a, 180 + b);
glVertex2f(40 + a, 160 + b);
glVertex2f(35 + a, 160 + b);
glEnd();


glBegin(GL_LINES);//left hand
glColor3f(1, 1, 1);
glVertex2f(18 + a, 210 + b);
glVertex2f(12 + a, 205 + b);
glVertex2f(12 + a, 205 + b);
```

```
 glVertex2f(12 + a, 195 + b);
 glEnd();

 glBegin(GL_LINES);//right hand
 glColor3f(1, 1, 1);
 glVertex2f(42 + a, 210 + b);
 glVertex2f(50 + a, 200 + b);
 glEnd();




}




void tree1(float x1, float y1)
{
    float x2 = x1+40,y2=y1, x3=x1+20,y3=y1+50;
    glColor3ub(11, 70, 11);
    glBegin(GL_TRIANGLES);
        glVertex2d(x1, y1);
        glVertex2d(x2, y2);
        glVertex2d(x3, y3);
    glEnd();
    glBegin(GL_QUADS);
        glVertex2d(x1+10.5, y1);
        glVertex2d(x1, y1-40);
        glVertex2d(x2, y1-40);
        glVertex2d(x2-10.5, y1);

        glVertex2d(x1+10.5, y1-40);
        glVertex2d(x1, y1-80);
        glVertex2d(x2, y1-80);
        glVertex2d(x2-10.5, y1-40);
        glColor3ub(68, 43, 2);
        glVertex2d(x1+10.5, y1-80);
        glVertex2d(x1+10, y1-140);
        glVertex2d(x2-10, y1-140);
        glVertex2d(x2-10.5, y1-80);



    glEnd();



}

void car()
{

   //Bottom Part
   glLoadIdentity();
   counter=counter-0.05;
   glColor3f(r1,g1,b1);
   glTranslated(counter,80,0.0);
   //glTranslated can change path or shift car under green then on road
   if(counter<-1000.00)
   {

       c++;
       counter=700.0;
       if(c%2==0)
       {
           r1=1.0;
           g1=0.0;
           b1=0.0;

       }
       else if(c%3==0)
       {
           r1=0.0;
           g1=2.0+c;
           b1=1.0+c;
       }
       else if(c%5==0)
       {
           r1=1.0;
           g1=1.0;
           b1=0.0;
       }
       else
       {
           r1=0.0;
           g1=1.0;
           b1=0.0;
       }

   }
   glScaled(0.5,0.5,0.0);
   glBegin(GL_POLYGON);
   glVertex2f(100,100);
   glVertex2f(100,160);
   glVertex2f(450,160);
   glVertex2f(450,100);
   glEnd();

   //Top Part
   glBegin(GL_POLYGON);
   glVertex2f(150,160);
   glVertex2f(200,200);
   glVertex2f(400,200);
   glVertex2f(450,160);

   glEnd();



   window(200,270);
   window(280,330);
   window(340,390);
   wheel(200,100);
   wheel(380,100);
```

```
}

void truck()
{

  //Bottom Part

  glLoadIdentity();
  glColor3f(r2,g2,b2);
  cnt=cnt+0.04;
  if(cnt>1300.00)
  {
     cnt=-250.0;
     d++;
     if(d%2==0)
     {
        r2=r2+d;
        g2=0.0;
        b2=1.0;

     }
     else if(d%3==0)
     {
        r2=0.0;
        g2=3.0+d;
        b2=5.0+d;
     }
     else if(d%5==0)
     {
        r2=5.0;
        g2=0.0;
        b2=1.0;
     }
     else
     {
        r2=0.0;
        g2=1.0;
        b2=0.0;
     }

  }
  glTranslated(cnt,200,0.0);
  glScaled(0.4,0.4,0.0);
  glBegin(GL_POLYGON);
  glVertex2f(100,100);
  glVertex2f(100,160);
  glVertex2f(450,160);
  glVertex2f(450,100);
  glEnd();

  //Top Part
  glBegin(GL_POLYGON);
  glVertex2f(350,160);
  glVertex2f(350,200);
  glVertex2f(400,200);
  glVertex2f(450,160);

  glEnd();

  window(365,400);
  wheel(200,100);
  wheel(380,100);

}

void sq()
{
  glBegin(GL_POLYGON);
  glColor3f(0.9,0.2,0.1);
  glVertex2f(100,120);
  glVertex2f(100,170);
  glVertex2f(470,170);
  glVertex2f(470,120);
```

```
  glEnd();

}

void bus()
{

  glLoadIdentity();
  bc=bc+0.05;
  glColor3f(1.0,1.0,1.0);
  glTranslated(bc,180,0.0);
  if(bc>1300.00)
  {
     bc=-260.0;

  }
  glScaled(0.4,0.4,0.0);
  glBegin(GL_POLYGON);
  glVertex2f(100,100);
  glVertex2f(100,220);
  glVertex2f(470,220);
  glVertex2f(470,100);
  glEnd();

  buswindow(110,160);
  buswindow(170,220);
  buswindow(230,270);
  buswindow(280,330);
  buswindow(340,390);
  buswindow(400,450);
  wheel(200,100);
  wheel(380,100);

}

void display()
{
```

```
glClear(GL_COLOR_BUFFER_BIT);
//Push and pop matrix for separating circle object from Background graphic

road();
zebra_crossing();
zebra_crossing1();
zebra_line1();
zebra_line2();
zebra_line3();
zebra_line4();
zebra_line5();
zebra_line6();
zebra_line7();
zebra_line8();
zebra_line9();
zebra_line10();
grass();
grass2();
line1();
line2();
line3();
line4();
car();
truck();
bus();
sq();
text();
tc();
light();
light3();
tree1(20,430);
tree1(50,430);
tree1(80,430);
tree1(650,430);
tree1(620,430);
tree1(590,430);
```

```
  boy(200,-150);

  if(counter<-25)
  {
    light4();
  }
  if(bc>420 && cnt>420)
  {
     light2();
  }



  glutSwapBuffers();//Performs a buffer swap on the layer in use for the current window.
  glFlush();//The glFlush function empties all these buffers


}

int main(int argc, char **argv)
{
    glutInit(&argc,argv); //  glutInit is used to initialize the GLUT library.
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA|GLUT_DEPTH); //glutInitDisplayMode
sets the initial display mode.
    glutInitWindowSize(700,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Animated Road Crossing Alert System");
    initOpenGl();
    glutDisplayFunc(display); //glutDisplayFunc sets the display callback for the current window.
    glutIdleFunc(display); //glutIdleFunc sets the global idle callback.
    glutMainLoop(); //glutMainLoop enters the GLUT event processing loop
    return 0;
}
```

CHAPTER 5

RESULTS AND SNAPSHOTS

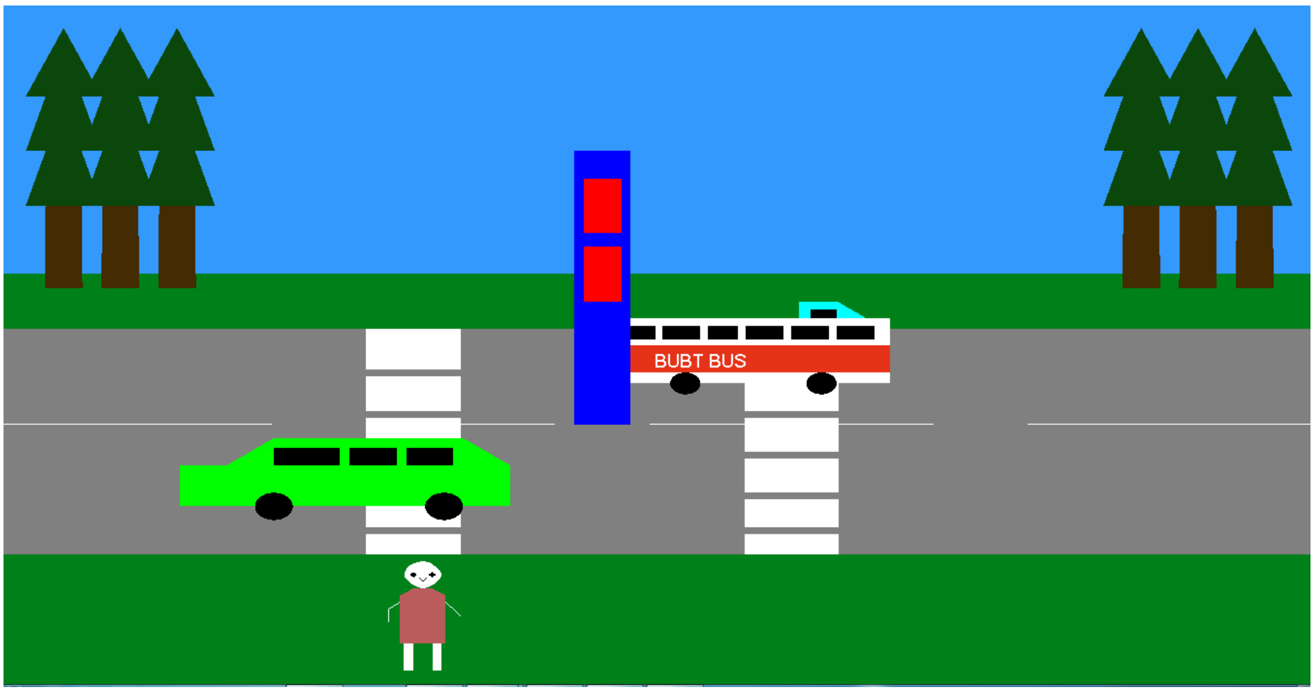


Fig1:Signal is Red when vehicles are crossing

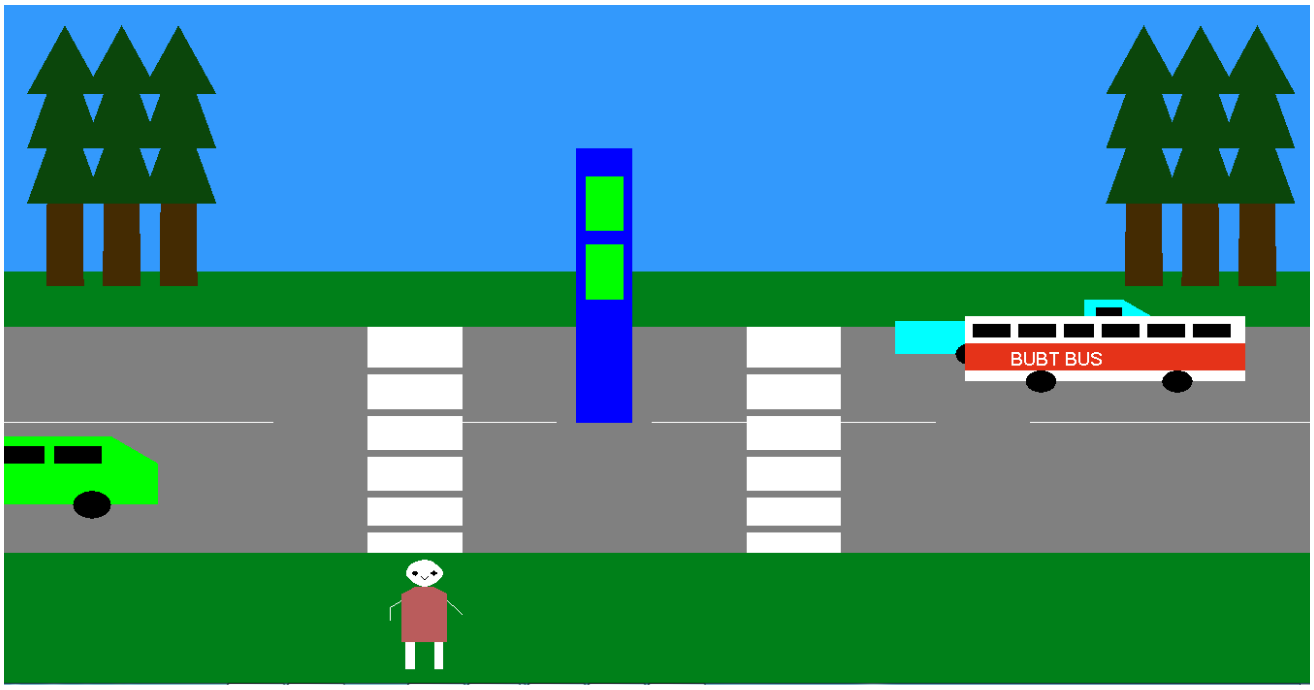


Fig2:Signal is Green when vehicles are already crossed

CHAPTER 6

CONCLUSION

The project is well suited for designing 2D and 3D objects, as well as for carrying out basic graphics functionalities like drawing a simple line, creating a cube, circle, square, erasing and filling them. However, if implemented on large scale with sufficient resources, it has the potential to become a standard stand-alone GUI based application for Windows Operating System.

Out of the many features available, the project demonstrates some popular and commonly used features of OpenGL such as Rendering, Transformation, Rotation, Lighting, Scaling etc. These graphic functions will also work in tandem with various rules involved in the game. Since this project works on dynamic values, it can be used for real time computation.

The project enabled to work with mid-level. OpenGL complexity and the project demonstrates the scope of OpenGL platform as a premier game developing launch pad. Hence, it has indeed been useful in developing many games. OpenGL in its own right is good for low cost and simple game development. It serves as an important stepping stone for venturing into other fields of Computer Graphics design and applications.