# CHAPTER 1

# INTRODUCTION TO NETWORK ROUTING PROTOCOL VISUALIZTION

Our project "Network Routing Protocols Visualization" is an interactive simulation program depicting the working of bellman ford algorithm and networking related specifications. In this project, we have basically three modes of operation which are visualisation mode, link failure mode, buffer mode. In the visualisation mode of operation we show the movement of packets for a given single source and for multiple destinations according to bellman ford algorithm.

In link failure mode, we allow the user to break any of the links through keyboard interaction and according to the user input the link will be failed and the operation continues in visualisation mode that is the movement of packets and here the link which the user gives can fail dynamically and then bellman ford algorithm finds the new shortest path.

In buffer mode of operation we try to show the real time scenario of the networking protocols that is the packet will be huge in size, so it cannot be sent in a single transmission from a given source to destination so we divide the packets into fixed size fragments and each fragment is sent individually to given source to destination and in this mode for the storage of fragments we keep a buffer of fixed size and if it exceeds then the packet from the source is dropped that is the buffer overflow occurs. In overall in this project we try to show the working of network protocols using bellman ford algorithm and give a clear picture of network specifications.

# CHAPTER 2

# SYSTEM REQUIREMENTS

## 2.1 HARDWARE REQUIREMENTS

The Hardware requirements are very minimal and the program can be run on most of the machines.

| | |
|---|---|
| Processor | : Intel Core i2 |
| Processor Speed | : 2.20 GHZ |
| RAM | : 4 GB |
| Storage Space | : 500 GB |
| Monitor Resolution | : 1024*768 or 1336*768 or 1280*1024 |

## 2.2 SOFTWARE REQUIREMENTS

| | |
|---|---|
| Operating System | : Windows 7 |
| IDE | : Microsoft Visual Studio 2008/2010 |

# CHAPTER 3

# FUNCTION DESCRIPTIONS

The description of all the function used in the program is given below:

- **Void glutInitDisplayMode(unsigned int mode)**

  This function requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE).

- **Void glutInitWindowPosition(int width,int height)**

  This specifies the initial position of top-left corner of the windows in pixels.

- **Void glutCreateWindow (char *title)**

  This function creates 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 glutdisplayFunc(void(*func)(void))**

  This function registers the display function that is executed when the window needs to be redrawn.

- **Void glClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a)**

  This sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating point numbers between 0 and 1.

- **Void glClear(GLbitfield mask)**

It clears buffers to present values. The value of mask is determined by the bitwise OR of options GL_COLOR_BUFER_BIT or GL_DEPTH_BUFFER_BIT.

- **Void glutPostRedisplay()**

This function requests that the callback be executed after the current callback returns.

- **Void glutReshapeFunc(int width,int height)**

This function registers the reshape callback function. The call back function returns the height and width of the new window. The reshape callback invokes a display callback.

- **Void glviewport (int x,int y, GLsizei width, Glsizei height)**

This function specifies a width* height viewport in pixels whose lower left corner is at(x,y) measured from the origin of the window.

- **Void glMatrixMode(GLenum mode)**

This function specifies which matrix will be affected by subsequent transformations.Mode can be GL_MODEL_VIEW, GL_PROJECTION, GL_TEXTURE

- **Void glLoadIdentity()**

This function sets the current transformation matrix to an identity matrix.

- **Void gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)**

This function defines two dimensional viewing rectangle in the plane z=0.

- **Void glutMouseFunc ( int button,int state,int x,int y)**

  This function registers the mouse call back function.The function returns the state of the button , the button pressed and the position of the mouse relative to the top left corner of the window.

- **Void glVertex3f(TYPE xcoordinate, TYPE ycoordinate, TYPE zcoordinate)**

This specifies the position of a vertex in 3 dimensions. If v is present, the argument is a pointer to an array containing the coordinates.

- **Void glBegin(glEnum mode)**

This function initiates a new primitive of type mode and starts the collection of vertices.

- **Void glEnd()**

  This function termiates a list of vertices.

- **Void glutMainLoop()**

This function cause the program to enter an infinite loop. It should be the last statement in the main.

# Chapter 4

# SOURCE CODE

```
#pragma once
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<stdio.h>
#include<GL\glut.h>
using namespace std;
#define PI 3.142857
#define LNW 2
int Gtemp[20][20];
int costcoord[100][2] =
{{45,260},{230,460},{120,350},{120,170},{230,40},{300,330},{450,345},{300,180},{
480,170} };
char seq[2], ed[20][2], g1[20][20], buf[256][2],buf[256][2];
int link1, link2;
int fps = 100;
float x, y;
int flag = 0,fbuf = 0,ind = 0,lnf = 0;
int pathbuf[100], destbuf, vert[10][2][2] = { {{10,510},{40,510}},{ { 10, 120 },{ 40,120
} }, { {410,510},{ 440,510 }},{ { 280,240 },{ 310,240 } },{ { 440,30 },{ 470,30 } },{ {
550,130 },{ 580,130 } } };
int fragcount=0,sta=1;
int presentationmode = 1, normalmode = 0, interactivemode = 0, sourcemode = 0,
linkbreakmode = 0,buffermode=0,helpmode=0;
int dest, S;
GLubyte linehcolor[3] = { 131, 255, 43 };
GLubyte srcnodecolor[3] = { 229, 199, 199 };
GLubyte destnodecolor[3] = { 224, 141, 40 };
GLubyte srctextcolor[3] = { 16, 232, 145 };
```

```
GLfloat node_rad = 30;
int n;
int cost[100][100];
int mypath[100][100], pathhighlight[100][100];
GLfloat x_packet_size = 25, y_packet_size = 25, h_length = 20;
GLfloat node_cord[10][2] = { { 60, 450 },{ 60, 60 },{ 380, 450 },{ 220, 270 },{ 380, 60
},{ 540, 270 } };
GLfloat packet_cord[20];
int V, edge[20][2], i, j, E = 0;
int G[20][20];
int packet1;
int path[50][50];
int sz[10];
int Bellman_Ford(const int G[20][20], int V, int E, int edge[20][2])
{
	int i, u, v, k, distance[20], parent[20], flag = 1, m = 0, dummy[20], j, size[6] = { 0
};
	for (i = 0; i<V; i++)
		distance[i] = 1000, parent[i] = -1;
	distance[S - 1] = 0;
	for (i = 0; i<V - 1; i++)
	{
		for (k = 0; k<E; k++)
		{
			u = edge[k][0], v = edge[k][1];
			if (distance[u] + G[u][v] < distance[v])
				distance[v] = distance[u] + G[u][v], parent[v] = u;
		}
	}
	for (k = 0; k<E; k++)
	{
		u = edge[k][0], v = edge[k][1];
		if (distance[u] + G[u][v] < distance[v])
```

```
                flag = 0;
        }
        if (flag)
                for (i = 0; i<V; i++)
                        printf("Vertex %d -> cost = %d parent = %d\n", i + 1, distance[i],
                        parent[i] + 1);
        for (i = 0; i<V; i++) { dummy[i] = parent[i] + 1; }
        for (i = 0; i<V; i++)
        {
                m = 0;
                path[i][m++] = i + 1;
                size[i]++;
                if ((parent[i] + 1) == S)
                {
                        path[i][m++] = S; size[i]++;
                }
                else if ((parent[i] + 1) == 0)
                {
                        path[i][m++] = 0; size[i]++;
                }
                else {
                        while (1)
                        {
                                path[i][m++] = dummy[i];
                                size[i]++;
                                if (dummy[i] == S)break;
                                dummy[i] = dummy[dummy[i] - 1];
                        }
                        for (j = 0; j<V; j++) { dummy[i] = parent[i] + 1; }
                }
        }
        for (i = 0; i<V; i++)
        {
```

```
                printf("\npath %d:  ", i + 1);
                for (j = 0; j<size[i]; j++)
                        printf("%d<---", path[i][j]);
        }
        for (i = 0; i < V; i++)
                sz[i] = size[i];
        m = 0;
        for (i = 0; i<V; i++)
        {
                m = 0;
                for (j = size[i] - 1; j >= 0; j--) {
                        mypath[i][m] = path[i][j];
                        pathhighlight[i][m++] = path[i][j];
                }
        }
        printf("\n\nMYPATH\n\n");
        for (i = 0; i<V; i++)
        {
                printf("\npath %d size=%d:  ", i + 1, sz[i]);
                for (j = 0; j<size[i]; j++)
                        printf("%d\t", mypath[i][j]);
        }
        return flag;
}

//project.cpp
#include "inc.h"
float xm = 10, ym = 20, xmm = 300, ymm = 65;glBegin(GL_POLYGON);

void link()
{
        g1[link1][link2] = 0;
        g1[link2][link1] = 0;
```

```
        for (int i1 = 0; i1<18; i1++)
        {
                if (edge[i1][0] == link1&&edge[i1][1] == link2)
                        continue;
                else if (edge[i1][0] == link2&&edge[i1][1] == link1)
                        continue;
                else {
                        ed[i1][0] = edge[i1][0];
                        ed[i1][1] = edge[i1][1];
                }
        }
        Bellman_Ford(g1, V, E - 2, ed);
}
void calc()
{

        int br = 1, l, q;
        cout << flag << "----" << ind << endl;
        if (flag == 0)
        {

                for (l = 0; l < 6; l++)
                {
                        for (q = 0; q < sz[l] - 1; q++)
                        {
                                if (l + 1 == S)
                                {
                                        break;
                                }
                                else if (mypath[l][q] == 0)
                                {
                                        continue;
                                }
```

```
                                else
                                {
                                        flag = mypath[l][q] * 10 + mypath[l][q + 1];
                                        mypath[l][q] = 0;
                                        br = 0;
                                        dest = path[l][0];
                                        ind = l;
                                        break;
                                }
                        }
                        if (br == 0)
                        {
                                break;
                        }
                }
        }
}
void timer(int val)
{

        glutTimerFunc(1000 / fps, timer, 100);
        //1--->3 (path movement)
        if (flag == 13 || flag == 25) {
                if (x <= node_cord[2][0] - x_packet_size - 25 - node_cord[0][0])
                {
                        x += 1;
                        y = 0;
                }
                else
                {
                        flag = 0;
                        x = 0;
                        y = 0;
```

```
		}
	}
	//3--->1
	if (flag == 31 || flag == 52) {
		if (abs(x) <= node_cord[2][0] - node_cord[0][0])
		{
			x -= 1;
			y = 0;
		}
		else
		{
			flag = 0;
			x = 0;
			y = 0;
		}
	}
	//1--->4
	if (flag == 14)
	{
		if (x <= node_cord[3][0] - node_cord[0][0])
		{
			x += 1;
				y = ((-9 * (x + node_cord[0][0]) / 8) + (1035 / 2)) -
	node_cord[0][1];
		}
		else
		{
			flag = 0;
			x = 0;
			y = 0;
		}
	}
	//4--->1
```

```
        if (flag == 41)
        {
                if (abs(x) <= node_cord[3][0] - node_cord[0][0])
                {
                        x -= 1;
                        y = ((-9 * (x + node_cord[0][0]) / 8) + (1035 / 2)) -
node_cord[0][1];
                }
                else
                {
                        flag = 0;
                        x = 0;
                        y = 0;

                }
        }
        //1--->2
        if (flag == 12)
        {
                if (abs(y) <= node_cord[0][1] - node_cord[1][1])
                {
                        y -= 1;
                        x = 0;
                }
                else
                {
                        x = 0;
                        y = 0;
                        flag = 0;
                }
        }
        //2--->1
        if (flag == 21)
```

```
{
        if (y <= node_cord[0][1] - node_cord[1][1])
        {
                y += 1;
                x = 0;
        }
        else
        {
                x = 0;
                y = 0;
                flag = 0;

        }
}
//2--->4
if (flag == 24)
{
        if (x <= node_cord[3][0] - node_cord[1][0])
        {
                x += 1;
                y = ((21 * (x + node_cord[1][0]) / 16) - (75 / 4)) - node_cord[1][1];
        }
        else
        {
                x = 0;
                y = 0;
                flag = 0;

        }
}
//4--->2
if (flag == 42)
{
```

```
        if (abs(x) <= node_cord[3][0] - node_cord[1][0])
        {
                x -= 1;
                y = ((21 * (x + node_cord[1][0]) / 16) - (75 / 4)) - node_cord[1][1];
        }
        else
        {
                x = 0;
                y = 0;
                flag = 0;

        }
}
//3--->4
if (flag == 34)
{
        if (abs(x) <= node_cord[2][0] - node_cord[3][0])
        {
                x -= 1;
                y = ((9 * (x + node_cord[2][0]) / 8) + (45 / 2)) - node_cord[2][1];
        }
        else
        {
                x = 0;
                y = 0;
                flag = 0;

        }
}

//4--->3
if (flag == 43)
{
```

```
                if (abs(x) <= node_cord[2][0] - node_cord[3][0])
                {
                        x += 1;
                        y = ((9 * (x + node_cord[3][0]) / 8) + (45 / 2)) - node_cord[3][1];
                }
                else
                {
                        x = 0;
                        y = 0;
                        flag = 0;
                }
        }
        //4--->5
        if (flag == 45)
        {
                if (abs(x) <= node_cord[4][0] - node_cord[3][0])
                {
                        x += 1;
                        y = ((-21 * (x + node_cord[3][0]) / 16) + (2235 / 4)) -
node_cord[3][1];
                }
                else
                {
                        x = 0;
                        y = 0;
                        flag = 0;

                }
        }
        //5--->4
        if (flag == 54)
        {
                if (abs(x) <= node_cord[4][0] - node_cord[3][0])
```

```
                {
                        x -= 1;
                        y = ((-21 * (x + node_cord[4][0]) / 16) + (2235 / 4)) -
node_cord[4][1];
                }
                else
                {
                        x = 0;
                        y = 0;
                        flag = 0;

                }
        }
        //3--->6
        /*if (flag == 36) {
        if (abs(x) <= node_cord[5][0] - node_cord[2][0])
        {
        x += 1;
        y = ((-9 * (x + node_cord[2][0]) / 8) + (1755 / 2)) - node_cord[2][1];
        }
        else
        {
        x = 0;
        y = 0;
        flag = 0
        }
        }
        */
        //6--->3
        if (flag == 63) {
                if (abs(x) <= node_cord[5][0] - node_cord[2][0])
                {
                        x -= 1;
```

```
                        y = ((-9 * (x + node_cord[5][0]) / 8) + (1755 / 2)) -
node_cord[5][1];
                }
                else
                {
                        x = 0;
                        y = 0;
                        flag = 0;
                }
        }

        //3--->6
        if (flag == 36) {
                if (abs(x) <= node_cord[5][0] - node_cord[2][0])
                {
                        x += 1;
                        y = ((-9 * (x + node_cord[2][0]) / 8) + (1755 / 2)) -
node_cord[2][1];
                }
                else
                {
                        x = 0;
                        y = 0;
                        flag = 0;
                }
        }
        //5--->6
        if (flag == 56) {
                if (abs(x) <= node_cord[5][0] - node_cord[4][0])
                {
                        x += 1;
                        y = ((21 * (x + node_cord[4][0]) / 16) - (1755 / 4)) -
node_cord[4][1];
```

```
                }
                else
                {
                        x = 0;
                        y = 0;
                        flag = 0;
                }
        }

        //6--->5
        if (flag == 65) {
                if (abs(x) <= node_cord[5][0] - node_cord[4][0])
                {
                        x -= 1;
                        y = ((21 * (x + node_cord[5][0]) / 16) - (1755 / 4)) -
node_cord[5][1];
                }
                else
                {
                        x = 0;
                        y = 0;
                        flag = 0;
                }
        }
        glutPostRedisplay();
}
void disppk()
{
        /*if (nodef==0) {
        glColor3ubv(srcnodecolor);
        circle_draw(node_cord[S - 1][0], node_cord[S - 1][1], node_rad);
        char number[2];
        _itoa_s(S, number, 5);
```

```
drawtext(node_cord[S-1][0], node_cord[S-1][1], number);
nodef = 1;
}*/
if (flag == 54 || flag == 52 || flag == 56)
{
        glPushMatrix();
        glTranslatef(x, y, 0.0);
        packet(5);
        glPopMatrix();
}
if (flag == 12 || flag == 14 || flag == 13)
{
        glPushMatrix();
        glTranslatef(x, y, 0.0);
        packet(1);
        glPopMatrix();

}
if (flag == 21 || flag == 24 || flag == 25)
{
        glPushMatrix();
        glTranslatef(x, y, 0.0);
        packet(2);
        glPopMatrix();

}

if (flag == 31 || flag == 34 || flag == 36)
{
        glPushMatrix()
        glTranslatef(x, y, 0.0);
        packet(3);
        glPopMatrix();
```

```
        }

        if (flag == 63 || flag == 65)
        {
                glPushMatrix();
                glTranslatef(x, y, 0.0);
                packet(6);
                glPopMatrix();
        }

        if (flag == 41 || flag == 42 || flag == 43 || flag == 45)
        {
                glPushMatrix();
                glTranslatef(x, y, 0.0);
                packet(4);
                glPopMatrix();

        }
}
void bufcalc()
{
        int q;
        cout << flag << endl;
        if (flag == 0)
        {
                for (q = 0; q < sz[destbuf - 1] - 1; q++)
                {

                        if (pathbuf[q] == 0)
                        {
                                continue;
                        }
```

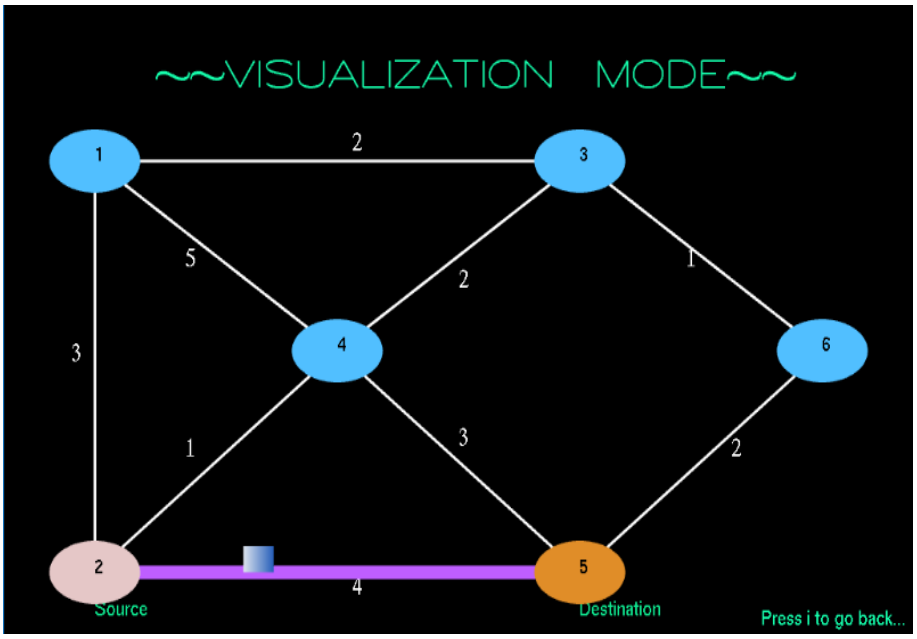
```
                else
                {
                        flag = pathbuf[q] * 10 + pathbuf[q + 1];
                        pathbuf[q] = 0;
                        ind = 0;
                        //dest = path[l][0];
                        //ind = l;
                        break;
                }
        }
        if (pathbuf[sz[destbuf - 1] - 2] == 0)
                sta = 0;
    }
}
void linehighlight()
{
    glLineWidth(13);
    switch (ind)
    {
    case 0:
        for (int i1 = 0; i1 < sz[ind] - 1; i1++)
        {
            glBegin(GL_LINES);//1-3
            glVertex2fv(node_cord[pathhighlight[0][i1] - 1]);
            glVertex2fv(node_cord[pathhighlight[0][i1 + 1] - 1]);
            glEnd();
        }
        break;
    case 1:
        for (int i1 = 0; i1 < sz[ind] - 1; i1++)
        {
            glBegin(GL_LINES);//1-3
            glVertex2fv(node_cord[pathhighlight[1][i1] - 1]);
```

```
                glVertex2fv(node_cord[pathhighlight[1][i1 + 1] - 1]);
                glEnd();
        }
        break;
case 2:
        for (int i1 = 0; i1 < sz[ind] - 1; i1++)
        {
                glBegin(GL_LINES);//1-3
                glVertex2fv(node_cord[pathhighlight[2][i1] - 1]);
                glVertex2fv(node_cord[pathhighlight[2][i1 + 1] - 1]);
                glEnd();
        }
        break;
case 3:
        for (int i1 = 0; i1 < sz[ind] - 1; i1++)
        {
                glBegin(GL_LINES);//1-3
                glVertex2fv(node_cord[pathhighlight[3][i1] - 1]);
                glVertex2fv(node_cord[pathhighlight[3][i1 + 1] - 1]);
                glEnd();
        }
        break;
case 4:
        for (int i1 = 0; i1 < sz[ind] - 1; i1++)
        {
                glBegin(GL_LINES);//1-3
                glVertex2fv(node_cord[pathhighlight[4][i1] - 1]);
                glVertex2fv(node_cord[pathhighlight[4][i1 + 1] - 1]);
                glEnd();
        }
        break;
case 5:
        for (int i1 = 0; i1 < sz[ind] - 1; i1++)
```

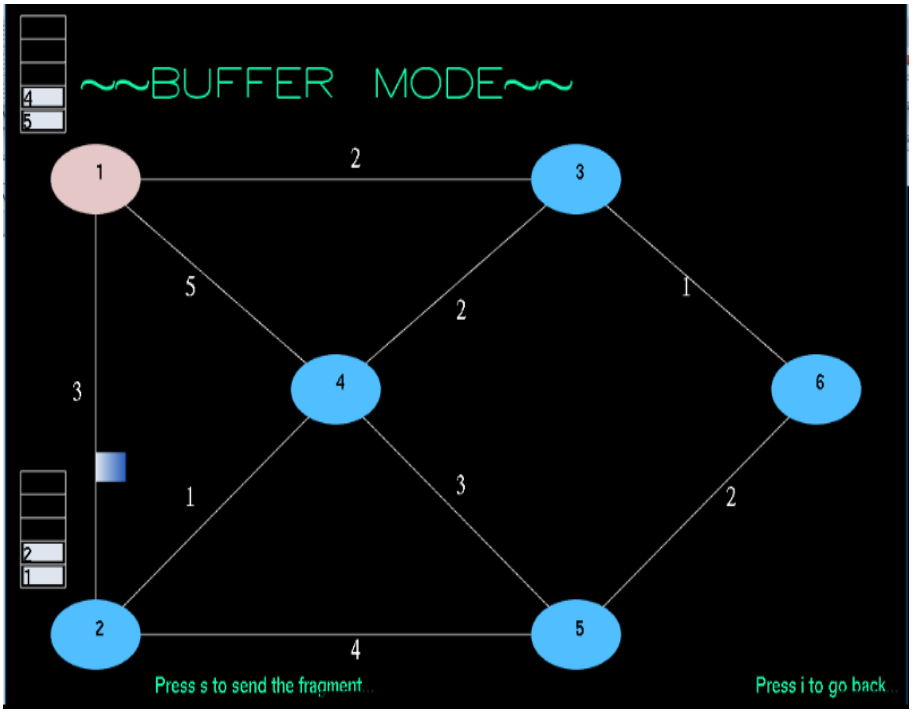
```
                {
                        glBegin(GL_LINES);//1-3
                        glVertex2fv(node_cord[pathhighlight[5][i1] - 1]);
                        glVertex2fv(node_cord[pathhighlight[5][i1 + 1] - 1]);
                        glEnd();
                }
                break;
        }
        glLineWidth(1);
}
```

# Chapter 5

# PROJECT SNAPSHOTS




**1.VISUALIZATION MODE**




**2.BUFFER MODE**

# Chapter 6

# CONCLUSION

Perhaps the dominant characteristics of this millennium is how computer and communication technologies have become dominant forces in our lives. Activities as wide ranging as filmmaking, publishing ,banking and education continue to undergo revolutionary changes as these technologies alter the ways in which we conduct out daily activities. The combination of computers, networks and the complex human visual system through computer graphics has led to new ways of displaying information, seeing virtual worlds and communicating with people and machines since the project developed is mainly academics based, the project consists of only a few functions and implementations of few concepts.

This mini project is developed using openGL software ad so far has proved to be having good performance. The user can easily understand the mini project. Over all, the mini project was good experience and provides knowledge of working on OpenGL.

# Chapter 7

# REFERENCES

[1] Interactive Computer Graphics A Top-Down Approach with OpenGL- Edward Angel,5th Edition,Addison-Wesley,2008

[2] Edward Angel, Interactive Computer Graphics, A Top-Down Approach Using OpenGL, 5th edition

[3] OpenGL Architecture Review Board, Dave Shreiner, Opengl® Reference Manual,4th edition

[3] OpenGL ® SuperBible: Comprehensive Tutorial and Reference, by Richard S. Wright (Author), Benjamin Lipchak (Author), Nicholas Haemel (Author), 4th edition

[4] www.wikipedia.org

[5] www.stackoverflow.com