1.Introduction

1.1 Problem Definition:

To generate a user interface to implement dijkstra's algorithm to find single source shortest path for a graph.

Definition. A graph G consists of a vertex set V and an edge set E. Every element of E is an unordered pair of vertices. We will write undirected edges as {u,v}.

Definition. A path in G is a sequence of vertices v0; v1; v2; : : : ; vk such that there is an edge between any two adjacent vertices vi ; vi+1 in the sequence. We will sometimes refer to the edges in the path, although it is formally defined as a sequence of vertices.

Definition. A graph is connected if there is a path between any two vertices in the graph.

Definition. A tree is a graph that is minimally connected – that is, any tree T is a connected graph, but removing any edge will disconnect it.

Definition. A spanning tree of G is a subgraph of G that is a tree and that covers every vertex in G.

Definition. A forest is a set of trees.

In computer science, a graph is a kind of data structure, specifically an abstract data type

((ADT), that consists of a set of nodes and a set of edges that establish relationships (connections) between the nodes. The graph ADT follows directly from the graph concept from mathematics.
Informally, G=(V,E) consists of vertices, the elements of V, which are connected by edges, the elements of E.

Formally, a graph, G, is defined as an ordered pair, G=(V,E), where V is a finite set and E is a set consisting of two element subsets of V.

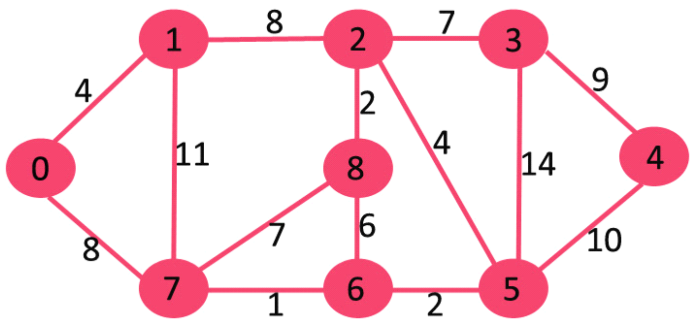


Fig 1.1 Example of a graph

It’s a set of nodes(0,1,2,3,4,5,6,7 and 8) and the edges(lines) that interconnect them.

An important thing to note about this graph is that the edges are bidirectional, i.e. if A is connected to B, then B is connected to A. This makes it an undirected graph.

A common extension is to attribute weights to the edges.

Single source shortest path:

Dijkstra’s Algorithm

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

- Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.

- Mark all nodes unvisited. Set the initial node as current. Create a set of the unvisited nodes called the *unvisited set* consisting of all the nodes except the initial node.

- For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be 6+2=8. If this distance is less than the previously recorded tentative distance of B, then overwrite that distance. Even though a neighbor has been examined, it is not marked as "visited" at this time, and it remains in the *unvisited set*.

- When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.

- If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal), then stop. The algorithm has finished.

- Select the unvisited node that is marked with the smallest tentative distance, and set it as the new "current node" then go back to step 3.

For the example graph, here’s how it would run:

Start with only node 0 in the tree.

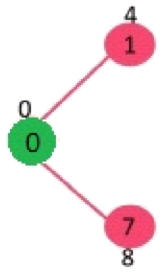


Fig 1.2

Find the closest node to the tree, and add it.

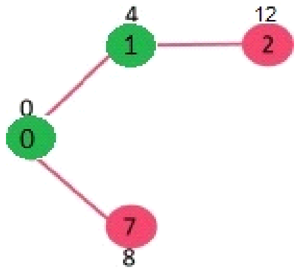


Fig 1.3

Repeat until there are n – 1 edges in the tree.

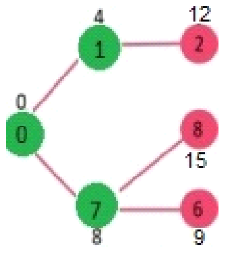


Fig 1.4

All vertices have been visited

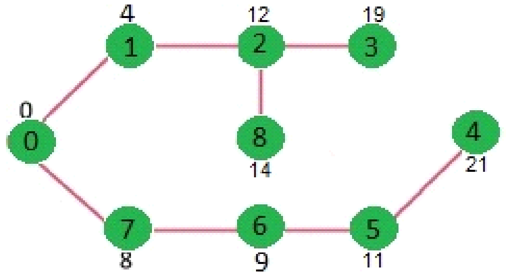


Fig 1.6

1.2 About OpenGl:

OpenGL provides the programmer with an interface to graphics hardware. It is a powerful, low-level rendering and modelling software library, available on all major platforms, with wide hardware support. It is designed for use in any graphics applications, from games to modelling in CAD.

OpenGL intentionally provides only low-level rendering routines, allowing the programmer a great deal of control and flexibility. The provided routines can easily be used to build high-level rendering and modelling libraries, and in fact, the OpenGL Utility Library (GLU), which is included in most OpenGL distributions, does exactly that. Note also that OpenGL is just a graphics library; unlike DirectX, it does not include support for sound, input, networking, or anything else not directly related to graphics.

1.3 The OpenGL Utility Library:

The OpenGL Utility Library, or GLU, supplements OpenGL by providing higher-level functions. GLU offers features that range from simple wrappers around OpenGL functions to complex components supporting advanced rendering techniques. Its features include:

- 2D image scaling
- Rendering 3D objects including spheres, cylinders, and disks
- Automatic bitmap generation from a single image
- Support for curves surfaces through NURBS
- Support for tessellation of non-convex polygons
- Special-purpose transformations and matrices

1.4 What is GLUT?

GLUT, short for OpenGL Utility Toolkit, is a set of support libraries available on every major platform. OpenGL does not directly support any form ofwindowing, menus, or input. That's where GLUT comes in. It provides basic functionality in all of those areas, while remaining platform independent, so that you can easily move GLUT-based applications from, for example, Windows to UNIX with few, if any, changes.

GLUT is easy to use and learn, and although it does not provide you with all the functionality the operating system offers, it works quite well for demos and simple applications.

1.5 OpenGL architecture

OpenGL is a collection of several hundred functions providing access to all the features offered by your graphics hardware. Internally, it acts as a state machine--a collection of states that tells OpenGL what to do. Using the API, you can set various aspects of the state machine, including such things as the current colour, lighting, blending, and so on. When rendering, everything drawn is affected by the current settings of the state machine. It's important to be aware of what the various states are, and the effect they have, because it's not uncommon to have unexpected results due to having one or more states set incorrectly.

At the core of OpenGL is the rendering pipeline, as shown in Figure 1.1. You don't need to understand everything that happens in the pipeline at this point, but you should at least be aware that what you see on the screen results from a series of steps. Fortunately, OpenGL handles most of these steps for you.

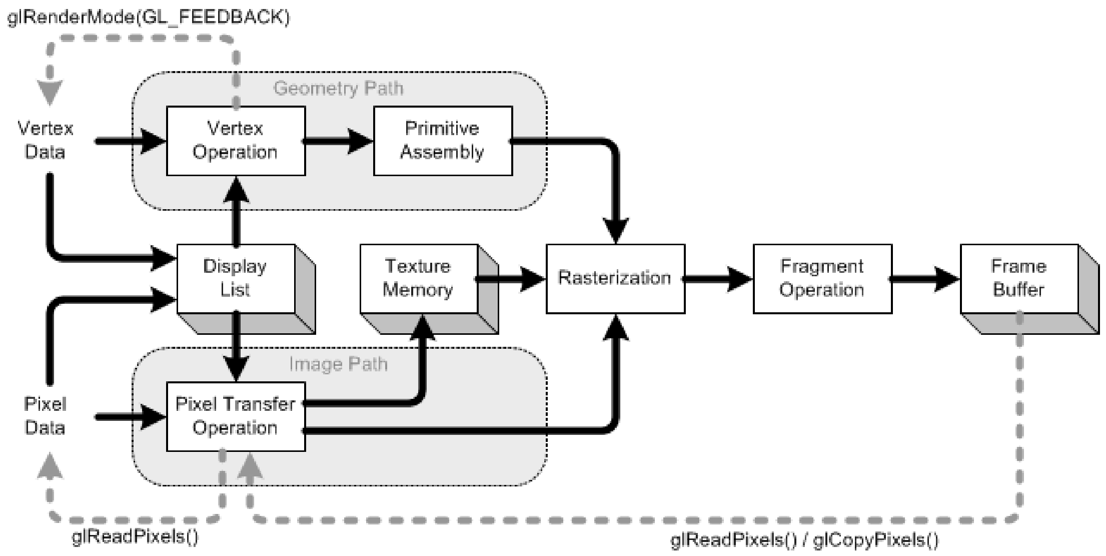


Fig 1.1 The OpenGL rendering pipeline.

Under Windows, OpenGL provides an alternative to using the Graphics Device Interface (GDI). GDI architects designed it to make the graphics hardware entirely invisible to Windows programmers. This provides layers of abstraction that help programmers avoid dealing with Device-specific issues. However, GDI is intended for use with applications and thus lacks the speed required for games. OpenGL allows you to bypass GDI entirely and deal directly with graphics hardware. Figure 1.1 illustrates the OpenGL hierarchy under Windows.

2.Requirements

2.1 Hardware requirements:

- Processor: Intel Pentium 3.0, 1.5 GHz
- RAM: 512 MB
- Monitor: 1024 * 768 display resolution , High Colors 16-bit
- Hard Disk Space: 4.0 GB
- DVD drive
- Keyboard and Mouse

2.2 Software requirements:

- An MS-DOS based operating system like Windows 98 or Window 2000 or Window XP is the platform required to develop the 2D or 3D graphics application.
- A Visual C/C++ compiler like Microsoft Visual Studio 2005 is required for compiling the source code to make the executable file which can then be directly executed.
- Microsoft Office for documentation and presentation of the mini graphics project.
- OpenGL , a software interface for graphics hardware with built in graphics libraries like glut and glut32, and header files like glut.h.

3. Implementation

Inbuilt functions:

glBegin

C Specification: void **glBegin** (Glenum mode);

Parameters: mode

Description: Specifies the primitive(s) that will be created from vertices present between glBegin and the subsequent glEnd. Ten symbolic constants are accepted.

glEnd

C Specification: void **glEnd()**;

Description: 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. Taking 'n' as an integer count starting at one, and 'N' as the total number of vertices specified, the interpretations are as follows:

GL_POINTS: Treats each vertex as a single point. Vertex n defines point n. N points are drawn.

GL_LINES: Treats each pair of vertices as an independent line segment. Vertices 2n-1 and 2n define line n. N/2 lines are drawn.

GL_LINE_STRIP: Draws a connected group of line segments from the first vertex to the last. Vertices n and n+1 define line n.N-1 lines are drawn.

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

GL_TRIANGLES: Treats each triplet of vertices as an independent triangle. Vertices 3n-2, 3n-1 and 3n define triangle n. N/3 triangles are drawn.

GL_TRIANGLE_STRIP: Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. For odd n, vertices, n+1 and n+2 define n. For even n, vertices n+1, n, and n+2 define triangle n. N-2 triangles are drawn.

GL_TRIANGLE_FAN: Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. Vertices 1, n+1, and n+2 define triangle n. N-2 triangles are drawn.

GL_QUADS: Treats each group of four vertices as an independent quadrilateral. Vertices 4n-3, 4n-2, 4n-1 and 4n define quadrilateral n. N/4 quadrilaterals are drawn.

GL_POLYGON: Draws a single, convex polygon. Vertices 1 through N define this polygon.

glClearColor

C Specification: void **glClearColor**(GLClamp red, GLClamp green, GLClamp blue, GLClamp alpha);

Parameters: red, green, blue, alpha specifies the red, blue, green and alpha values used when the color buffers are cleared, the initial values are all 10.

Description: 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, 1].

glClear

C specification: void**glClear**(GLbitfield mask);

Parameters: 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.

Description: glClear takes a single argument that is bitwise OR of several values indicating which buffer is to be cleared. The values are as follows: GL_COLOR_BUFFER_BIT-indicates the buffers currently enabled for color writing.

GL_DEPTH_BUFFER_BIT - indicates the depth buffer.
GL_ACCUM_BUFFER_BIT - indicates the accumulation buffer.

GL_STENCIL_BUFFER_BIT - indicates the stencil bit.

glFlush

C Specification: void **glFlush()**;

Description:glFlush empties all the buffers causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine.

glLoadIdentity

C specification: void **glLoadIdentity()**;

Description:glLoadIdentity replaces the current matrix with identity matrix.

glMatrixMode

C Specification: void**glMatrixMode**(Glenum mode);

Parameters: mode specifies which matrix stack is the target for subsequent matrix operations. Three values accepted are: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE.

Description: glMatrixMode sets the current matrix mode. mode can assume one of the following values:
GL_MODEL_VIEW applies subsequent matrix operations to the modelview matrix stack
GL_PROJECTION applies subsequent matrix operations to the projection matrix stack.

glOrtho

C Specification: void **glOrtho**(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdoubleznear,GLdoublezfar);

Description: specifies the coordinate system OpenGL assumes as it draws the final image and how the image gets mapped to the screen.

glVertex

C Specification: void glVertex{234}{sifd}[v](TYPEcoords);

Description:glVertex commands are used within glBegin/glEnd pairs to specify point, line and polygon vertices. It specifies a pointer to an array of two, three or four elements.

glViewport

C Specification: void **glViewport**(Glint x, Glint y, Glsizei width, Glsizei height);

Parameters:
-x, y specify the lower left corner of the viewport rectangle, in pixels.
-width, height specify the width and height of the viewport.

Description:glViewport specifies the affine transformation of x and from the normalized device coordinates to window coordinates.

glutBitmapCharacter

C Specification: void **glutBitmapCharacter**(void *font, int character);

Description: glutBitmapCharacter renders the character in the named bitmap font.

glutCreateWindow

C Specification: int**glutCreateWindow**(char *name);

Description: glutCreateWindow creates a top-level window. The name will be provided to the window's name.

glutDisplayFunc

C Specification: void **glutDisplayFunc** (void (*func)(void));

Description:glutDisplayFunc sets the display for the current window.

glutReshapeFunc

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

Description: glutReshapeFunc sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped.

glutKeyboardFunc

C Specification:void **glutKeyboardFunc**(void (*func)(char key, intx,inty));

Description: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.

glutMainLoop

C Specification: void **glutMainLoop**(void);

Description: glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in GLUT.

glutMouseFunc

C specification: void glutMouseFunc(void(*func)(int button, int state, int x, int y));

Description: glutMouseFunc set the mouse callback for the current window. When a user presses and releases the 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, and GLUT_RIGHT_BUTTON. The state parameter is either GLUT_UP or GLUT_DOWN indicating whether the callback was due to a release or press.

Code for user defined functions

display():

Display the output on the screen.

```
void display (void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush();
 }
```

Drawstring():

Display text on the screen

```
void drawstring(GLfloat x,GLfloat y,char *string)
{
  char *c;
 glRasterPos2f(x,y);
 for(c=string;*c!='\0';*c++)
  {
glutBitmapCharacter(currentfont,*c);
```

```
   }
}
```

drawsquare():

Draws a point or a node at the position (x,y) and is called by the mouse function.

```
voiddrawSquare(int x, int y)
{
            if(i<=n)
           {
                      y = 500-y;
                      glPointSize(25);
                      glColor3f(0.0f, 0.0f, 0.0f);
                      glBegin(GL_POINTS);
                      glVertex2f(x , y);
                      glEnd();
                      a[i]=x;
                      b[i]=y;
                      raster(x,y,i);
                      glutSwapBuffers();
           }
           i=i+1;
 }
```

drawline():

This function is used to connect the nodes based on the cost matrix by drawing an directed edges between them.

```
void drawline()
{
   int j,k,x1,x2,y1,y2;
            for(j=1;j<=n;j++)
            {
                       for(k=1;k<=n;k++)
                       {
                                  if(cost[j][k]!=999 && j<k)
                                  {
                                             x1=a[j];
                                             y1=b[j];
                                             x2=a[k];
                                             y2=b[k];
                                             glColor3f(0.0,0.5,0.0);
```

```
                                        glLineWidth(3);
                                        glBegin(GL_LINES);
                                        glVertex2i(x1-7,y1+10);
                                        glVertex2i(x2-7,y2+10);
                                        glEnd();
                                                s1=itoa(cost[j][k],s,10);
                                                drawstring((x1+x2-16)/2,(y1+y2+22)/2,s1);
                                        glFlush();
                                }
                                if(cost[j][k]!=cost[k][j] && cost[j][k]!=999 && j>k)
                                {
                                        x1=a[j];
                                        y1=b[j];
                                        x2=a[k];
                                        y2=b[k];
                                        glColor3f(1.0,0.5,0.0);
                                        glBegin(GL_LINES);
                                        glVertex2i(x1+10,y1+18);
                                        glVertex2i(x2+10,y2+18);
                                        glEnd();
                                        s1=itoa(cost[j][k],s,10);
                                        drawstring((x1+x2+20)/2,(y1+y2+36)/2,s1);
                                        glFlush();
                                }
                        }
                }
 }
```

Read():

Read number of nodes and cost matrix.

```
void read()
 {
printf("enter the number of nodes\n");
        scanf("%d",&n);
        printf("enter the cost matrix\n");
        for(int j=1;j<=n;j++)
                for(int k=1;k<=n;k++)
                {
                        scanf("%d",&cost[j][k]);
                        if(cost[j][k]==0)
```

```
                                        cost[j][k]=999;
                    }
      printf("enter the source\n");
          scanf("%d",&src);
 }
```

Shortestpath():

Generates the shortest path for a source vertex for the given graph.

```
void shortestpath()
{

          int w,j,u,v,k,p,q,x1,y1,x2,y2,x,y;
          int dist[MAX],visit[MAX],parent[MAX],mincost,min;
          for(w=1;w<=n;w++)
          {
                    visit[w]=0;
                    dist[w]=cost[src][w];
                    parent[w]=src;
          }
          visit[src]=1;
          mincost=0;
          k=1;
          for(w=1;w<n;w++)
          {
                    min=999;
                    u=-1;
                    for(j=1;j<=n;j++)
                    {
                              if(!visit[j]&&dist[j]<min)
                              {
                                        min=dist[j];
                                        u=j;
                              }
                    }
                    visit[u]=1;
                    mincost=mincost+dist[u];
                    tree[k][1]=parent[u];
                    tree[k++][2]=u;
                    for(v=1;v<=n;v++)
                    {
                              if(!visit[v]&&(cost[u][v]+dist[u]<dist[v]))
                              {
```

```
                                        dist[v]=dist[u]+cost[u][v];
                                        parent[v]=u;
                              }
                    }
          }


          for(int r=1;r<=n;r++)
          {
                     x=a[r];
              y=b[r];
          glPointSize(25);
          if(r==src)
          glColor3f(0.7f, 0.4f, 0.0f);
          else
   glColor3f(0.5f, 0.5f, 0.8f);
          glBegin(GL_POINTS);
          glVertex2f(x,y+250);
          glEnd();
          glColor3f(0.0,1.0,0.0);
          s1=itoa(r,s,10);
          drawstring(x,y+250,s1);
          glFlush();



           for(int x=1;x<n;x++)
          {
                    p=tree[x][1];
                    q=tree[x][2];
                    x1=a[p];
                    y1=b[p];
                    x2=a[q];
                    y2=b[q];
                    if(p<q)
                    {
                              glColor3f(0.0,0.5,0.0);
                    glBegin(GL_LINES);
                                        glVertex2i(x1,y1+250);
                                        glVertex2i(x2,y2+250);
                                        glEnd();
                                        s1=itoa(cost[p][q],s,10);
                                        drawstring((x1+x2)/2,(y1+y2+500)/2,s1);
                    }
                    else
                    {
```

```
            glColor3f(1.0,0.5,0.0);
                        glBegin(GL_LINES);
                                    glVertex2i(x1,y1+250);
                                    glVertex2i(x2,y2+250);
                                    glEnd();
                                    s1=itoa(cost[p][q],s,10);
                                    drawstring((x1+x2)/2,(y1+y2+500)/2,s1);
                 }
         }
                                    glFlush();
        }
 }
```

mouse():

This function is called on a left click of a mouse button on the graphics window. It generates a point (x,y) which is then sent to drawSquare() function to generate a node at that particular position.

```
void mouse(int bin, int state , int x , int y)
{
        if(bin==GLUT_LEFT_BUTTON&&state==GLUT_DOWN)
        drawSquare(x,y);
}
```

topmenu():

It is used to create a menu and based on the users selection the corresponding set of functions are called.

```
void top_menu(int option)
{

        switch(option)
        {
        case 1:read();
                glutPostRedisplay();
                  break;
        case 2:drawline();
                glutPostRedisplay();
       break;
        case 3:shortestpath();
                glutPostRedisplay();
                  break;
        case 4:exit(0);
        }
}
```

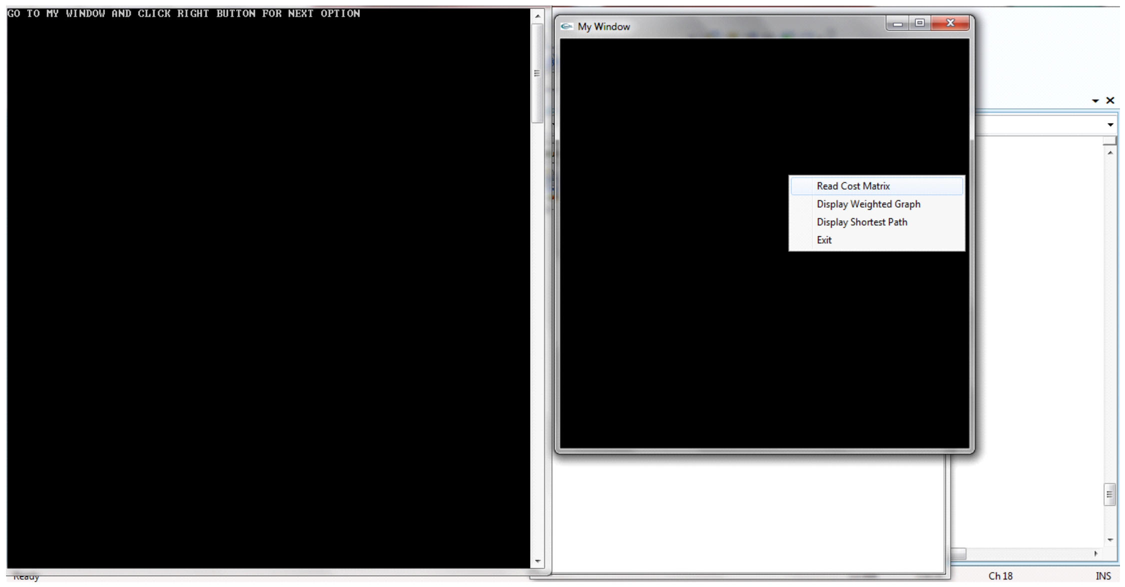
init():

Used for initializing,setting the background,foregroundcolor.
Mapping of the image to the window.

```
void init (void)
{
	glClearColor (1.0, 1.0, 1.0, 1.0);
	glClear(GL_COLOR_BUFFER_BIT);
	glViewport( 0,0, 500, 500 );
	glMatrixMode( GL_PROJECTION );
	glOrtho( 0.0, 500.0, 0.0, 500.0, 1.0, -1.0 );
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	glFlush();
}
```

4. Results & Snapshots

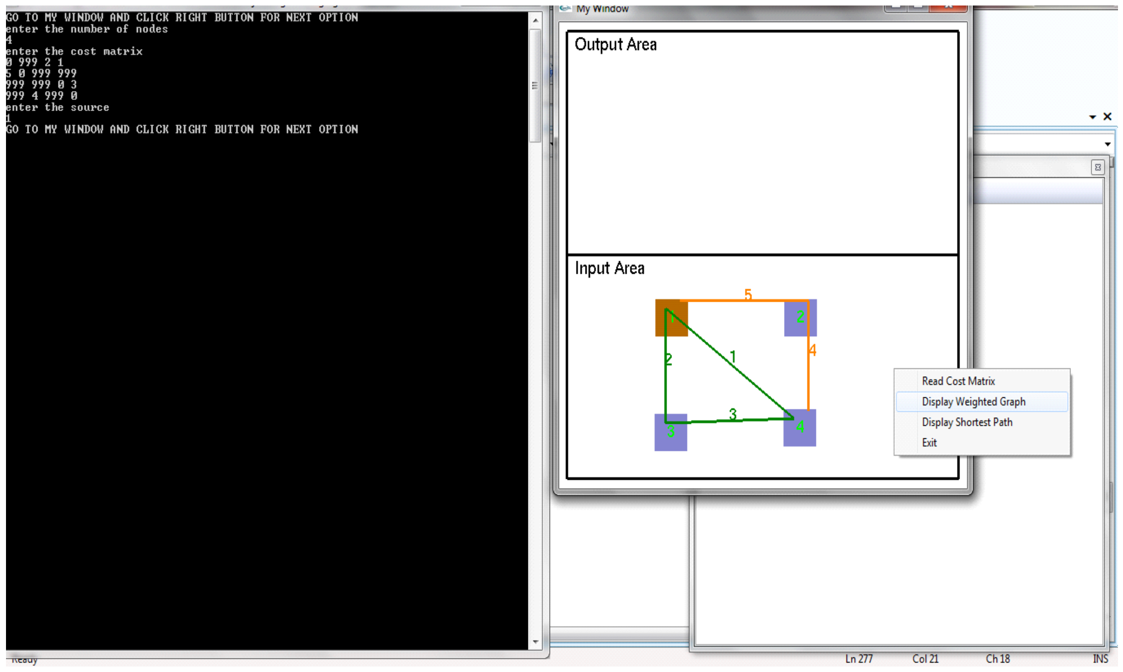
The dijkstra’s algorithm is successfully implemented and has been made interactive.



```
GO TO MY WINDOW AND CLICK RIGHT BUTTON FOR NEXT OPTION
```

Fig 4.1 Menu for selecting options

First the user has to decide the number of nodes to be used. Then the user is prompted to enter the cost matrix for the graph.



```
GO TO MY WINDOW AND CLICK RIGHT BUTTON FOR NEXT OPTION
enter the number of nodes
4
enter the cost matrix
0 999 2 1
5 0 999 999
999 999 0 3
999 4 999 0
enter the source
1
GO TO MY WINDOW AND CLICK RIGHT BUTTON FOR NEXT OPTION
```

Fig 4.2 Entering the cost matrix and displaying weighted graph

After the cost matrix is entered, the user can see a window. Left mouse button clicks in this window determines the position of the nodes. Then select Display weighted graph.

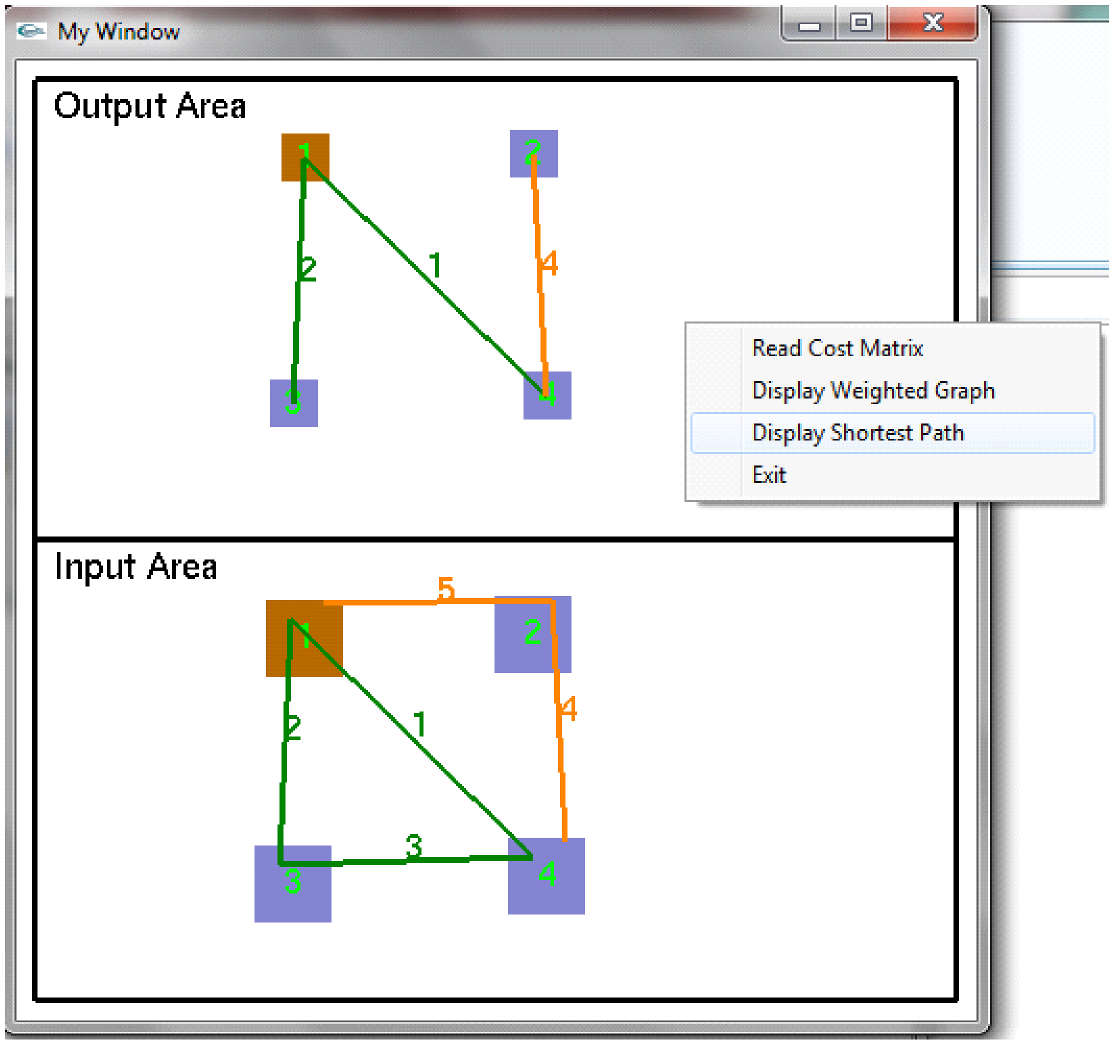


Fig 4.3 Displaying the shortest path

The shortest path from the source vertex to all other vertices is displayed along with the cost in the output area.

.

5. Conclusion and future enhancements

The project demonstrates how openGL can be used to implement Dijkstra’s Algorithm which is used to find single source shortest path for the given graph. This product has been demonstrated to fulfill the requirements.

Limitations would be that the project is entirely in 2 dimensional.

Further we could make a 3 dimensional implementation of the same.

6. Bibliography

Books:

Edward Angel : Interactive Computer Graphics A Top-Down ApproachUsing OpenGL, Pearson Education, 5th Edition.

F. S. Hill, Jr : Computer Graphics Using OpenGL Pearson Education, 3rd Edition.

Foley VanDamFeiner Hughes: Computer Graphics Principles & Practice, Pearson Education, 2nd Edition in C.

Donald Hearn, M. Pauline Baker : Computer Graphics 'C' version Pearson Education, 2nd Edition.

Websites:

http://www.opengl.org/

http://www.geeksforgeeks.org

http://compprog.wordpress.com

http://homes.cs.washington.edu