

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**  
**Jnana Sangama, Belgaum-590018**



**A Computer Graphics & Visualization Mini Project Report  
on**  
**“PRIM’S ALGORITHM”**

Submitted in Partial fulfillment of the Requirements for VI Semester of the Degree of

**Bachelor of Engineering  
In  
Computer Science & Engineering  
By  
MADHUSOODHANAN KM  
(1CR15CS089)**

**RISHABH D  
(1CR15CS129)**

**Under the Guidance of  
Mr.Kartheek GCR  
Asst. Professor, Dept. of CSE**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
CMR INSTITUTE OF TECHNOLOGY**

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI,  
BANGALORE-560037

# **CMR INSTITUTE OF TECHNOLOGY**

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI,  
BANGALORE-560037

## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



## **CERTIFICATE**

This is to certify that the Computer Graphics & visualization project work entitled "**Prim's Algorithm**" has been carried out by **Madhusoodhanan KM (1CR15CS089)** and **Rishabh D (1CR15CS129)** bonafide students of CMR Institute of Technology in partial fulfillment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year **2017-2018**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. This CG project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

---

### **Signature of Guide**

**Mr. Kartheek GCR  
Asst. Professor  
Dept. of CSE, CMRIT**

---

### **Signature of HOD**

**Dr. Jhansi Rani P  
Professor & Head  
Dept. of CSE, CMRIT**

### External Viva

Name of the examiners

1.

2.

Signature with date

## **ABSTRACT**

OpenGL provides a set of commands to render a three dimensional scene. That means you provide them in an Open GL-useable form and Open GL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop Open GL-applications without licensing.

Open GL is a hardware- and system dependent interface. An Open GL-application will work on every platform, as long as there is an installed implementation, because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

This project includes the working of Prim's algorithm, and the displaying of the minimum spanning tree. We have implemented Prim's algorithm where the number of nodes and the weighted graph is read as input.

## **ACKNOWLEDGEMENT**

Behind every success there is a master hand. A master hand will create unperturbed concentration, dedication and encouragement in everything good and bad, without whose blessing this would have never come into existence.

Firstly, I thank God for showering the blessings on me. I am grateful to my institution CMRIT for providing me a congenial atmosphere to carry out the project successfully.

I would like to express my heartfelt gratitude to **Dr. Sanjay Jain**, Principal, CMRIT, Bangalore, for extending his support.

I am highly thankful to **Mrs. Jhansi Rani**, HOD of Computer Science and Engineering, CMRIT, Bangalore for her support and encouragement given to carry out the project.

I am very grateful to my guide, **Mr. Kartheek G C R**, Assistant Professor, Department of Computer Science, for his able guidance and valuable advice at early stage of my project which helped me in successful completion of my project.

Finally, I would like to thank my parents and friends who helped me with the content of this report, without which the project would not have become a reality.

**MADHUSOODHANAN KM (1CR15CS089)**  
**RISHABH D (1CR15CS129)**

## LIST OF FIGURES

1.	<b>Fig. 1.1:</b> Graphics System	1
2.	<b>Fig. 1.2:</b> Library organization of OpenGL	3
3.	<b>Fig. 1.3:</b> Graphics system as a black box	3
4.	<b>Fig. 4.1:</b> Flowchart	22
5.	<b>Fig. 4.2:</b> Initial Start Page	23
6.	<b>Fig. 4.3:</b> Input page	23
7.	<b>Fig. 4.4:</b> Node positioning	24
8.	<b>Fig. 4.5:</b> Displaying the weighted graph	24
9.	<b>Fig. 4.6:</b> Final display of the minimum spanning tree and min cost	25

# CONTENTS

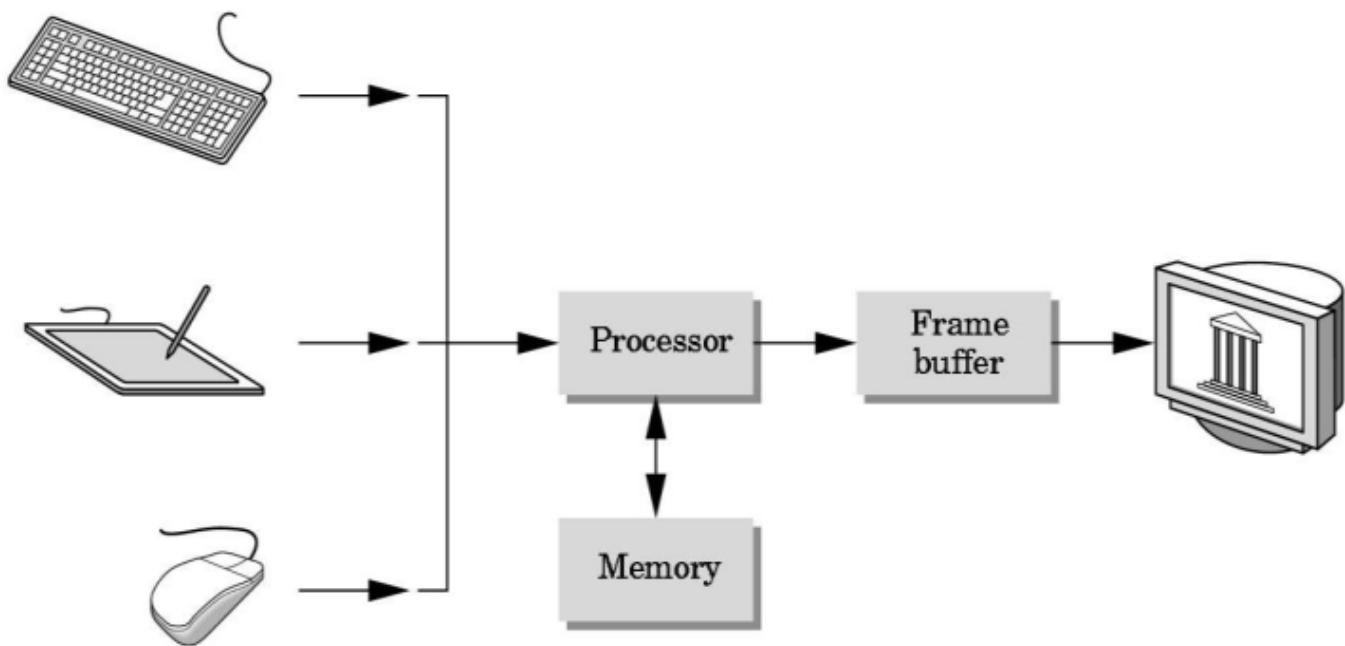
<b>Abstract</b>	i
<b>Acknowledgement</b>	ii
<b>List of figures and tables</b>	iii
<b>Contents</b>	iv
<b>1. Introduction</b>	1
1.1 Introduction to Computer graphics	
1.2 Areas of application of Computer graphics	
1.3 Introduction to OpenGL	
1.3.1 The OpenGL interface	
1.3.2 Graphics Functions	
<b>2. Requirements Specification</b>	5
2.1 Purpose of the requirements document	
2.1.1 Scope of the project	
2.1.2 Definition	
2.1.3 Acronyms & Abbreviations	
2.1.4 References	
2.2 Specific requirements	
2.2.1 User Requirements	
2.2.2 Software Requirements	
2.2.3 Hardware Requirements	
<b>3. Implementation</b>	7
3.1 OpenGL functions details	
3.2. Code in C Language	
<b>4. Description and Snapshots</b>	21
4.1 Description and Flowchart	
4.2 Screen Snapshots	
<b>5. Conclusion and Future Scope</b>	26
<b>References</b>	

# CHAPTER 1

## REQUIREMENTS SPECIFICATION

### 1.1 Introduction to Computer Graphics

- Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D or 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly.
- Computers have become a powerful medium for the rapid and economical production of pictures.
- Graphics provide a so natural means of communicating with the computer that they have become widespread.
- Interactive graphics is the most important means of producing pictures since the invention of photography and television.
- We can make pictures of not only the real world objects but also of abstract objects such as mathematical surfaces on 4D and of data that have no inherent geometry.
- A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.



**Fig 1.1:** Graphic System

## 1.2 Areas of Application of Computer Graphics

- User interfaces and Process control
- Cartography
- Office automation and Desktop publishing
- Plotting of graphs and charts
- Computer aided Drafting and designs
- Simulation and Animation

## 1.3 Introduction to OpenGL

**OpenGL** is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms.

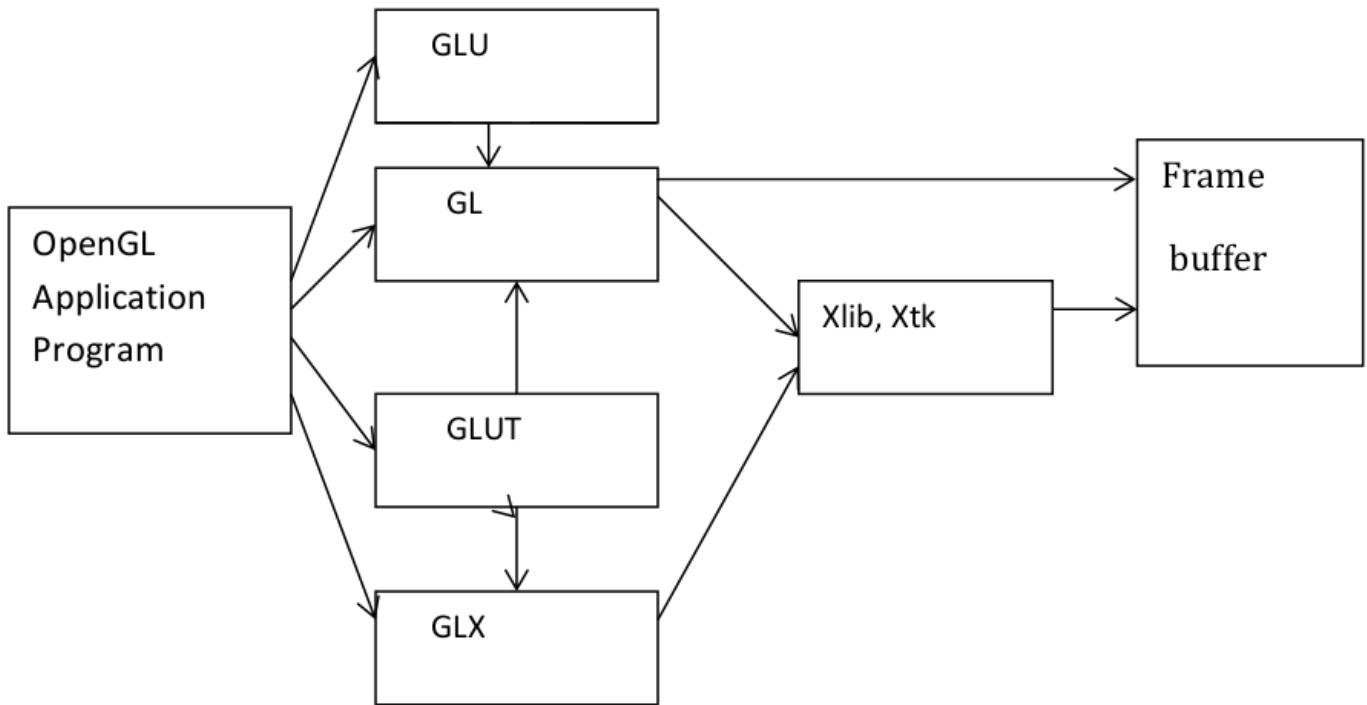
**OpenGL** fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

**OpenGL** available Everywhere: Supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and MacOS PC, no other graphics API operates on a wider range of hardware platforms and software environments.

**OpenGL** runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, Open Step, and BeOS; it also works with every major windowing system, including Win32, MacOS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies.

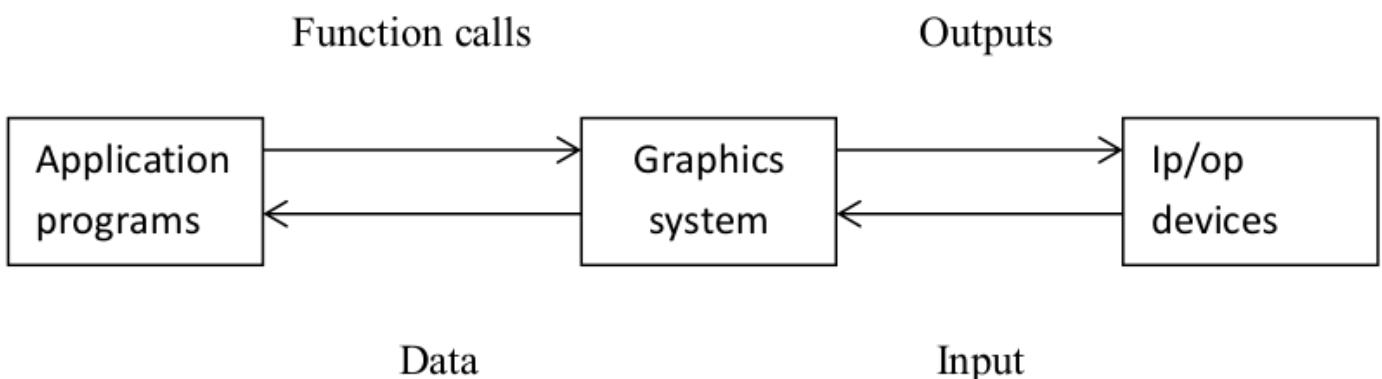
### 1.3.1 The OpenGL interface

Our application will be designed to access OpenGL directly through functions in three libraries namely: gl, glu, glut.



**Fig 1.2:** Library organization of OpenGL

### 1.3.2 Graphics Functions



**Fig 1.3:** Graphics system as a black box.

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 describe an API through the functions in its library. Some of the functions are:

- The primitive functions define the low-level objects or atomic entities that our system can 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 the inside of a polygon, to selecting a typeface for the titles of a graph.
- Transformation function allows carrying out transformations of objects, such as rotation, translation, and scaling.
- A set of input functions allow us to deal with the diverse forms of input that characterize modern graphics systems.
- The control functions enable us to communicate with the window systems, to initialize our programs, and to deal with any errors that take place during the execution of programs.

## CHAPTER 2

# REQUIREMENTS SPECIFICATION

### 2.1 Purpose of the requirements document

The software requirement specification is the official statement of what is required for development of particular project. It includes both user requirements and system requirements. This requirement document is utilized by variety of users starting from project manager who gives project to the engineer responsible for development of project.

It should give details of how to maintain, test, verify and what all the actions to be carried out through life cycle of project.

#### 2.1.1 Scope of the project

The scope is to use the basic primitives defined in openGL library creating complex objects. We make use of different concepts such as glColor(), gluOrtho2D(), timer function.

#### 2.1.2 Definition

This project is created to demonstrate OpenGL's concepts. It encompasses some of the skills learnt in our OpenGL classes such as glColor(), gluOrtho2D(), timer function.

#### 2.1.3 Acronyms & Abbreviations

OpenGL provides a powerful but primitive set of rendering command, and all higher-level design must be done in terms of these commands. OpenGL Utility Toolkit(GLUT): -windows-system-independent toolkit.

#### 2.1.4 References

OpenGL tutorials

Interactive Computer Graphics (Edward Angel)

## 2.2 Specific requirements

### 2.2.1 User Requirement:

- Easy to understand and should be simple.
- The built-in functions should be utilized to maximum extent.
- OpenGL library facilities should be used.

### 2.2.2 Software Requirements:

- Platform used: UBUNTU
- Technology used: OpenGL Libraries such has OpenGL Utility library, OpenGL Utility toolkit
- Language: C

### 2.2.3 Hardware Requirements:

- Processor-Intel or AMD(Advanced Micro Devices)
- RAM-512MB(minimum)
- Hard disk-1MB(minimum)
- Mouse
- Keyboard
- Monitor

## CHAPTER 3

# IMPLEMENTATION

### 3.1 OpenGL Function details

➤ **GlutInitDisplayMode** — sets the initial display mode.

- Declaration: void glutInitDisplayMode (unsigned int mode);

▪ Remarks: The initial display mode is used when creating top-level windows, sub windows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

➤ **glutInitWindowposition** --- set the initial window position.

- Declaration: void glutInitWindowPosition(int x, int y);

x: Window X location in pixels.

y: Window Y location in pixels.

➤ **glutInitWindowSize** --- set the initial window size.

- Declaration: void glutInitWindowSize(int width,int height);

width: Width in pixels

height: Height in pixels.

➤ **glutCreateWindow** --- set the title to graphics window.

- Declaration: Int glutCreateWindow(char \*title);

▪ Remarks: This function creates a window on the display. The string title can be used to label the window. The integer value returned can be used to set the current window when multiple windows are created.

➤ **glutDisplayFunc**

- Declaration: void glutDisplayFunc(void(\*func)void));

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

➤ **glClear:**

- Declaration: void glClear();

▪ Remarks: This function clears the particular buffer.

➤ **glClearColor:**

- Declaration: void glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha);

▪ Remarks: This function sets the color value that is used when clearing the color buffer.

### ➤ glEnd

- Declaration: void glEnd();
- Remarks: This function is used in conjunction with glBegin to delimit the vertices of an opengl primitive.

### ➤ glMatrixMode

- Declaration: void glMatrixMode(GLenum mode);
- Remarks: This function specifies which matrix will be affected by subsequent transformations mode can be GL\_MODELVIEW,GL\_PROJECTION or GL\_TEXTURE..

### ➤ gluOrtho2D

- Declaration: void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top);
- Remarks: It defines an orthographic viewing volume with all parameters measured from the center of the projection plane.

## 3.2. Code in C++ Language

```
#ifdef _MSC_VER // Windows
#include <windows.h>
#include <direct.h>
#include <process.h>
#else // Linux
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#endif /* _MSC_VER */

///#include<process.h>
#include<GL/glut.h>
#include<math.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
///#include<windows.h>
#define GL_PI 3.14
#define MAX 25
#define INF 999

int n,i=1,a[25],b[25],cost[25][25],cost1[25][25],tree[25][25],l[2];
int src,mincost=0;
int visited[25];
char s[20],*s1;
```

## Prim's Algorithm

```

int ch=1;
void *currentfont;

typedef struct {
    int u, v, w;
} Edge;

//const int NODES = 5 ; /* the number of nodes */
int EDGES=0;          /* the number of edges */
Edge edges[32];       /* large enough for n <= 2^NODES=32 */
// edges is an array of structures

```

## //FUNCTION TO SELECT BITMAP FONT

```

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

```

## //FUNCTION TO DRAW sentence string at (x,y)

```

void drawstring(GLfloat x,GLfloat y,char *c)
{
    glRasterPos2f(x,y);

    //This loop ensures that the whole sentence is drawn out
    while(*c!='\0')
    {
        glutBitmapCharacter(currentfont,*c);
        *c++;
    }
}

```

## //FUNCTION TO DELAY- This simply gives the animations a fluid like feel

```

void delay()
{
    for(int i=0;i<25000;i++)
        for(int j=0;j<25000;j++);
}

```

## //DISPLAY FUNCTION FOR TITLE PAGE

## Prim's Algorithm

```

void title()
{
    glLineWidth(5);                                //This line loop is just to make a
    border within the window for asthetics
    glColor3f(1,1,0);                            //Yellow border
    glBegin(GL_LINE_LOOP);
        glVertex2f(10,10);
        glVertex2f(10,490);
        glVertex2f(490,490);
        glVertex2f(490,10);
    glEnd();

    setFont(GLUT_BITMAP_HELVETICA_18);

    glColor3f(1.0,1.0,1.0);
    drawstring(100,440,"Topic: Prim's Algorithm");
    glColor3f(1.0,1.0,1.0);
    drawstring(100,400,"Submitted by");
    glColor3f(1.0,0.0,0.0);
    drawstring(100,360,"Rishabh D\at 1CR15CS129");
    glColor3f(1.0,0.0,0.0);
    drawstring(100,320,"Madhusoodhanan KM\at 1CR15CS089");
    glColor3f(1.0,0.0,0.0);
    drawstring(100,280,"6th CSE B");
    glColor3f(1.0,1.0,1.0);
    drawstring(100,100,"Right click in My Window for options");

    glFlush();
}

```

```

//DISPLAY FUNCTION FOR INITIALIZING (DRAWING) THE INPUT AND OUTPUT AREAS
void initialDraw()
{
    glClear(GL_COLOR_BUFFER_BIT);

    setFont(GLUT_BITMAP_HELVETICA_18);

    glColor3f(0.0,0.0,0.0);
    drawstring(20,230,"Input Area");
    drawstring(20,470,"Output Area");

    glColor3f(0.0,0.0,0.0);                      // Color Black
    glLineWidth(2.5);
    glBegin(GL_LINE_LOOP);                      // These Line loops are to border the input and output
    boxes in the My Window
    glVertex2f(10,10);

```

## Prim's Algorithm

```

glVertex2f(490,10);
    glVertex2f(490,250);
    glVertex2f(10,250);
glEnd();

    glBegin(GL_LINE_LOOP);
    glVertex2f(10,255);
    glVertex2f(490,255);
        glVertex2f(490,490);
    glVertex2f(10,490);

glEnd();

glFlush();
}

//BLANK DISPLAY FUNCTION
void display (void)
{
    glFlush();
}

//DRAW A BITMAP NUMBER i at (x,y)
void raster(int x,int y,int i)
{
    char z=i+'0';
    glRasterPos2f(x-5,y-5);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,z);
}

//DRAW THE NODES (SQUARES)
void drawSquare(int x, int y)
{
    if(i<=n)                                // This ensures that one cannot place
more than assigned nodes onto the screen
    {
        y = 500-y;                          //Convert from screen coordinates
        glPointSize(35);                    // This sets a single point size to be that big

        if(i==src)

```

## Prim's Algorithm

```

glColor3f(0.7f, 0.4f, 0.0f);
else
    glColor3f(0.5f, 0.5f, 0.8f);

glBegin(GL_POINTS);
    glVertex2f(x , y);
glEnd();

a[i]=x;
b[i]=y;

glColor3f(0.0f, 1.0f, 0.0f);
int i1=sprintf(s,"%d",ch);           // Prints the name of the node
s1=s;
drawstring(x-5,y-5,s1);

glFlush();
}

ch++;                                // Increments the letter to name the node
i++;                                    // Increment as the num of nodes
formed increases
}

//READ DATA: |V|,COST MATRIX
void read()
{
    printf("Enter the number of vertices:\n");
    scanf("%d",&n);
    printf("Enter the cost adjacency matrix:\n");
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
    {
        scanf("%d",&cost[i][j]);
        if(cost[i][j]==0)           // Means if node is unreachable via that path
            cost[i][j]=999;

        else
        {
            edges[EDGES].u=i;
            edges[EDGES].v=j;
        }
    }
}

```

## Prim's Algorithm

```

edges[EDGES].w=cost[i][j];
    EDGES++;
}
}

for(int i=1;i<=n;i++)
{
    for(int j=1;j<=n;j++)
    {
        cost1[i][j]=cost[i][j];
        if(cost1[i][j]==0) // Means if node is unreachable via that path
            cost1[i][j]=999;
    }
}
printf("\nGO TO MY WINDOW, PLACE THE NODES IN INPUT AREA AND THEN CLICK RIGHT BUTTON
FOR NEXT OPTION\n");
initialDraw(); //Draw the initial screen
}

```

//DRAW THE EDGES THAT CONNECTING THE NODES

```

void drawlineCN()
{
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(x1-6,y1+10);
    glVertex2i(x2-6,y2+10);
glEnd();

```

```
int i2=sprintf(s,"%d",cost[j][k]);
```

Prints the weight onto the Edge

```

s1=s;
drawstring((x1+x2-16)/2,(y1+y2+22)/2,s1);

```

## Prim's Algorithm

```

glFlush();
}

if(cost[j][k]!=cost[k][j] && cost[j][k]!=999 && j>k)           // This checks if the same path
weight A-->B and B-->A is there... If so, then dont draw
{
    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();
    int i3=sprintf(s,"%d",cost[j][k]);                                // Prints the
weight onto the Edge
    s1=s;
    drawstring((x1+x2+20)/2,(y1+y2+36)/2,s1);
    glFlush();
}
}
}

```

```

void shortestpath(int src)
{
int p,q,x,y,x1,x2,y1,y2,j;
int k=1,u=0,v=0,min;
int l=0;

while(k<n)
{
    for(i=1,min=999;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            if(cost[i][j]<min)
            {
                if(visited[i]==1)
                {
                    u=i;
                    v=j;
                    min=cost[i][j];
                }
            }
        }
    }
}

```

## Prim's Algorithm

```

        }
    }

//Setting the end vertices of an edge to tree array
tree[l][1]=u;
tree[l++][2]=v;
printf("%d-->%d=%d\n",u,v,min);

k++;
mincost=mincost+min;
visited[v]=1;
cost[u][v]=cost[v][u]=999;

}

printf("The minimum cost is: %d\n",mincost);

//DRAW THE SPANNING TREE
int cch=1; // In the output section while naming
the spanning tree
for(int r=1;r<=n;r++)
{
    x=a[r];
    y=b[r];

    glPointSize(35);
    if(r==src)
        glColor3f(0.7f, 0.4f, 0.0f);
    else
        glColor3f(0.5f, 0.5f, 0.8f);

    glBegin(GL_POINTS);
    glVertex2f(x+5,y+254); // DO NOT change this value... It was
set by MD such that the letter appears mid of node
    glEnd();

    glColor3f(0.0,1.0,0.0);
    int i4=sprintf(s,"%d",cch); // Prints the name of the node
    s1=s;
    drawstring(x,y+250,s1);

    glFlush();
}

```

## Prim's Algorithm

```

cch++;
// So
that next node has next letter
}

for(int x=0;x<l;x++)
{
    p=tree[x][1];
    q=tree[x][2];
    if(p==0||q==0) continue;

    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();
        int i5=sprintf(s,"%d",cost1[p][q]);
        // Prints the weight onto the Edge

        s1=s;

        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();
        int i6=sprintf(s,"%d",cost1[p][q]);
        // Prints the weight onto the Edge

        s1=s;

        drawstring((x1+x2)/2,(y1+y2+500)/2,s1);

    }
}

```

## Prim's Algorithm

```

}

glFlush();

}

void mouse(int button, int state , int x , int y)           // This takes in inputs when the mouse is
interacted with... The x & y are the coordinates of wherw the button is clicked
{
    if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        drawSquare(x,y);

}

void top_menu(int option)                                // This is the right mouse button click options
{
    int x,y;
    switch(option)
    {
        case 1:
            read();                                     // Reads the nodes and cost matrix
            glutPostRedisplay();
            break;

        case 2:
            drawlineCN();
            glutPostRedisplay();
            break;

        case 3:
            {
                for(i=1;i<=n;i++)
                    visited[i]=0;

                printf("Enter the source vertex:");
                scanf("%d",&src);
                visited[src]=1;

                glClear(GL_COLOR_BUFFER_BIT);           // Clears
screen
                initialDraw();
                int cch=1;
                for(int r=1;r<=n;r++)
                {

```

## Prim's Algorithm

```

x=a[r];
y=b[r];

glPointSize(35);

if(r==src)
    glColor3f(0.7f, 0.4f, 0.0f);
else
    glColor3f(0.5f, 0.5f, 0.8f);

glBegin(GL_POINTS);
    glVertex2f(x,y);
glEnd();

```

// THE  
REASON WE RE draw even the inputs is coz we do clear color each iteration... For the Prims part, we wont need this section at all

```

glColor3f(0.0,1.0,0.0);
int i6=sprintf(s,"%d",cch);
s1=s;
drawstring(x-5,y-5,s1);
setFont(GLUT_BITMAP_HELVETICA_18);
glColor3f(0.0,0.0,0.0);
drawstring(130,470,"For source");
glFlush();

cch++;
}

drawlineCN();
int i6=sprintf(s,"%d",src);
// This line is the OP line for 1,2,3,....
s1=s;
setFont(GLUT_BITMAP_HELVETICA_18);
glColor3f(0.0,0.0,0.0);
drawstring(225,470,s1);
glutPostRedisplay();
shortestpath(src);

}
break;

case 4:
exit(0);
}

```

## Prim's Algorithm

---

}

```

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();
}

void myInit1()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glColor3f(0.0f,0.0f,0.0f);
    glPointSize(5.0);
    gluOrtho2D(0.0,500.0,0.0,500.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    setFont(GLUT_BITMAP_HELVETICA_18);
}

void display1(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    title();
}

int main (int argc,char** argv)
{
    glutInit(&argc,argv);

    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowPosition(0,0);                                // x,y where 0,0 is top left corner and y
increases downward
    glutInitWindowSize(500,500);                                // Hz x Vert
    glutCreateWindow("CG Mini Project");
    glutDisplayFunc(display1);                                 // displays the contents on Title Sheet
    myInit1();
}

```

## Prim's Algorithm

```

glutInitDisplayMode( GLUT_SINGLE|GLUT_RGB );
glutInitWindowPosition(550,0);                                     // 0,0 is the origin
which is the top left most corner of the monitor screen- From there the size of the window is drawn
glutInitWindowSize(500,500);                                         // Breadth(Horizontal)
x Length(Vertical) of the window- Only works for 500x500
glutCreateWindow("My Window");
glutDisplayFunc(display);
glutMouseFunc(mouse);

glutCreateMenu(top_menu);                                            // This creates the
options menu to open once RMB is clicked
glutAddMenuEntry("Read Cost Matrix",1);
glutAddMenuEntry("Display Weighted Graph",2);
glutAddMenuEntry("Display Minimum Spanning tree",3);
glutAddMenuEntry("Exit",4);
glutAttachMenu(GLUT_RIGHT_BUTTON);

printf("\nGO TO MY WINDOW AND CLICK RIGHT BUTTON FOR NEXT OPTION\n"); // 
Displayed in terminal
init();

glutMainLoop();
}

```

## CHAPTER 4

# DESCRIPTION AND SNAPSHOTS

### 4.1 DESCRIPTION

In computer science, **Prim's algorithm** is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

The algorithm may informally be described as performing the following steps:

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.
2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
3. Repeat step 2 (until all vertices are in the tree).

In more detail, it may be implemented following the pseudocode below.

1. Associate with each vertex  $v$  of the graph a number  $C[v]$  (the cheapest cost of a connection to  $v$ ) and an edge  $E[v]$  (the edge providing that cheapest connection). To initialize these values, set all values of  $C[v]$  to  $+\infty$  (or to any number larger than the maximum edge weight) and set each  $E[v]$  to a special flag value indicating that there is no edge connecting  $v$  to earlier vertices.
2. Initialize an empty forest  $F$  and a set  $Q$  of vertices that have not yet been included in  $F$  (initially, all vertices).
3. Repeat the following steps until  $Q$  is empty:
  - a. Find and remove a vertex  $v$  from  $Q$  having the minimum possible value of  $C[v]$
  - b. Add  $v$  to  $F$  and, if  $E[v]$  is not the special flag value, also add  $E[v]$  to  $F$
  - c. Loop over the edges  $vw$  connecting  $v$  to other vertices  $w$ . For each such edge, if  $w$  still belongs to  $Q$  and  $vw$  has smaller weight than  $C[w]$ , perform the following steps:
    - i. Set  $C[w]$  to the cost of edge  $vw$
    - ii. Set  $E[w]$  to point to edge  $vw$ .
4. Return  $F$

As described above, the starting vertex for the algorithm will be chosen arbitrarily, because the first iteration of the main loop of the algorithm will have a set of vertices in  $Q$  that all have equal weights, and the algorithm will automatically start a new tree in  $F$  when it completes a spanning tree of each connected component of the input graph. The algorithm may be modified to start with any particular vertex  $s$  by setting  $C[s]$  to be a number smaller than the other values of  $C$  (for instance, zero), and it may be modified to only find a single spanning tree rather than an entire spanning forest (matching more closely the informal description) by stopping whenever it encounters another vertex flagged as having no associated edge.

Different variations of the algorithm differ from each other in how the set  $Q$  is implemented: as a simple linked list or array of vertices, or as a more complicated priority queue data structure. This choice leads to differences in the time complexity of the algorithm. In general, a priority queue will be quicker at finding the vertex  $v$  with minimum cost, but will entail more expensive updates when the value of  $C[w]$  changes.

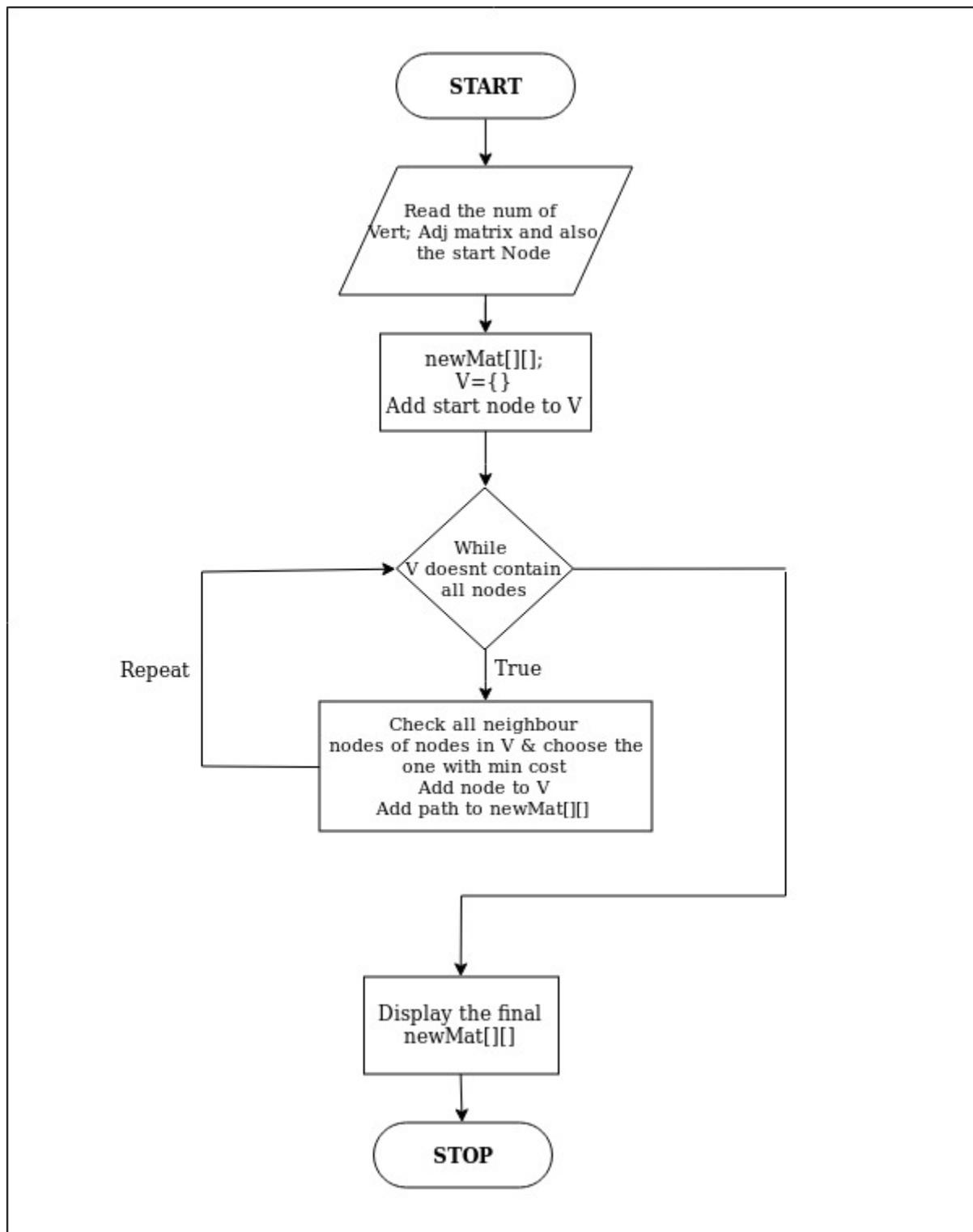
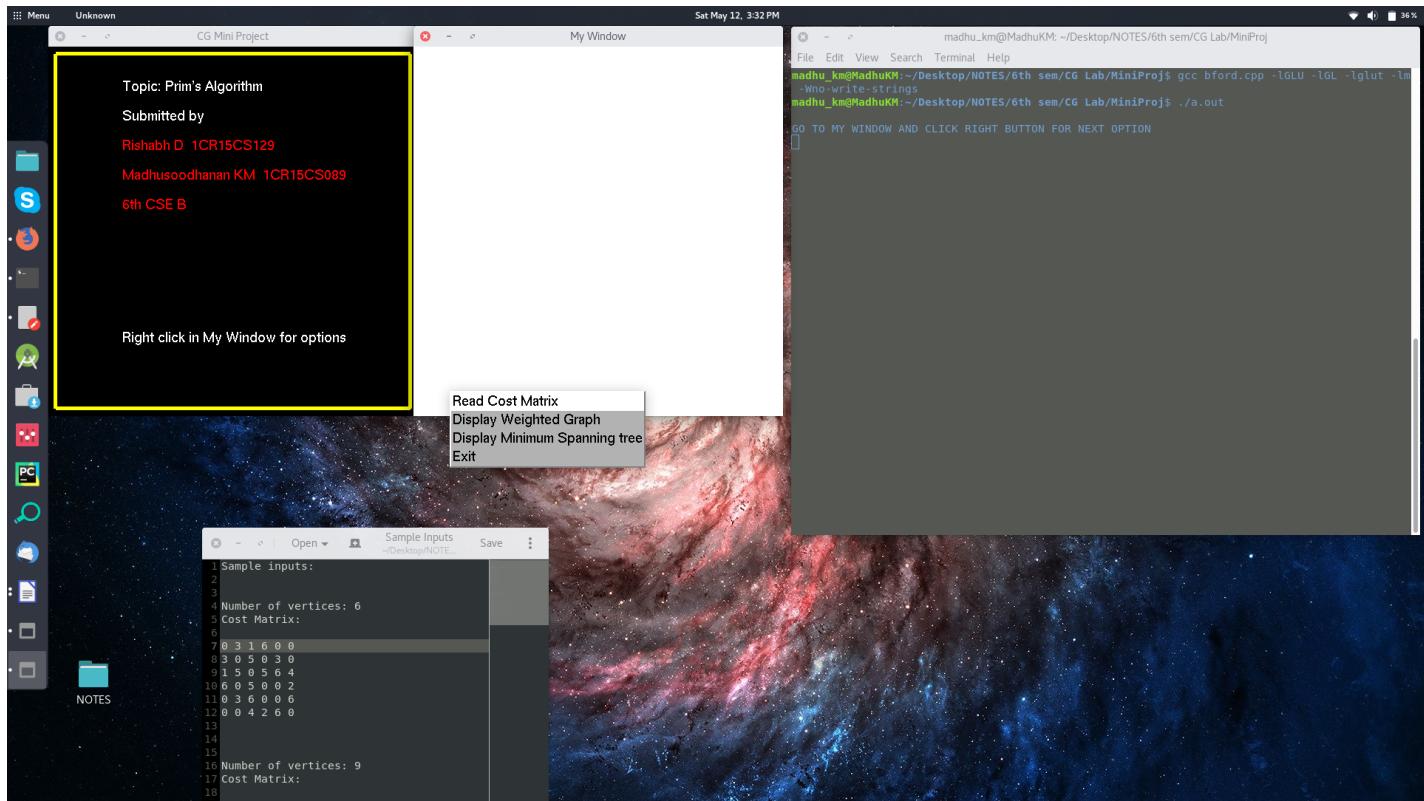


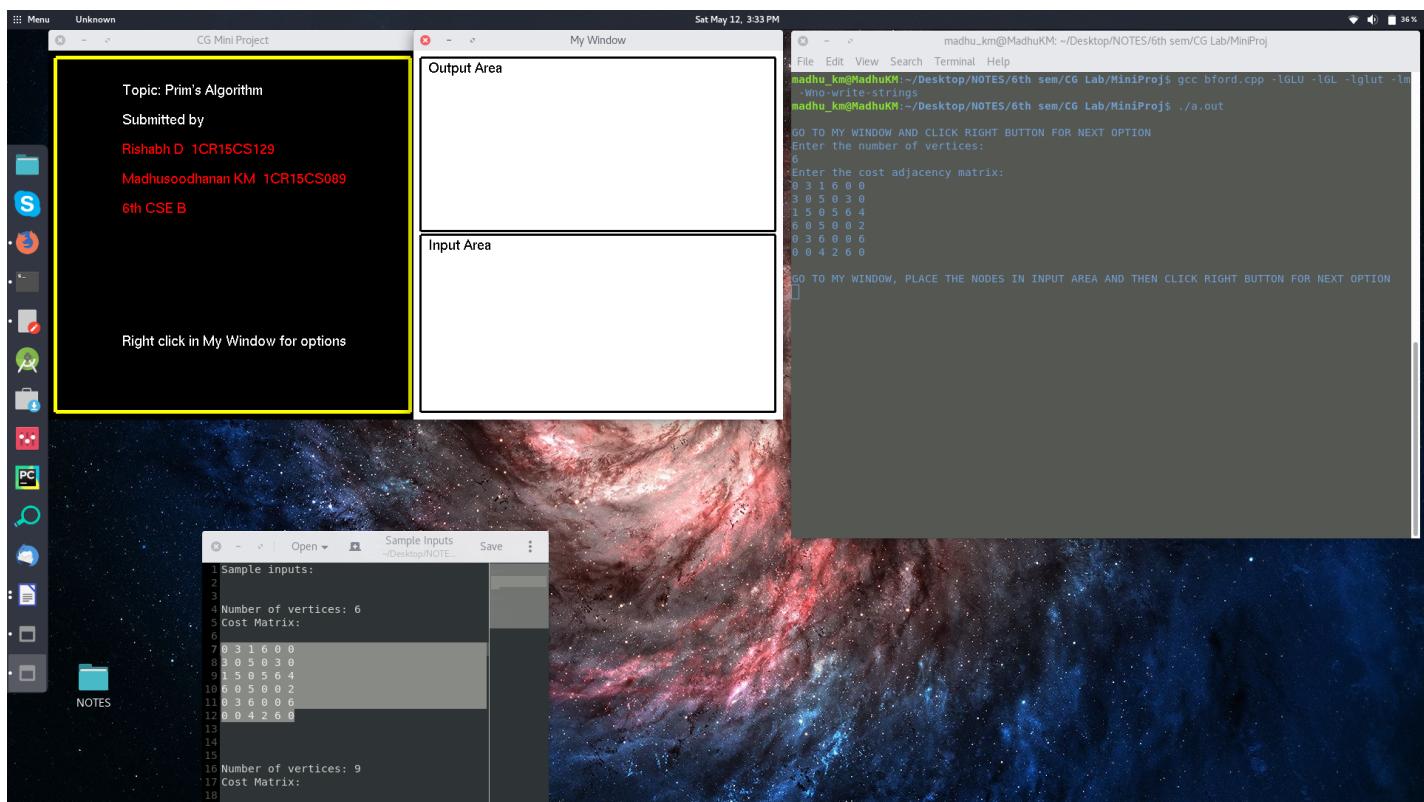
Fig 4.1: Flowchart

## Prim's Algorithm

### 4.2 SCREEN SNAPSHOTS



**Fig 4.2: Start page**



**Fig 4.3: Input page**

## Prim's Algorithm

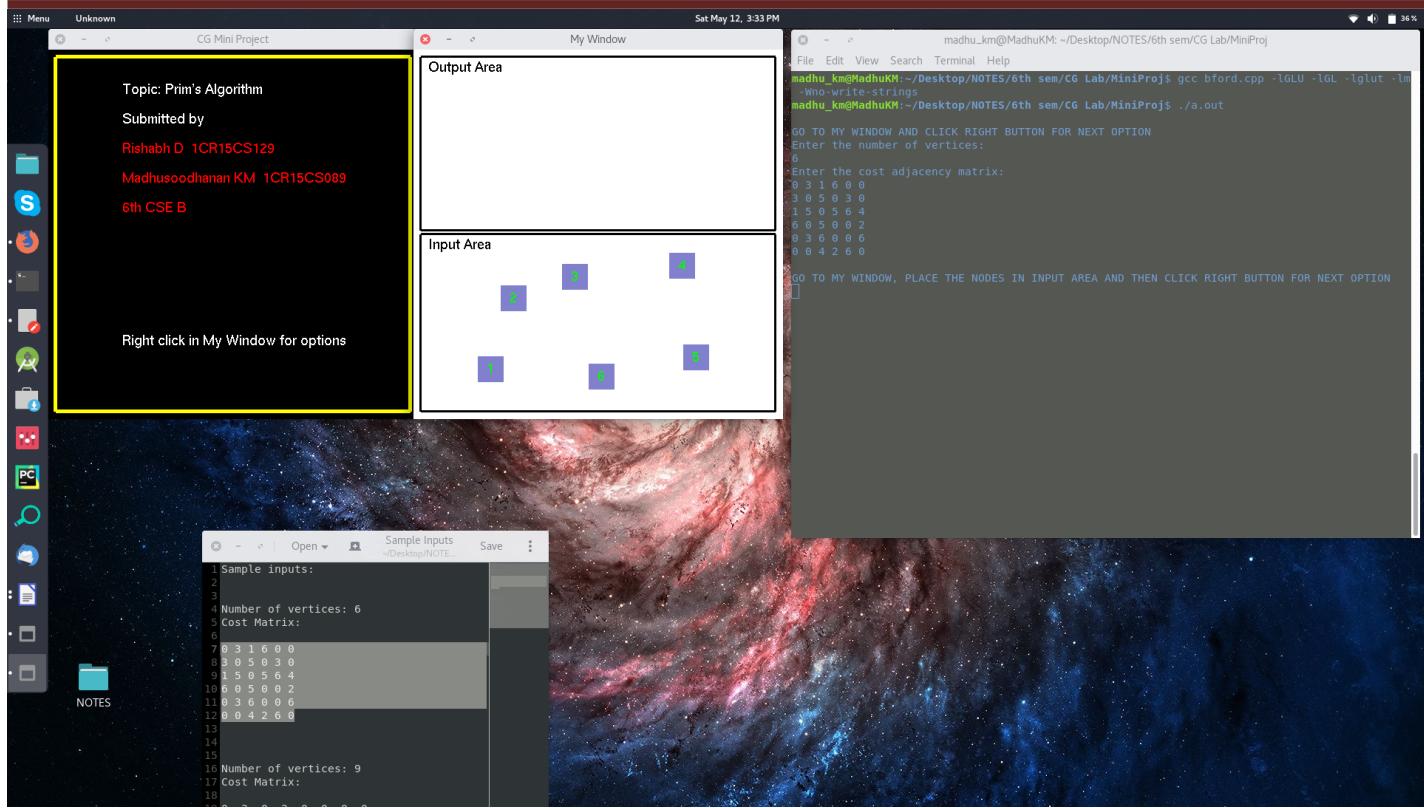


Fig 4.4: Node positioning

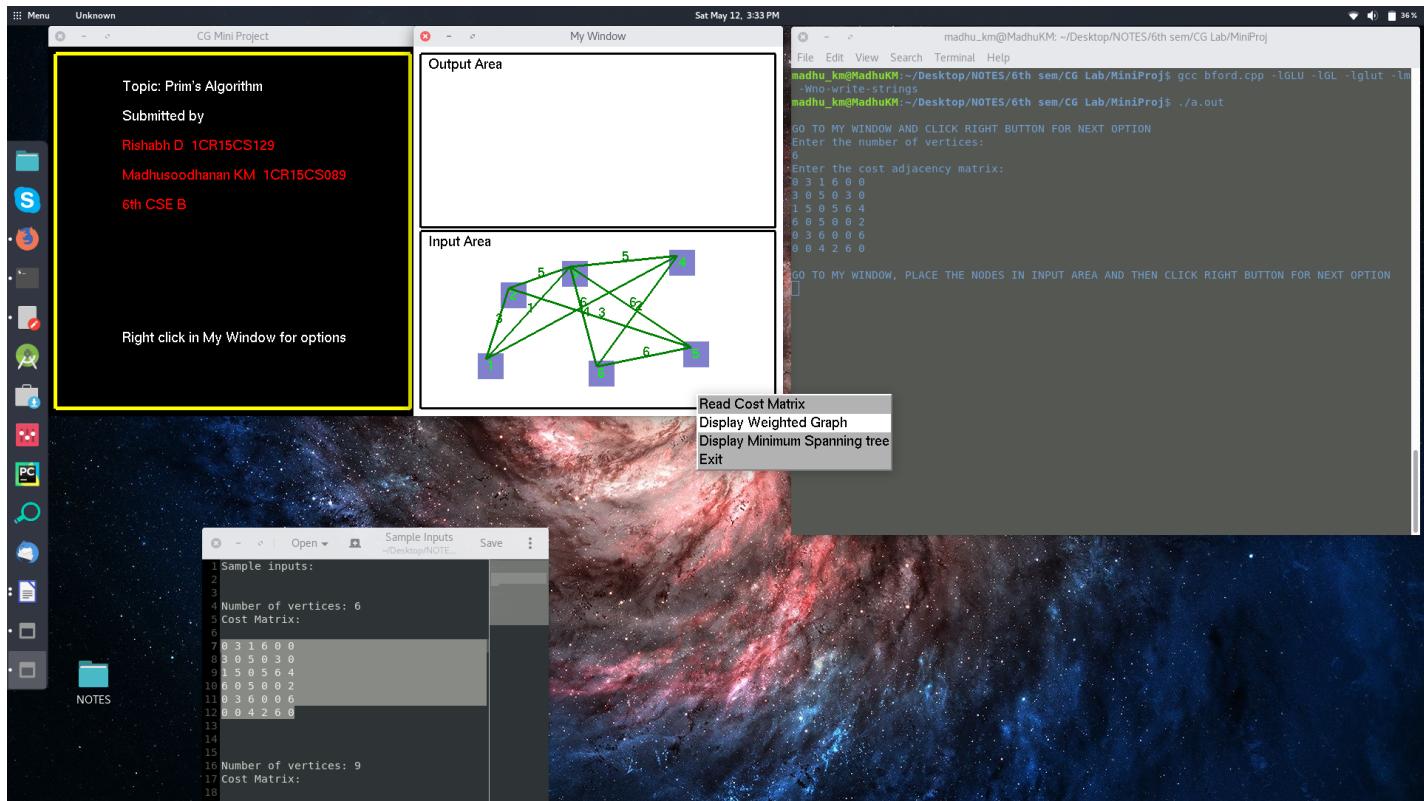


Fig 4.5: Displaying the weighted graph

## Prim's Algorithm

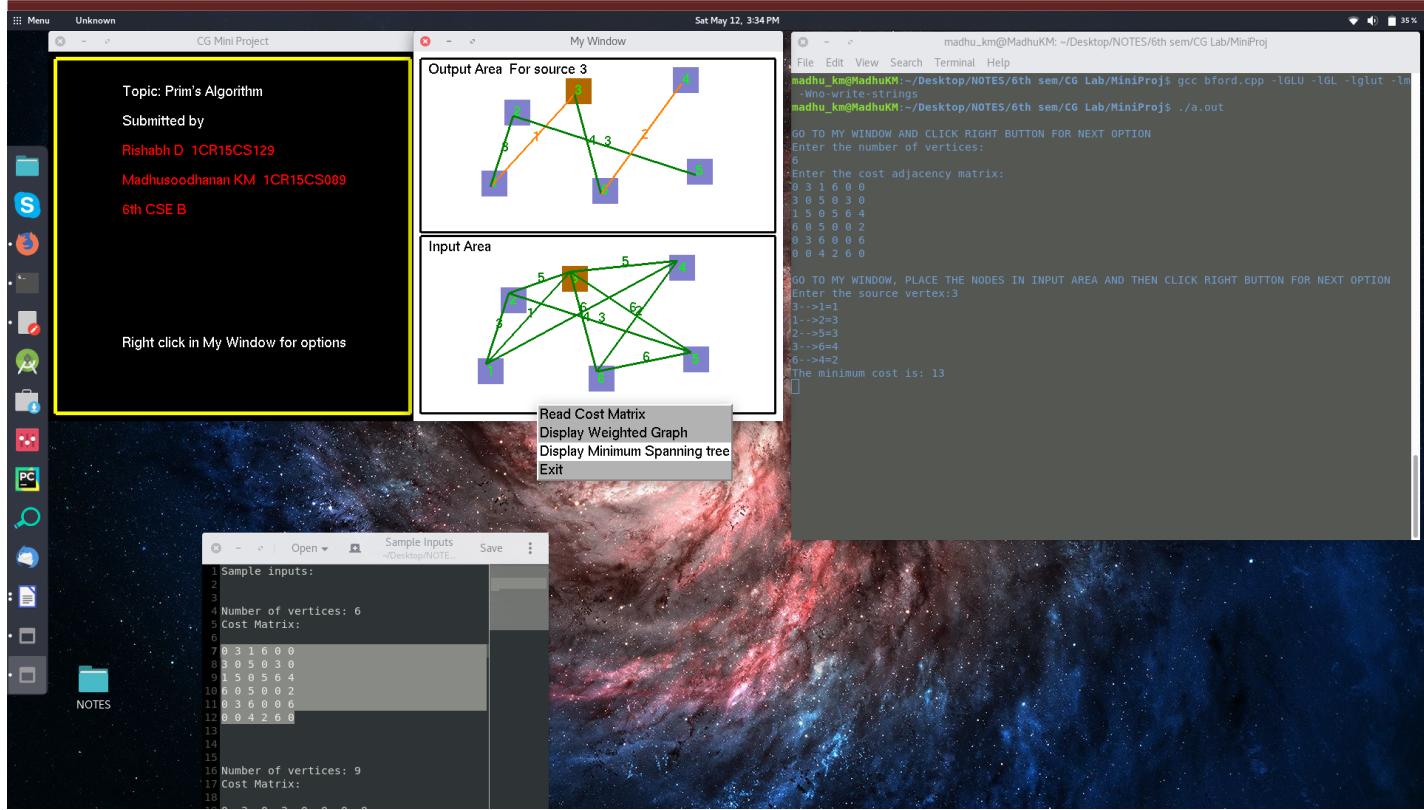


Fig 4.6: Final display of minumum spanning tree and min cost

## CHAPTER 5

### CONCLUSION AND FUTURE SCOPE

By implementing this project I got to know how to use some of the built in functions effectively and how to interact efficiently and easily. I got a good exposure of how games, animation and simulations are developed, by working on this project.

The OpenGL Utility Toolkit (GLUT) is a programming interface with ANSI C bindings for writing window system independent OpenGL programs.

One of the major accomplishments in the specification of OpenGL was the isolation of window system dependencies from OpenGL's rendering model. The result is that OpenGL is window system independent. Window system operations such as the creation of a rendering window and the handling of window system events are left to the native window system to define. Necessary interactions between OpenGL and the window system such as creating and binding an OpenGL context to a window are described separately from the OpenGL specification in a window system dependent specification.

The GLUT application-programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT API (like the OpenGL API) is stateful. Most initial GLUT state is defined and the initial state is reasonable for simple programs. The GLUT routines also take relatively few parameters. No pointers are returned. The only pointers passed into GLUT are pointers to character strings (all strings passed to GLUT are copied, not referenced) and opaque font handles.

## REFERENCES

- [1] Edward Angel, “Interactive Computer Graphics”, 5th edition, Pearson Education, 2005
- [2] “Computer Graphics”, Addison-Wesley 1997 James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes.
- [3] F.S.Hill and Stephen M.Kelly, “Computer Graphics using OpenGL “, 3rd edition
- [4] <http://www.opengl.org>
- [5] [http:// www.openglprojects.in](http://www.openglprojects.in)
- [6] <http://www.openglprogramming.com/>
- [7] <http://www.google.com/>
- [8] <http://www.stackoverflow.com/>