Computer Graphics & Multimedia

Lab Programs – 2020-21

Please Note:

- This document contains all CG lab programs along with the output.
- The programs are taken from the official CG Lab Manual, upon which small modifications are done to improve the code and reduce the redundancy.
- The lab programs in this document will provide correct outputs as shown and these are only for reference and not to be considered for any judgement.
- The document is an unofficial copy. Creator of this document is not responsible if you don't get the proper output for the programs during lab examination. So, please execute the programs and verify the output.
- If there are any mistakes please report or for any queries contact via Whatsapp (9483085114). The rectified new copy will be uploaded to google drive and you can find this updated copy under "VII Semester Study Materials".

Prepared by:
Shawn Linton Miranda
4NM17CS164

INDEX

* Items in table of contents are clickable so you can navigate to respective page by clicking the items.

PLEASE NOTE:

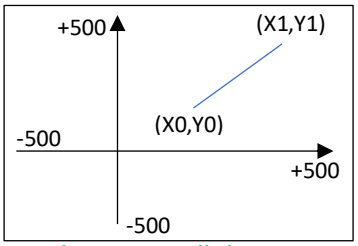
1. For easier understanding of the program, each line of code is given with the comment line. The same comments id not repeated again and again instead each comment is written in front of first occurrence of any line of code in this manual.
2. While executing in lab: CodeBlocks -> Console Application -> Execute as normal C program
3. While executing at home install the GLUT and go with GLUT Project instead of Console Application
4. For windows you need to include <windows.h> and for ubuntu please don't add it

1. *Mid-Point Line Algorithm*

Program to implement Mid-Point Line Algorithm. The line coordinates should be specified by the user.

Version-1: (Capable of drawing line only in 1st quadrant : X1<X2, Y1<Y2, slope>=0)

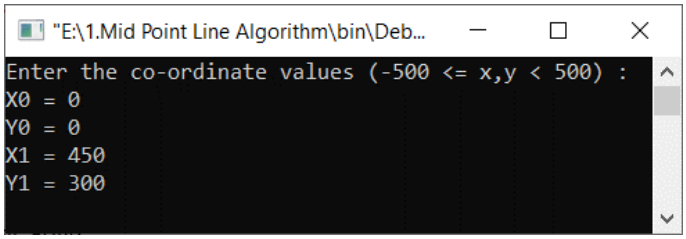
Program:

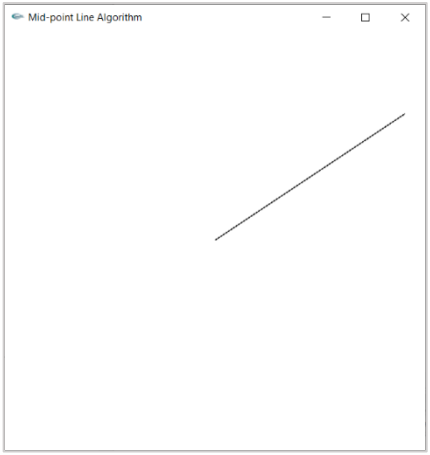



```
#include <windows.h>        //Don't include this for Ubuntu OS
#include <GL/glut.h>              //OpenGL library
#include <stdio.h>          //Standard Input Output
#include <math.h>           //For mathematical operations
int X0, Y0, X1, Y1;         //Variables to store end points of the line
          //Don't use y0 and y1 (Lower case) variables as global variables in ubuntu. It will throw error.
void init() {
   glutInitWindowSize(500,500);                    //Sets window size 500x500 pixels
   glutCreateWindow("Mid-point Line Algorithm");   //Creates window with the title
   glClearColor(1,1,1,1);                          //Background color for window – White (r,g,b,a)
   glClear(GL_COLOR_BUFFER_BIT);                   //Clear the color buffer bit i.e enable the color buffers
   gluOrtho2D(-500,500,-500,500);          //Set 2D cartesian plane within the window (left,right,bottom,top)
   glColor3f(0,0,0);                                     //Color of the point to be drawn – Black  (r,g,b)
   glPointSize(1);                                 //Thickness of the point to be drawn  (pixels)
}

void writePixel(int x, int y) {         //Draw a single point at (x,y) position
  glBegin(GL_POINTS);                   //Begin the list of point co-ordinates. Don't write GL_POINT its GL_POINTS only
  glVertex2f(x,y);                      //Specify the set of vertices   (x,y)
  glEnd();                              //End the list of point co-ordinates
}

void drawLine() {               //Function to draw the line as per the algorithm
   int x,y;
   double dx, dy, slope, d;
   dx = X1-X0;
   dy = Y1-Y0;
   slope = dy/dx;    //Compute the slope and assign x,y with the starting co-ordinates of line to be drawn
   x = X0;
   y = Y0;
   if(fabs(slope)<=1) {        //If line is near to the X-axis. Here we have taken float-abs i.e |slope|
      d = dy-(dx/2);           //Compute decision variable and proceed according to algorithm
      writePixel(x,y);
      while(x<X1) {
         x++;
         if(d<0)               //Update the decision variable
            d = d+dy;
         else {
            d = d+dy-dx;
            y++;
         }
         writePixel(x,y);      //Draw the point
      }
   }
   else {              //Line is near to Y-axis
      d = dx-(dy/2);
      writePixel(x,y);
      while(y<Y1) {
         y++;
```

```
            if(d<0)
               d = d+dx;
            else {
               d = d+dx-dy;
               x++;
            }
            writePixel(x,y);
         }
      }
      glFlush();          //Flush the previously drawn graphic elements to the screen so that they are visible
   }

   int main(int argc, char *argv[]) {                                    //Main function
      printf("Enter the co-ordinate values (-500 <= x,y < 500) :\n");    //Input end points of line
      printf("X0 = "); scanf("%d",&X0);
      printf("Y0 = "); scanf("%d",&Y0);
      printf("X1 = "); scanf("%d",&X1);
      printf("Y1 = "); scanf("%d",&Y1);

      glutInit(&argc,argv);                  //Initiate the glut
      init();                                //Make other initial setups

      glutDisplayFunc(drawLine);             //Callback that draws the elements to the graphic window
      glutMainLoop();                        //Infinite loop which updates the graphics
      return 0;
   }
```

Output:





<u>Version-2:</u> (Capable of drawing line in all quadrants of cartesian plane including line with negative slope)

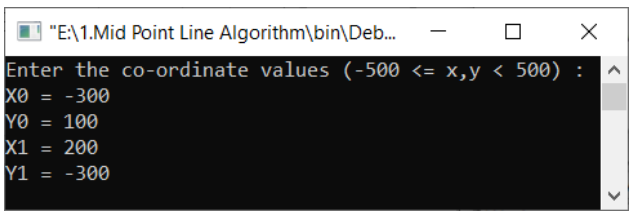
Program:

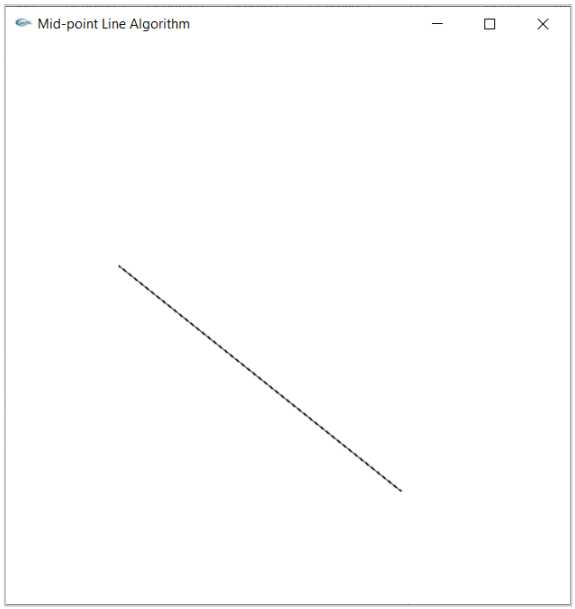
```
   #include <windows.h>       //Don't include this for Ubuntu OS
   #include <GL/glut.h>
   #include <stdio.h>
   #include <math.h>
   int X0, Y0, X1, Y1;

   void init() {
      glutInitWindowSize(500,500);
      glutCreateWindow("Mid-point Line Algorithm");
      glClearColor(1,1,1,1);
      glClear(GL_COLOR_BUFFER_BIT);
```

```
    gluOrtho2D(-500,500,-500,500);
    glColor3f(0,0,0);
    glPointSize(1);
}

void writePixel(int x, int y) {
    glBegin(GL_POINTS);
    glVertex2f(x,y);
    glEnd();
}

void swapXY() {                  //Swap end points of two lines such that we draw always from bottom to top
    int temp;
    temp = X0;    X0 = X1;    X1 = temp;
    temp = Y0;    Y0 = Y1;     Y1 = temp;
}

void drawLine() {
    int x,y;
    double dx, dy, slope, d;
    if(Y0>Y1 || (Y0==Y1 && X0>X1))  swapXY();                    //If P1 is above P2 just swap the end points
    dx = X1-X0;    dy = Y1-Y0;
    slope = dy/dx;
    x = X0;
    y = Y0;
    if(slope>=0) {                  //Lines with positive slope and parallel to X-axis
        if(fabs(slope)<=1) {
            d = dy-(dx/2);
            writePixel(x,y);
            while(x<X1) {
                x++;
                if(d<0)
                    d = d+dy;
                else {
                    d = d+dy-dx;
                    y++;
                }
                writePixel(x,y);
            }
        }
        else {
            d = dx-(dy/2);
            writePixel(x,y);
            while(y<Y1) {
                y++;
                if(d<0)
                    d = d+dx;
                else {
                    d = d+dx-dy;
                    x++;
                }
                writePixel(x,y);
            }
        }
    }
```

```
    else {                      //Lines with negative slope. Version-1 of program cannot draw these type of lines
      if(fabs(slope)<=1) {
        d = -dy-(dx/2);         //Formulas and conditions for negative slope lines are different from the previous
        writePixel(x,y);        //Please check the conditions properly
        while(x>X1) {
          x--;
          if(d>0)
            d = d-dy;
          else {
            d = d-dy-dx;
            y++;
          }
          writePixel(x,y);
        }
      }
      else {
        d = dx-(-dy/2);
        writePixel(x,y);
        while(y<Y1) {
          y++;
          if(d>0)
            d = d+dx;
          else {
            d = d+dx+dy;
            x--;
          }
          writePixel(x,y);
        }
      }
    }
    glFlush();
}

int main(int argc, char *argv[]) {
    printf("Enter the co-ordinate values (-500 <= x,y < 500) :\n");
    printf("X0 = "); scanf("%d",&X0);
    printf("Y0 = "); scanf("%d",&Y0);
    printf("X1 = "); scanf("%d",&X1);
    printf("Y1 = "); scanf("%d",&Y1);

    glutInit(&argc,argv);
    init();

    glutDisplayFunc(drawLine);
    glutMainLoop();
    return 0;
}
```

Output:



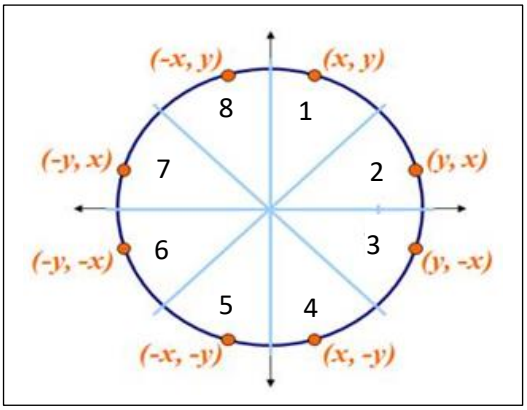


2. Mid-Point Circle Algorithm

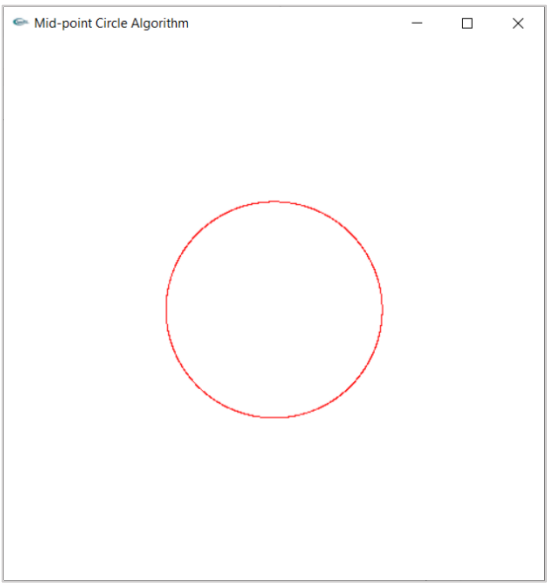
Program to implement Mid-Point Circle Algorithm. The radius should be specified by the user.

Program:

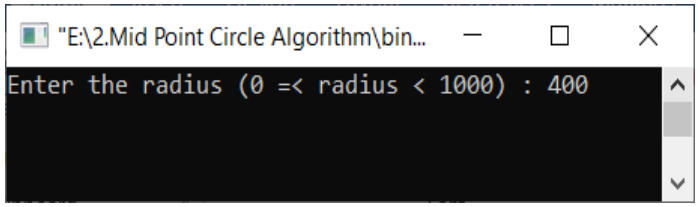
```
#include <windows.h>        //Don't include this for Ubuntu OS
#include <GL/glut.h>
#include <stdio.h>
int radius;

void init() {
   glutInitWindowSize(500,500);
   glutCreateWindow("Mid-point Circle Algorithm");
   glClearColor(1,1,1,1);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(-1000,1000,-1000,1000);
   glColor3f(1,0,0);
   glPointSize(1);
}

void writePixel(int x,int y) {
   glBegin(GL_POINTS);
   glVertex2f(x,y);
   glEnd();
}

void drawCircle() {             //Function to draw the circle as per the algorithm
   int x,y;
   double d;
   x=0;
   y=radius;
   d = 5.0/4-radius;            //Initialize decision variable for circle drawing. 5 is written as 5.0 instead of 5 because
   writePixel(x,y);                          int/int gives int and truncated the digits after decimal points.
   while(y>x) {                 //Get the x,y co-ordinates to draw the circle
      if(d<0)
         d = d + 2*x+3;
      else {
         d = d + 2*(x-y)+5;
         y--;
      }
      x++;
      writePixel(x,y);          //Point obtained from algorithm. Then other points are obtained by taking reflections
      writePixel(y,x);          //Point in octet-2
      writePixel(x,-y);         //Point in octet-4
      writePixel(y,-x);         //Point in octet-3
      writePixel(-x,-y);        //Point in octet-5
      writePixel(-y,-x);        //Point in octet-6
      writePixel(-x,y);         //Point in octet-8
      writePixel(-y,x);         //Point in octet-7
   }
   glFlush();
}

int main(int argc, char *argv[]) {
   printf("Enter the radius (0= < radius < 1000) : ");
   scanf("%d",&radius);
```



```
    glutInit(&argc,argv);
    init();
    glutDisplayFunc(drawCircle);
    glutMainLoop();
    return 0;
}
```



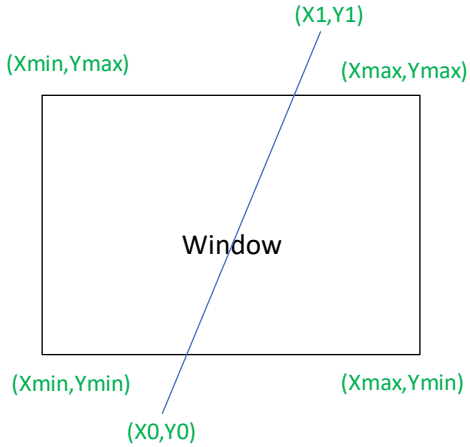
Output:



3. Liang Barsky Line Clipping Algorithm

Program to implement Liang-Barsky Line Clipping Algorithm. Line coordinates and viewport coordinates should be specified by the user. Display unclipped and clipped lines in separate viewports.

Program:

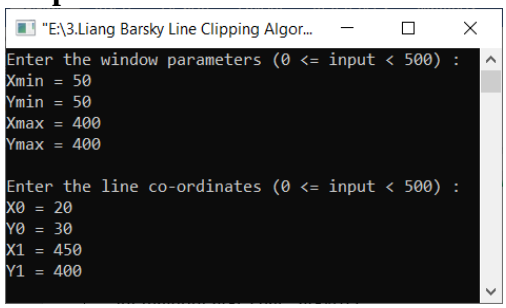


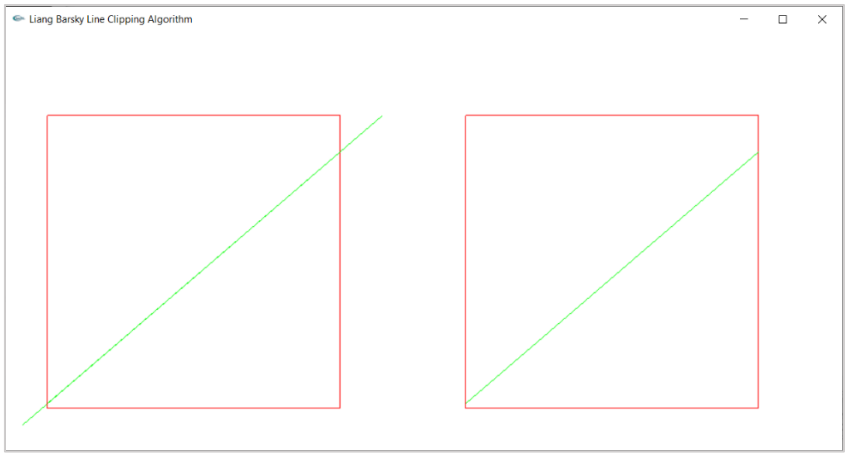

```
#include <windows.h>        //Don't include this for Ubuntu OS
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
int Xmin, Xmax, Ymin, Ymax, X0, Y0, X1, Y1;

void init() {
   glutInitWindowSize(1000, 500);
   glutCreateWindow("Liang Barsky Line Clipping Algorithm");
   glClearColor(1,1,1,1);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(0,1000,0,500);
   glPointSize(1);
}

double maximum(double p[], double r[]) {     //Finds maximum of qi/pi as per the algorithm
   int i;
   double max=0;              //Initialize max with 0 specified in algorithm i.e t1=0
   for(i=0;i<4;i++)           //Compare the max element with all the elements of array
      if(p[i]<0 && r[i]>max)  //As per algorithm qi/pi where pi < 0 are used to compute maximum
         max = r[i];
   return max;
}

double minimum(double p[], double r[]) {     //Finds minimum of qi/pi as per the algorithm
   int i;
   double min=1;      //Initialize min with 0 specified in algorithm i.e t2=1
   for(i=0;i<4;i++)
      if(p[i]>0 && r[i]<min)     //As per algorithm qi/pi where pi > 0 are used to compute minimum
         min = r[i];
  return min;
}
```

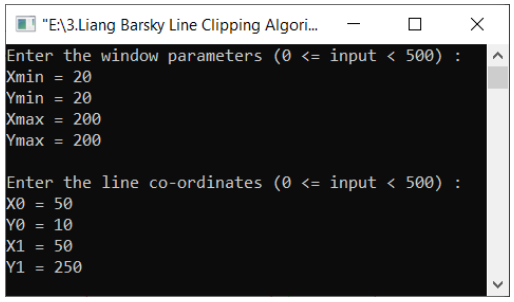
```
void drawLine(int type) {      //Draws the line (unclipped-0/clipped-1) based on passed parameter to function
  int leftMargin=0;            //If unclipped line then shows it in the left side of window so margin=0
  if(type==1)
    leftMargin=500;            //To show the output i.e clipped line in the right side, so margin = half the window size
  glColor3f(0,1,0);
  glBegin(GL_LINES);                  //Draws line between the specified pair of vertices Don't write GL_LINE
  glVertex2f(X0+leftMargin,Y0);       //Specify the points for drawing the lines
  glVertex2f(X1+leftMargin,Y1);
  glEnd();
}

void drawBoundary(int type) {         //Draws the viewport boundary based on passed parameter to function
  int leftMargin=0;
  if(type==1)
    leftMargin=500;
  glColor3f(1,0,0);
  glBegin(GL_LINE_LOOP);                  //Draws boundary i.e closed loop using the given points in sequence
  glVertex2f(Xmin+leftMargin, Ymin);      //List all the points in the order
  glVertex2f(Xmin+leftMargin, Ymax);      //Actually in lab manual they have multiplied by 4 i.e scaling so output
  glVertex2f(Xmax+leftMargin, Ymax);      //is shown in enlarged format. This doesn't make sense.
  glVertex2f(Xmax+leftMargin, Ymin);      //So here we displayed both the viewports beside each other with
  glEnd();                                //equal size
}

void LiangBarsky() {
  double dx,dy,p[4],q[4],r[4],t1,t2;          //Here r[i] is for storing q[i]/p[i]
  int i,reject=0;                             //Reject to identify whether to ignore line completely or not

  dx = X1-X0;
  dy = Y1-Y0;

  p[0] = -dx; q[0] = X0-Xmin;          //Left boundary
  p[1] = dx;  q[1] = Xmax-X0;          //Right boundary
  p[2] = -dy; q[2] = Y0-Ymin;          //Bottom boundary
  p[3] = dy;  q[3] = Ymax-Y0;          //Top boundary

  for(i=0; i<4; i++)                   //If value of any pi is zero then line is parallel to that i^th boundary
    if(p[i]==0 && q[i]<0) {            //If qi also zero for the same boundary then line is completely outside
      reject=1;                        //So reject that line
      break;
    }
    else
      r[i] = q[i]/p[i];                //Else compute q_i/p_i result

  t1 = maximum(p,r);                   //Find maximum of q_i/p_i and store in t1
  t2 = minimum(p,r);                   //Find minimum of q_i/p_i and store in t2
  drawBoundary(1);                     //Draw boundary for showing clipped line to the right of window

  if(reject==0 && t1<t2) {     //If line is not rejected then compute the clipped line coordinates as per algorithm
    X1 = round(X0+t2*dx);              //value is rounded up to convert obtained float value to nearest integer
    Y1 = round(Y0+t2*dy);
    X0 = round(X0+t1*dx);        //Don't change the sequence of these 4 lines because all 4 are dependent on
    Y0 = round(Y0+t1*dy);        //X0,Y0 values. So first update X1,Y1 then update X0,Y0 or use temporary variable
    drawLine(1);                       //Draw the clipped line to right of window
     }
}
```

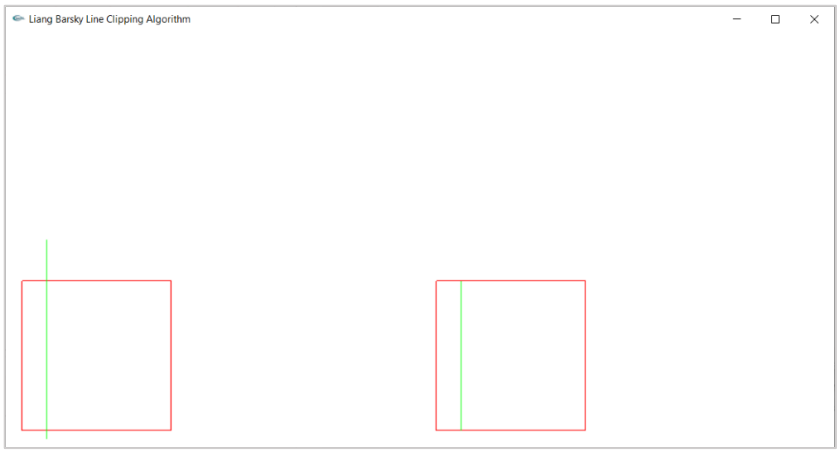
```
void display() {
   drawBoundary(0);                    //Draw boundary for unclipped line in left side of window
   drawLine(0);                        //Draw unclipped line in left side of window
   LiangBarsky();                      //Computes new line coordinates by applying algorithm
   glFlush();
}

int main(int argc,char *argv[]) {
   printf("Enter the window parameters (0 <= input < 500) :\n");          //Read window parameters
   printf("Xmin = "); scanf("%d",&Xmin);
   printf("Ymin = "); scanf("%d",&Ymin);
   printf("Xmax = "); scanf("%d",&Xmax);
   printf("Ymax = "); scanf("%d",&Ymax);
   printf("\nEnter the line co-\ordinates (0 <= input < 500) :\n");        //Read line parameters
   printf("X0 = "); scanf("%d",&X0);
   printf("Y0 = "); scanf("%d",&Y0);
   printf("X1 = "); scanf("%d",&X1);
   printf("Y1 = "); scanf("%d",&Y1);

   glutInit(&argc,argv);
   init();
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

Output:



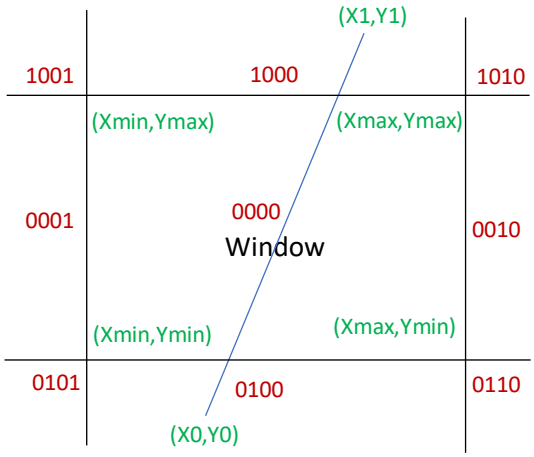






4. Cohen Sutherland Line Clipping Algorithm

Program to implement Cohen-Sutherland Line Clipping Algorithm. Line coordinates and viewport coordinates should be specified by the user. Display unclipped and clipped lines in separate viewports.



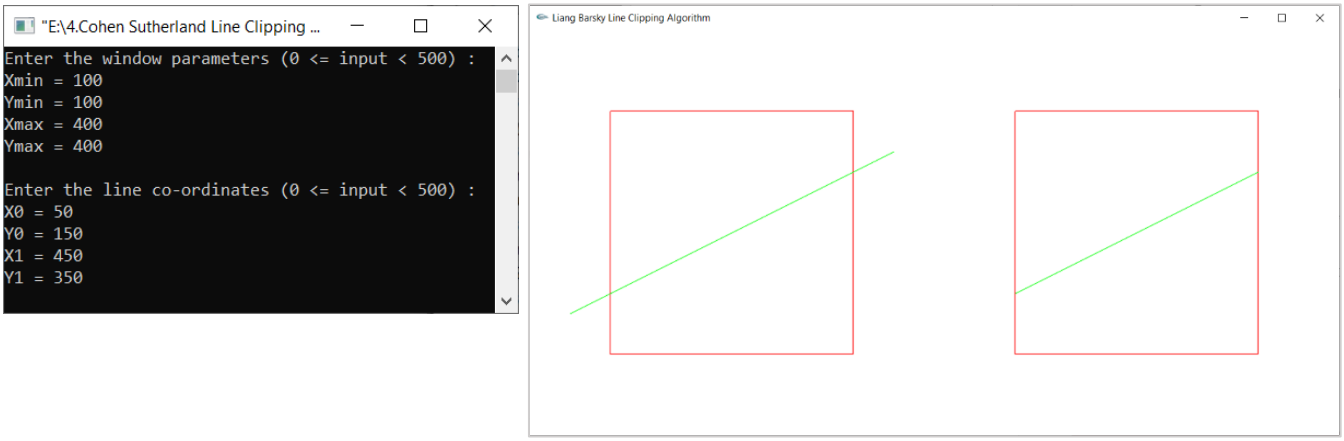
Program:

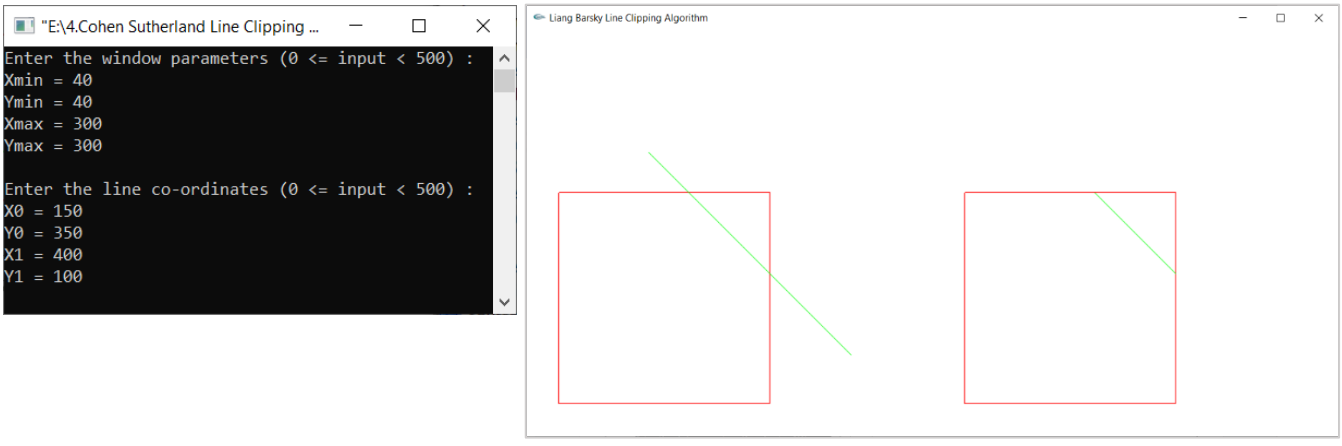
```
#include <windows.h>          //Don't include this for Ubuntu OS
#include <GL/glut.h>
#include <stdio.h>
const int INSIDE=0;           //Inside the window  (0000 – binary)
const int LEFT=1;             //Left of the window (0001)
const int RIGHT=2;            //Right of the window (0010)
const int BOTTOM=4;           //Bottom of the window (0100)
const int TOP=8;              //Top of the window (1000)
int Xmin, Xmax, Ymin, Ymax, X0, Y0, X1, Y1;

void init() {
   glutInitWindowSize(1000, 500);
   glutCreateWindow("Cohen-Sutherland Line Clipping Algorithm");
   glClearColor(1,1,1,1);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(0,1000,0,500);
   glPointSize(1);
}

void drawLine(int type) {
   int leftMargin=0;
   if(type==1)
      leftMargin=500;
   glColor3f(0,1,0);
   glBegin(GL_LINES);
   glVertex2f(X0+leftMargin,Y0);
   glVertex2f(X1+leftMargin,Y1);
   glEnd();
}

void drawBoundary(int type) {
   int leftMargin=0;
   if(type==1)
      leftMargin=500;
   glColor3f(1,0,0);
   glBegin(GL_LINE_LOOP);
   glVertex2f(Xmin+leftMargin, Ymin);
   glVertex2f(Xmin+leftMargin, Ymax);
   glVertex2f(Xmax+leftMargin, Ymax);
   glVertex2f(Xmax+leftMargin, Ymin);
   glEnd();
}

int computeCode(int x, int y) {        //Computes the code to find out where the end points of the line lies
   int code = INSIDE;                  //Assume the point is inside the window
   if(x<Xmin)                          //Outside left boundary
      code |= LEFT;    //Shorthand of code = code | LEFT. The operator is bitwise OR.  1|4 = 0001|0100 = 0101 = 5
   if(x>Xmax)                          //Outside right boundary
      code |= RIGHT;
   if(y<Ymin)                          //Outside bottom boundary
```

```
    code |= BOTTOM;
  if(y>Ymax)                          //Outside top boundary
    code |= TOP;
  return code;
}

void CohenSutherland() {
  int accept=0, code0, code1;
  code0=computeCode(X0,Y0);           //Compute code for 1st end of the line
  code1=computeCode(X1,Y1);           //Compute code for 2nd end of the line
  while(1) {                          //Until line comes within window
    if(!(code0|code1)){               //If both points are inside i.e bitwise-OR of codes==0
      accept = 1;                     //Accept the line
      break;                          //Accepted break out of loop and draw line
    }
    else if(code0&code1)              //If both points outside same boundary discard line
      break;
    else {
      int code, X, Y;
      double slope;
      code = code0? code0:code1;      //Take one code at a time. If one point is inside, then consider second
      slope = (Y1-Y0)/(double)(X1-X0);//Find slope – here also one operand is type casted to double
      if(code & LEFT) {               //adjust left boundary i.e find intersection of left boundary
        X = Xmin;
        Y = slope*(Xmin-X0) + Y0;     //Find intersection as per the algorithm
      }                               //Adjust right boundary
      else if(code & RIGHT) {
        X = Xmax;
        Y = slope*(Xmax-X0) + Y0;
      }
      else if(code & BOTTOM) {        //Adjust bottom boundary
        X = (Ymin-Y0)/slope + X0;
        Y = Ymin;
      }
      else if(code & TOP) {           //Adjust top boundary
        X = (Ymax-Y0)/slope + X0;
        Y = Ymax;
      }
      if(code == code0) {             //If first point considered, then update the coordinates and compute code
        X0 = X; Y0 = Y;
        code0 = computeCode(X0,Y0);
      }
      else {                          //If second point consideres, update its values
        X1 = X; Y1= Y;
        code1 = computeCode(X1,Y1);
      }
    }
  }
  drawBoundary(1);
  if(accept)                          //If line is accepted then draw the clipped line
    drawLine(1);
}
```

```
void display() {
   drawBoundary(0);
   drawLine(0);
   CohenSutherland();                    //Computes new line co-ordinates by applying algorithm
   glFlush();
}

int main(int argc,char *argv[]) {
   printf("Enter the window parameters (0 <= input < 500) :\n");
   printf("Xmin = "); scanf("%d",&Xmin);
   printf("Ymin = "); scanf("%d",&Ymin);
   printf("Xmax = "); scanf("%d",&Xmax);
   printf("Ymax = "); scanf("%d",&Ymax);

   printf("\nEnter the line co-ordinates (0 <= input < 500) :\n");
   printf("X0 = "); scanf("%d",&X0);
   printf("Y0 = "); scanf("%d",&Y0);
   printf("X1 = "); scanf("%d",&X1);
   printf("Y1 = "); scanf("%d",&Y1);

   glutInit(&argc,argv);
   init();
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

Output:

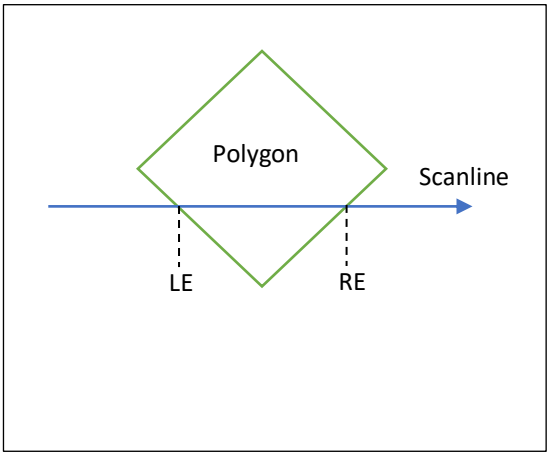




5. Scanline Polygon Filling Algorithm

Program to fill the polygon using Scan-Line Polygon Filling Algorithm. Vertices of the polygon should be specified by the user

Version-1: (Capable of drawing only quadrilateral)

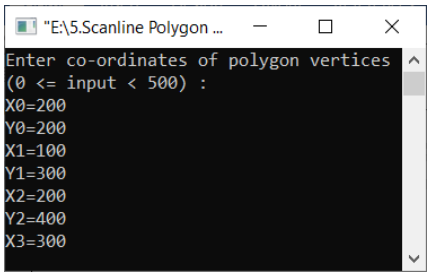


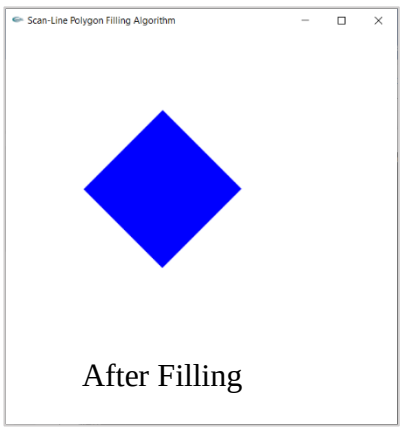
Program:

```
#include <windows.h>        //Don't include this for Ubuntu OS
#include <GL/glut.h>
#include <stdio.h>
int X0,Y0,X1,Y1,X2,Y2,X3,Y3;          //Vertices of quadrilateral

void init() {
   glutInitWindowSize(500, 500);
   glutCreateWindow("Scan-Line Polygon Filling Algorithm");
   glClearColor(1,1,1,1);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(0,500,0,500);
   glColor3f(0,0,1);
   glPointSize(1);
}

void edgeDetect(int x1,int y1,int x2,int y2,int LE[],int RE[]) { //Detects starting and ending edge along scan-line
    double slopeInverse;
    int i,temp;
    double x;
    if(y2<y1) {                 //If beginning point of edge is below ending point then swap the end points
       temp=y1;  y1=y2;  y2=temp;
       temp=x1;  x1=x2;  x2=temp;
    }
    if(y1!=y2)                  //Compute inverse slope when line is not parallel to x axis
       slopeInverse = (x2-x1)/(double)(y2-y1);
    else
       slopeInverse = x2-x1;          //Inverse slope when line is parallel to x-axis
x=x1;                                 //Start from first end of the polygon edge
    for(i=y1; i<=y2; i++) {           //For all the points of edges in vertical direction
       if(x<LE[i])                 //Compute the beginning and ending x-coordinate of the figure along the scan-line
          LE[i]=x;
       if(x>RE[i])                 //If x value is less than existing LE[i] then update LE[i]
          RE[i]=x;                 //If x value is greater than existing RE[i] then update RE[i]
       x+=slopeInverse;            //Add slopeInverse to x to get next x value
    }
}

void drawPixel(int x,int y) {
    glColor3f(0,0,1);
    glBegin(GL_POINTS);
    glVertex2f(x,y);
    glEnd();
}

void scanLineFill() {
    int i,j,LE[500],RE[500];   //Arrays to store left edge and right edge for each scanline. Length should be equal to
    for(i=0; i<500; i++) {              // Ymax set in gluOrtho2D function
       LE[i]=500;                       //Initialize LE to maximum width value (Xmax) of window
```

```
        RE[i]=0;                    //Initialize RE to minimum width value (Xmin) of window
    }
    edgeDetect(X0,Y0,X1,Y1,LE,RE);      //For each edges compute the LE and RE values
    edgeDetect(X1,Y1,X2,Y2,LE,RE);
    edgeDetect(X2,Y2,X3,Y3,LE,RE);
    edgeDetect(X3,Y3,X0,Y0,LE,RE);
    for(i=0; i<500; i++)                //For each scanline bottom to top
      if(LE[i]<=RE[i])                  //If LE<=RE i.e polygon has to be filling along that scanline
        for(j=LE[i]; j<=RE[i]; j++) {   //Draw all the points starting from LE to RE along that scanline
            drawPixel(j,i);
            glFlush();
        }
}

void display() {
    glBegin(GL_LINE_LOOP);              //Draw boundary of polygon before filling
    glVertex2f(X0,Y0);
    glVertex2f(X1,Y1);
    glVertex2f(X2,Y2);
    glVertex2f(X3,Y3);
    glEnd();
    scanLineFill();                     //Fill the polygon boundary
    glFlush();
}

int main(int argc,char *argv[]) {
   printf("Enter co-ordinates of polygon vertices (0 <= input < 500) :\n");      //Read 4 vertices of quadrilateral
   printf("X0="); scanf("%d",&X0);
   printf("Y0="); scanf("%d",&Y0);
   printf("X1="); scanf("%d",&X1);
   printf("Y1="); scanf("%d",&Y1);
   printf("X2="); scanf("%d",&X2);
   printf("Y2="); scanf("%d",&Y2);
   printf("X3="); scanf("%d",&X3);
   printf("Y3="); scanf("%d",&Y3);

   glutInit(&argc,argv);
   init();
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

Output:





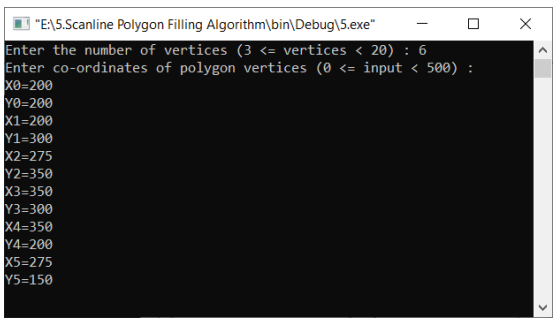
Version-2: (Capable of any polygon where a scan line passes through maximum of 2 edges)

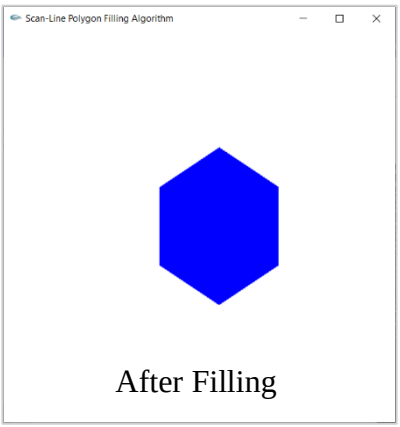
Program:

```
#include <windows.h>       //Don't include this for Ubuntu OS
#include <GL/glut.h>
#include <stdio.h>
int V,X[20],Y[20];    //Here maximum 20 vertices are supported. You can increase this number if required

void init() {
   glutInitWindowSize(500, 500);
   glutCreateWindow("Scan-Line Polygon Filling Algorithm");
   glClearColor(1,1,1,1);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(0,500,0,500);
   glColor3f(0,0,1);
   glPointSize(1);
}

void edgeDetect(int x1,int y1,int x2,int y2,int LE[],int RE[]) {          //Detect LE and RE for the edge
    double slopeInverse, x;
    int i,temp;

2   if(y2<y1) {
      temp=y1;   y1=y2;   y2=temp;
      temp=x1;   x1=x2;   x2=temp;
    }
    if(y1!=y2)
      slopeInverse = (x2-x1)/(double)(y2-y1);
    else
      slopeInverse = x2-x1;
    x=x1;
    for(i=y1; i<=y2; i++) {
      if(x<LE[i])
        LE[i]=x;
      if(x>RE[i])
        RE[i]=x;
      x+=slopeInverse;
    }
}

void drawPixel(int x,int y) {
    glColor3f(0,0,1);
    glBegin(GL_POINTS);
    glVertex2f(x,y);
    glEnd();
}

void scanLineFill() {
    int i,j,LE[500],RE[500];
    for(i=0; i<500; i++) {
      LE[i]=500;
      RE[i]=0;
    }
    for(i=0; i<V; i++)          //Detect LE and RE for all the input edges
      edgeDetect(X[i],Y[i],X[(i+1)%V],Y[(i+1)%V],LE,RE);   //Modulus is taken to loop back from last edge to first
```

```
                                        //i.e We pass values (V0,V1), (V1-V2) so on upto (Vn-1, V0) to get Vo we use %
    for(i=0; i<500; i++)
      if(LE[i]<=RE[i])
        for(j=LE[i]; j<=RE[i]; j++) {
            drawPixel(j,i);
            glFlush();
        }
}

void display() {
   int i;
   glBegin(GL_LINE_LOOP);
   for(i=0; i<V; i++)                    //Draw boundary using all the vertices
     glVertex2f(X[i],Y[i]);
   glEnd();
   glFlush();
   scanLineFill();
}

int main(int argc,char *argv[]) {
   int i;
   printf("Enter the number of vertices (3 <= vertices < 20) : ");          //Read number of vertices
   scanf("%d",&V);
   printf("Enter co-ordinates of polygon vertices (0 <= input < 500) :\n");  //Read coordinates of all vertices
   for(i=0; i<V; i++) {
     printf("X%d=",i); scanf("%d",&X[i]);
     printf("Y%d=",i); scanf("%d",&Y[i]);
   }

    glutInit(&argc,argv);
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Output:



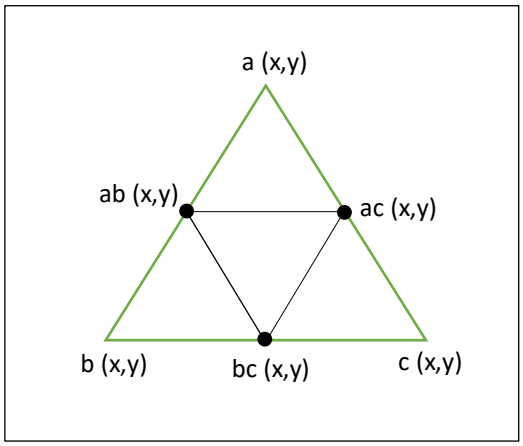


6. Sierpinski Gasket Algorithm

Program to recursively sub-divide a triangle to form 2D Sierpinski Gasket. The number of recursive steps should be specified by the user.

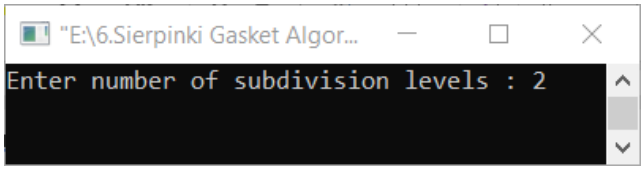
Program:

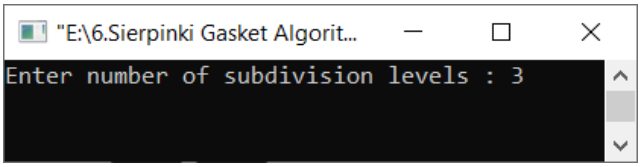
```
#include <windows.h>          //Don't include this for Ubuntu OS
#include <GL/glut.h>
#include <stdio.h>
int n;                        //Number of division levels

void init() {
   glutInitWindowSize(500, 500);
   glutCreateWindow("Sierpinski Gasket Algorithm");
   glClearColor(1,1,1,1);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(0,7,0,7);
   glColor3f(0,0,0);          //Color of interior triangles
}

void triangle(double a[],double b[], double c[]) {          //Draws a triangle using three vertices
   glBegin(GL_TRIANGLES);
   glVertex2f(a[0],a[1]);
   glVertex2f(b[0],b[1]);
   glVertex2f(c[0],c[1]);
   glEnd();
}

void drawTriangle(double a[],double b[],double c[],double k) {      //Subdivide triangle based on algorithm
   double ab[2],ac[2],bc[2];                  //Used to store the co-ordinates of three vertices of divided triangle
   int i;
   if(k>0) {                                  //If number of remaining subdivisions not zero
      for(i=0;i<2;i++) {
         ab[i]=(a[i]+b[i])/2;                 //Here idea is to find mid points all the three sides of current triangle
         bc[i]=(b[i]+c[i])/2;                 //Then use these three points to draw new smaller triangles
         ac[i]=(a[i]+c[i])/2;                 //These three lines find mid points of all the three edges
      }
      drawTriangle(a,ab,ac,k-1);              //Call same function recursively with new coordinates and new sublevel value
      drawTriangle(b,bc,ab,k-1);              //Pass one of the vertex of bigger triangle and midpoints of adjacent sides
      drawTriangle(c,ac,bc,k-1);
  }
   else
      triangle(a,b,c);                        //If k reaches zero, no further division. So just draw the outer triangle
}

void display() {                              //Initialize vertices of outermost triangle. As per problem statement these need not be
   double a[2]={1,1}, b[2]={6,1}, c[2]={3.5,5};   // taken from user. Directly assign such that triangle covers window
   drawTriangle(a,b,c,n);                     //Draw triangle using Sierpinski algorithm
   glFlush();
}

int main(int argc,char *argv[]) {
   printf("Enter number of subdivision levels : ");          //Input number of division levels
   scanf("%d",&n);
   glutInit(&argc,argv);
   init();
```

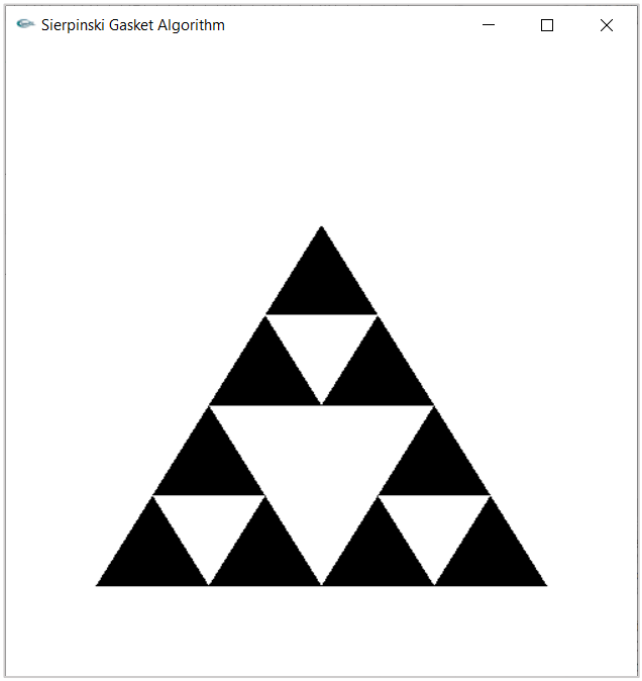


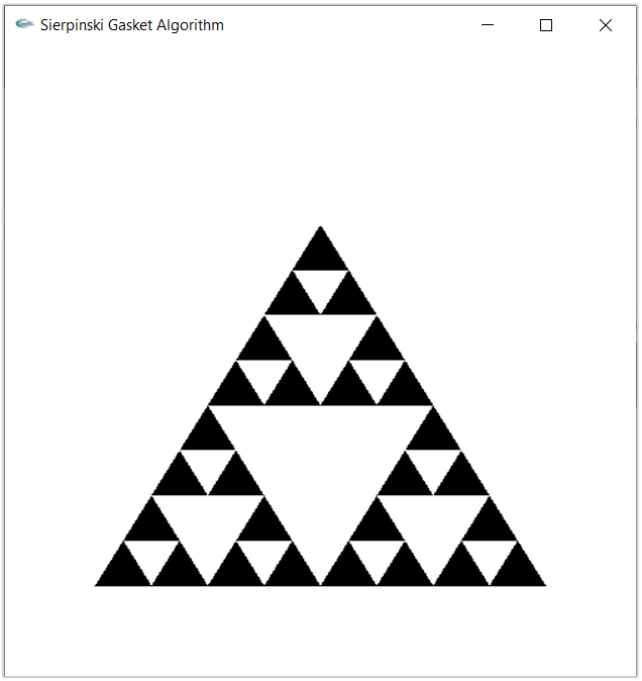
```
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Output:







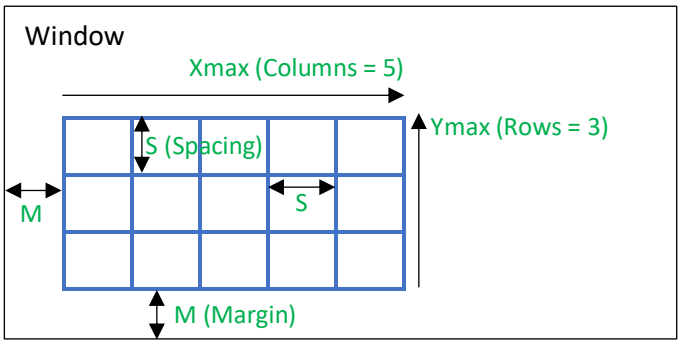


7. Rectangular Mesh

*Program to display a m*n rectangular mesh. Number of rows and columns of the mesh generation should be taken from the user.*

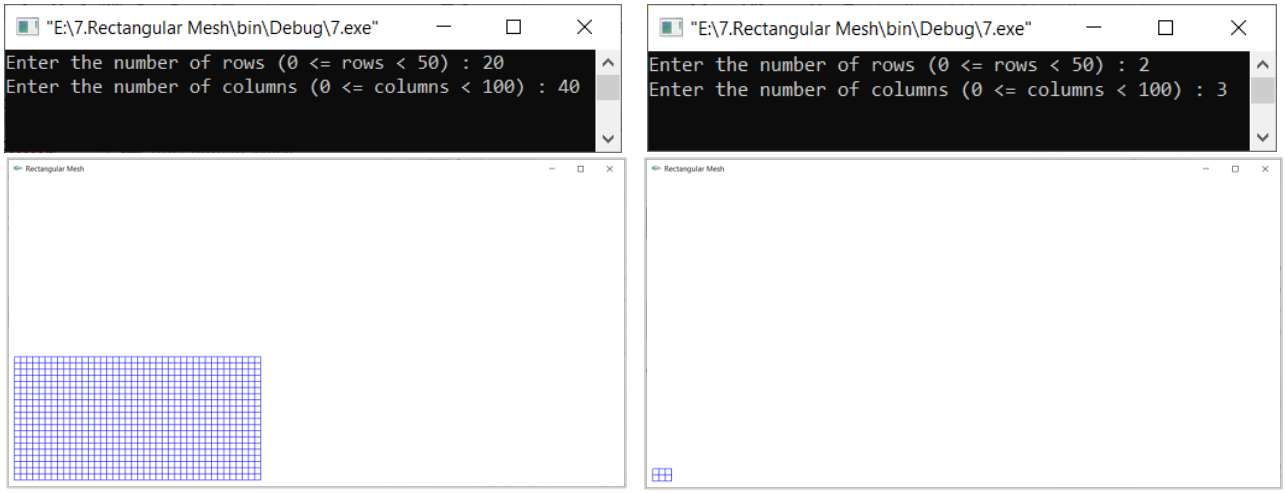
Program:

```
#include <windows.h>        //Don't include this for Ubuntu OS
#include <GL/glut.h>
#include <stdio.h>
#define S 10                //Spacing between two rows and columns while drawing the mesh
#define M 10                //Bottom and left margin to give some space around the mesh
int Xmax,Ymax;              //Variable to store rows and columns. Xmax means columns (along X-axis) and Ymax are rows
                    //In lab manual they used separate spacing and margin for row and column. You can refer that too

void init() {
    glutInitWindowSize(1000, 500);
    glutCreateWindow("Rectangular Mesh");
    glClearColor(1,1,1,1);
    glClear(GL_COLOR_BUFFER_BIT);
    gluOrtho2D(0,1000,0,500);
    glColor3f(0,0,1);
}

void display() {
    int i,j;
```



```
    for(i=0; i<Xmax; i++)          //For each columns i.e 0 to Xmax-1
      for(j=0; j<Ymax; j++) {      //For each rows i.e 0 to Ymax-1
        glBegin(GL_LINE_LOOP);     //Draw the smaller cells of the mesh
        glVertex2f(i*S+M, j*S+M);  //Multiply cell number with spacing and add the margin to get the position
        glVertex2f((i+1)*S+M, j*S+M);          //Previous one is bottom-left vertex. This is bottom-right
        glVertex2f((i+1)*S+M, (j+1)*S+M);      //Top-right
        glVertex2f(i*S+M, (j+1)*S+M);          //top-left
        glEnd();
      }
    glFlush();
}

int main(int argc, char *argv[]) {
    printf("Enter the number of rows (0 <= rows < 50) : ");       //Read number of rows
    scanf("%d",&Ymax);
    printf("Enter the number of columns (0 <= columns < 100) : ");      //Read number of columns
    scanf("%d",&Xmax);

    glutInit(&argc,argv);
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

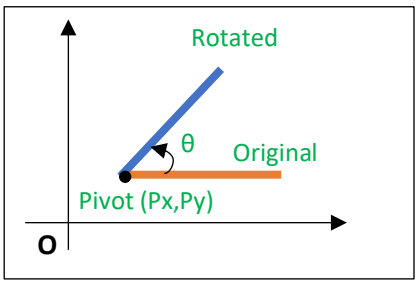
Output:



8. Rotation of Random Figure

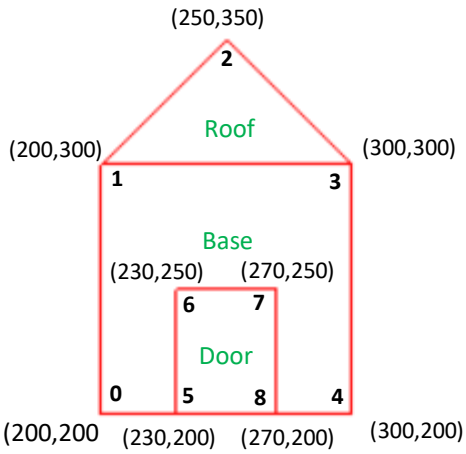
Program to rotate a random figure about a fixed point with an angle theta using transformation matrices. Vertices of random figure must be identified manually and stored in an array. Pivot point and angle of rotation should be specified by the user.

Program:




```
#include <windows.h>        //Don't include this for Ubuntu OS
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>

double theta;          //Angle theta
int Px,Py;             //Pivot point for rotation
float figure[9][2]={{200,200},{200,300},{250,350},{300,300},{300,200},{230,200},{230,250},{270,250},{270,200}};
//Here they taken random figure as house but examiner may ask any figure. So you have to draw manually in paper
and identify the (x,y) coordinates for all the vertices in the figure and store it in the array and draw accordingly
```

```
void init() {
   glutInitWindowSize(500,500);
   glutCreateWindow("2D Rotation");
   glClearColor(1,1,1,1);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(0,500,0,500);
}

void drawFigure() {
   glBegin(GL_LINE_LOOP);   //Draw Base
   glVertex2fv(figure[0]);
   glVertex2fv(figure[1]);
   glVertex2fv(figure[3]);
   glVertex2fv(figure[4]);
   glEnd();
   glBegin(GL_LINE_LOOP);   //Draw roof
   glVertex2fv(figure[1]);
   glVertex2fv(figure[2]);
   glVertex2fv(figure[3]);
   glEnd();
   glBegin(GL_LINE_LOOP);   //Draw door
   glVertex2fv(figure[5]);
   glVertex2fv(figure[6]);
   glVertex2fv(figure[7]);      //Here glVertex2fv takes vector i.e it takes array of values. You can also use
   glVertex2fv(figure[8]);      //glVertex2f(figure[8][0], figure[8][1]);
   glEnd();
}

void display() {
   float m[16];                 //Creating transformation matrix as shown – Here we use 1D representation of matrix
   m[0] = cos(theta);
   m[1] = sin(theta);
   m[4] = -sin(theta);
   m[5] = cos(theta);
   m[12] = -Px*(cos(theta)-1)+Py*sin(theta);
   m[13] = -Py*(cos(theta)-1)-Px*sin(theta);
   m[10]=m[15]=1;
   m[2]=m[3]=m[6]=m[7]=m[8]=m[9]=m[11]=m[14]=0;      //These positions are to be set to 0 as per the matrix
   glColor3f(1,0,0);            //Set color for original figure (Red)
   drawFigure();                //Draw original figure
   glPushMatrix();              //Push a existing matrix to the transformation buffer
   glMultMatrixf(m);            //Multiply the matrix with rotational matrix to obtain rotated figure
   glColor3f(0,0,1);            //Set color for rotated figure (Blue)
   drawFigure();                //Draw the rotated figure
   glPopMatrix();               //Pop the matrix after displaying the rotated figure
   glFlush();
}

int main(int argc,char *argv[]) {
   printf("Enter the angle of rotation (in degrees) : ");      //Input angle of rotation in degrees
   scanf("%lf",&theta);                                        //%lf for reading double value
   printf("\nEnter Pivot point for rotation :\n");             //Input pivot point for rotation
   printf("Px="); scanf("%d",&Px);
   printf("Py="); scanf("%d",&Py);
```

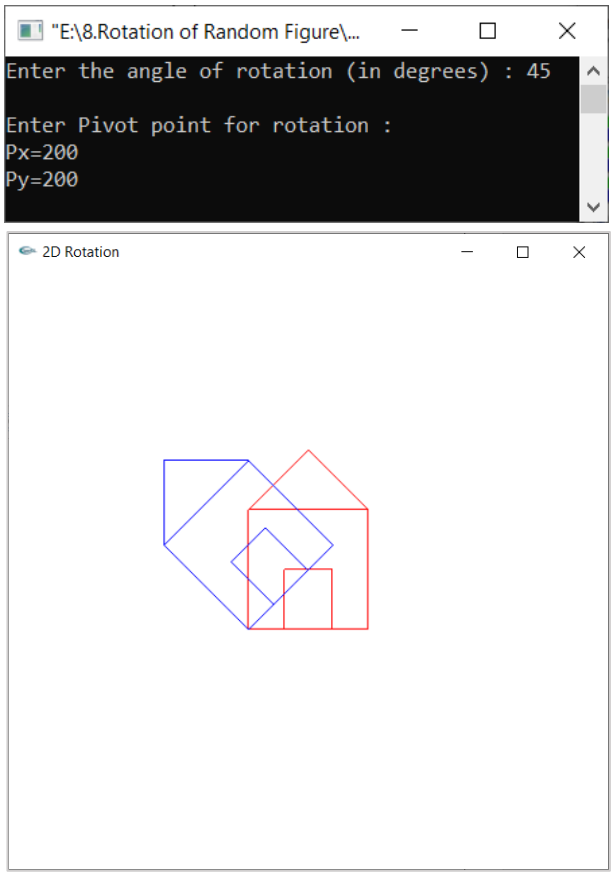


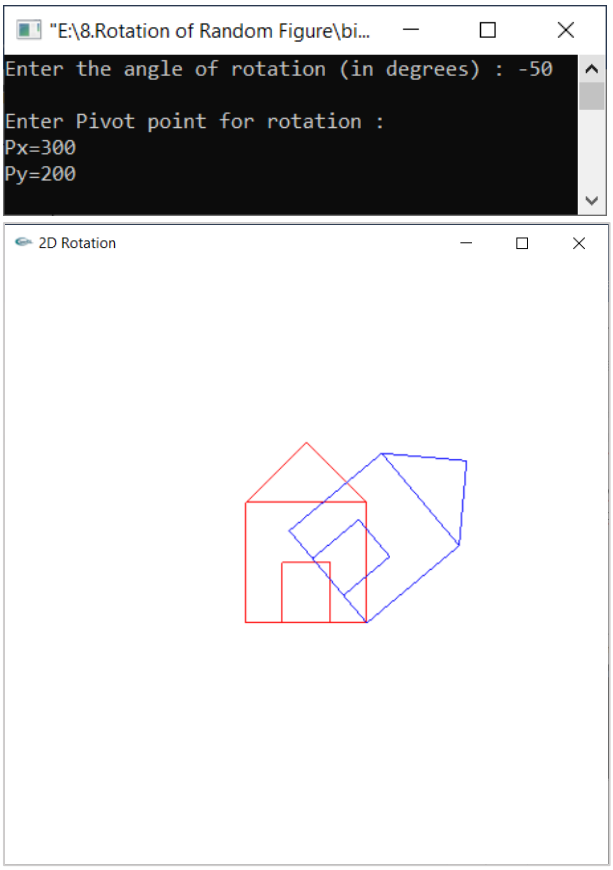
You can store vertices in the array in any order. Just make sure you use them properly while drawing

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -P_x(\cos\theta-1)+P_y\sin\theta & -P_y(\cos\theta-1)-P_x\sin\theta & 0 & 1 \end{bmatrix}$$

```
    theta = theta*3.142/180;                    //Convert degree to radian since glut uses radian
    glutInit(&argc,argv);
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Output:





9. Mouse & Keyboard Functions

Program to implement the mouse functions (LEFT CLICK, RIGHT CLICK) and keyboard functions (T o t, S or s and P or p) through OpenGL functions such that for each function some object has to be rendered.

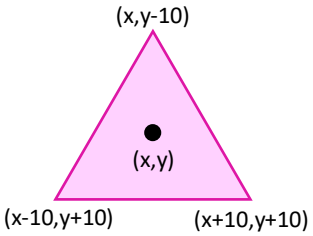
Program:

```
#include <windows.h>        //Don't include this for Ubuntu OS
#include <GL/glut.h>
#include <stdio.h>

void init() {
    glutInitWindowSize(500,500);
    glutCreateWindow("Mouse and Keyboard Operations");
    glClearColor(1,1,1,1);
    glClear(GL_COLOR_BUFFER_BIT);
    gluOrtho2D(0,500,500,0);          //Here for y axis they take bottom as 500 because when mouse returns position
}                                     //of cursor it starts counting from top of the window

void drawPoint(int x,int y) {   //Draw point at (x,y)
    glColor3f(1,0,0);           //Color of point - red
    glPointSize(10);            //Thickness of point – 10pixels
```

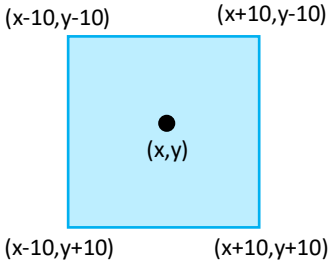
```
    glBegin(GL_POINTS);
    glVertex2f(x,y);
    glEnd();
    glFlush();
}
```


```
void drawLine(int x,int y) {    //Draw a line at (x,y)
    glColor3f(0,0,1);           //Color of line - blue
    glLineWidth(10);            //Thickness of line – 10pixels
    glBegin(GL_LINES);
    glVertex2f(x-25,y);         //Refer the diagram for vertice determination
    glVertex2f(x+25,y);
    glEnd();
    glFlush();
}
```

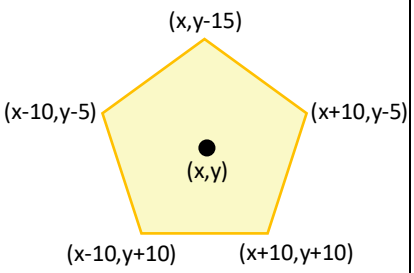

```
void drawTriangle(int x,int y) {     //Draw a triangle at (x,y)
    glColor3f(1,0,1);                //Color of triangle - pink
    glBegin(GL_TRIANGLES);
    glVertex2f(x,y-10);              //Refer the diagram for vertice determination
    glVertex2f(x-10,y+10);      //Here to move up we need to subtract from Y and
    glVertex2f(x+10,y+10);      //to move down add to Y. This is contradiction to
    glEnd();                    //cartesian plane. But we use so because we defined
    glFlush();                  //y-axis from top to bottom (Please refer init() function)
}
```




```
void drawSquare(int x,int y) {       //Draw a square at (x,y)
    glColor3f(0,1,1);                //Color of square - cyan
    glBegin(GL_POLYGON);
    glVertex2f(x-10,y-10);           //Refer the diagram for vertice determination
    glVertex2f(x-10,y+10);
    glVertex2f(x+10,y+10);
    glVertex2f(x+10,y-10);
    glEnd();
    glFlush();
}
```



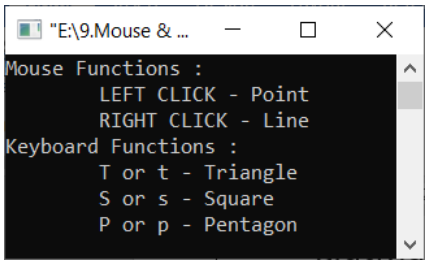

```
void drawPentagon(int x,int y) {     //Draw a pentagon at (x,y)
    glColor3f(1,1,0);                //Color of pentagon - yellow
    glBegin(GL_POLYGON);
    glVertex2f(x,y-15);              //Refer the diagram for vertice determination
    glVertex2f(x-10,y-5);
    glVertex2f(x-10,y+10);
    glVertex2f(x+10,y+10);
    glVertex2f(x+10,y-5);
    glEnd();
    glFlush();
}
```

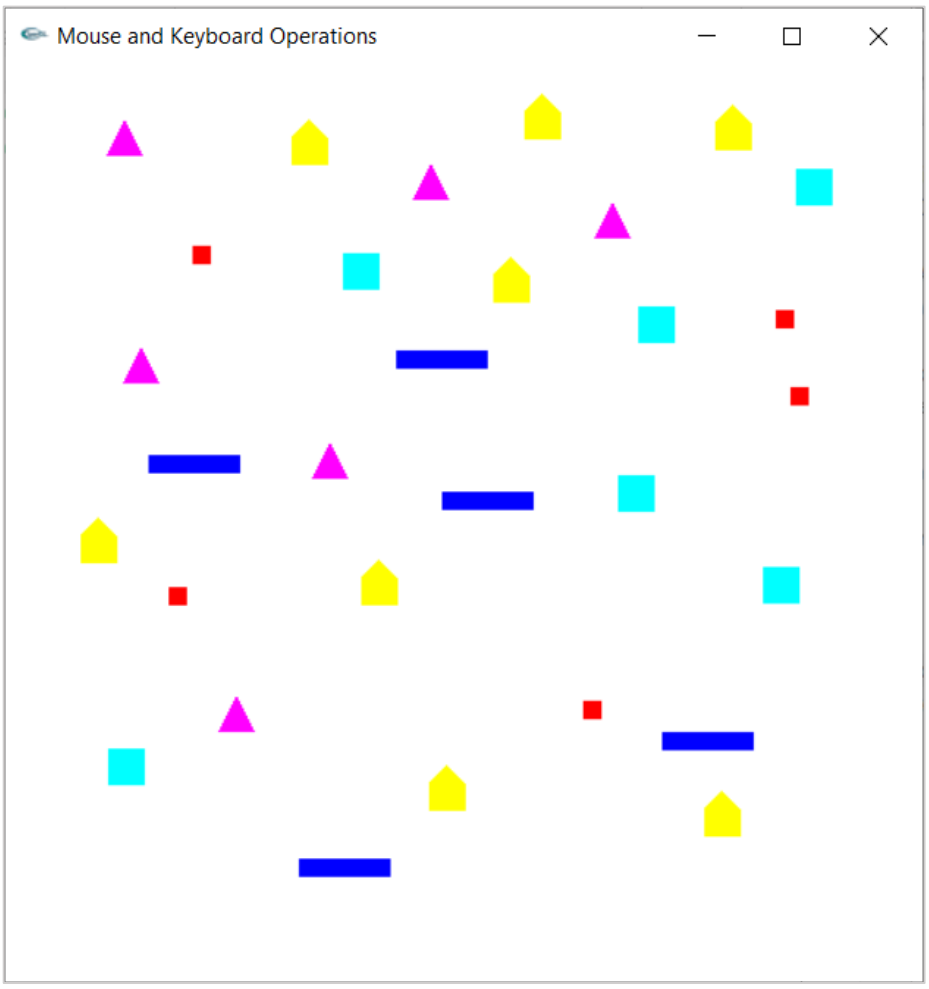



```
void mouse(int button,int state,int x,int y) {     //Callback function for mouse event. It return clicked mouse
                                                   //button code, state and position of cursor at the time of click
    if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)     //If left button clicked and it down
        drawPoint(x,y);                                  //draw point at (x,y)
    if(button==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)    //If right button clicked and it is down
        drawLine(x,y);                                   //draw line at (x,y)
}
```

```
void keyboard(unsigned char key,int x,int y) {          //Callback function for keyboard event. It returns code of
                                                        // key and position of mouse cursor when key is pressed
 if(key=='t' || key=='T')                 //If letter T (uppercase/lowercase) is pressed
   drawTriangle(x,y);                             //draw triangle at (x,y)
 if(key=='s' || key=='S')                 //If letter S (uppercase/lowercase) is pressed
   drawSquare(x,y);                               //draw square at (x,y)
 if(key=='p' || key=='P')                 //If letter P (uppercase/lowercase) is pressed
   drawPentagon(x,y);                             //draw pentagon at (x,y)
}

void display() {
   glFlush();
}

int main(int argc, char *argv[]) {
   printf("Mouse Functions :\n\tLEFT CLICK - Point\n\tRIGHT CLICK - Line\n");          //Display instructions
   printf("Keyboard Functions :\n\tT or T - Triangle\n\tS or s - Square\n\tP or p - Pentagon\n");//these can be ignored
   glutInit(&argc, argv);
   init();
   glutDisplayFunc(display);
   glutMouseFunc(mouse);              //Register listener for mouse event
   glutKeyboardFunc(keyboard);        //Register listener for keyboard event
   glutMainLoop();
   return 0;
}
```

Output:





10. Spinning Color Cube

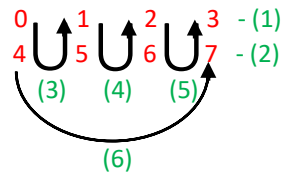
Program to draw color cube and spin it using OpenGL transformation matrices along x,y and z axis. Rotations axes are controlled using mouse clicks (LEFT CLICK, SCROLL WHEEL CLICK , RIGHT CLICK).

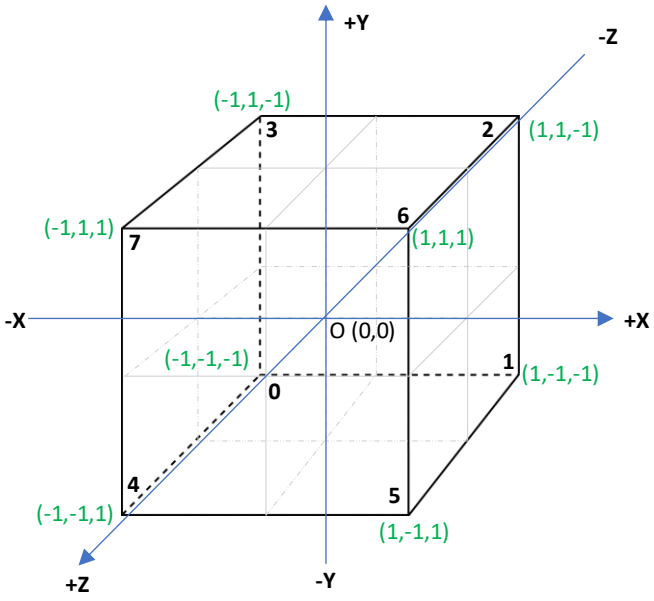
Program:

```
#include <windows.h>
#include <GL/glut.h>

float vertices[][3]={{-1,-1,-1},{1,-1,-1},{1,1,-1},{-1,1,-1},{-1,-1,1},{1,-1,1},{1,1,1},{-1,1,1}};   //8 Vertices of cube
float normals[][3]={{-1,-1,-1},{1,-1,-1},{1,1,-1},{-1,1,-1},{-1,-1,1},{1,-1,1},{1,1,1},{-1,1,1}};   //Vertex normals
float colors[][3]={{0,0,0},{0,0,1},{0,1,0},{0,1,1},{1,0,0},{1,0,1},{1,1,0},{1,1,1}};                 //Color at each vertex
float theta[]={0,0,0};          //Angle of rotation along x, y and z axes respectively
int axis=2;    //Initialize rotation along Z-axis (2 stand for Z axis)
               //Colors above remebered using 0-7  3-bit binary numbers and for vertices refer the diagram below

void init() {
   glutInitWindowSize(500,500);
   glutCreateWindow("Spinning Cube");
   glClearColor(0,0,0,0);
   glClear(GL_COLOR_BUFFER_BIT);
   glEnable(GL_DEPTH_TEST);          //Sense the depth of the vertices and edges since this is 3D figure
}

void polygon(int a,int b,int c,int d) {   //Draw a single face of the cube
   glBegin(GL_POLYGON);
   glColor3fv(colors[a]);             //Set color of vertex
   glNormal3fv(normals[a]);           //Set vertext normal
   glVertex3fv(vertices[a]);          //Specify the vertex
   glColor3fv(colors[b]);
   glNormal3fv(normals[b]);
   glVertex3fv(vertices[b]);
   glColor3fv(colors[c]);
   glNormal3fv(normals[c]);
   glVertex3fv(vertices[c]);
   glColor3fv(colors[d]);
   glNormal3fv(normals[d]);
   glVertex3fv(vertices[d]);
   glEnd();
}

void colorCube() {              //Creates color cube
   polygon(0,1,2,3);            //Back face of cube
   polygon(4,5,6,7);            //Front face of cube
   polygon(0,4,5,1);            //Bottom face of cube
   polygon(1,5,6,2);            //Right face of cube
   polygon(2,6,7,3);            //Top face of cube
   polygon(0,4,7,3);            //Left face of cube
}

void display() {
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);  //Clear buffer for color and depth drawing
   glLoadIdentity();                  //Load identity matrix for rotation
   glRotatef(theta[0],1,0,0);         //rotate along X-axis with angle theta[0]   (angle, x, y z);
   glRotatef(theta[1],0,1,0);         //rotate along Y-axis with angle theta[1]
   glRotatef(theta[2],0,0,1);         //rotate along Z-axis with angle theta[2]
```

Note: Normals will help Graphic Driver to know the directions of each faces of the cube. Here program gives output even if you don't specify the normals. So if you are finding it difficult you can ignore normal array and glNormal3fv() function. If you encounter any error please add the normal

How to remember?
First write (0,1,2,3) and (4,5,6,7) in sequence then consider 2 adjacent columns and traverse in U shape to get remaining





Here the cube is symmetric about all the 3 axes i.e origin lies in the center of the cube

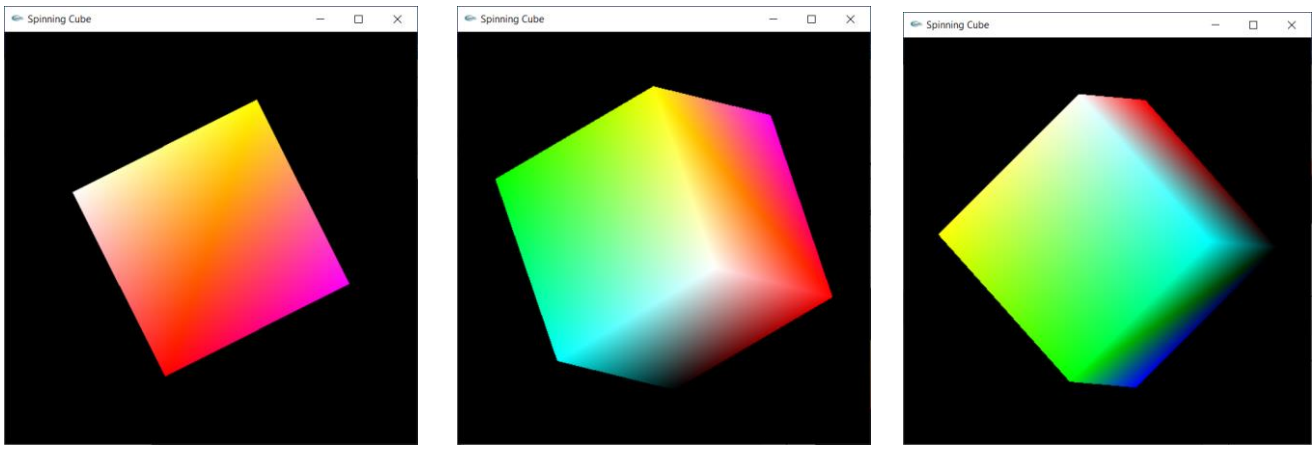
You can remember these vertex number to store the vertices in the array

```
  colorCube();                    //draw color cube. The function rotates and draws cube continuously
  glFlush();
}

void spinCube() {                 //Spins cube continuously along currently chosen axis
  theta[axis]+=0.01;              //Increment angle. Increase or decrease the step angle to increase/decrease speed
  if(theta[axis]>360)             //If angle crosses 360 degree i.e one rotation, subtract 360 degree to prevent overflow
    theta[axis]-=360;
  glutPostRedisplay();            //Redisplay cube with incremented angle
}

void mouse(int btn,int state,int x,int y) {                          //Callback for mouse events
  if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)   axis=0;             //On LEFT CLICK, axis = X-axis
  if(btn==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN)   axis=1;          //On SCROLL WHEEL CLICK axis = Y-axis
  if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)   axis=2;           //On RIGHT CLICK axis = Z-axis
}

void reshape(int w,int h) {               //Listener when window parameters changed this will called
  glViewport(0,0,w,h);                        //Set viewport to new width and hight of window
  glMatrixMode(GL_PROJECTION);                //Set matrix mode to projection matrix
  glLoadIdentity();
  if(w<=h)
    glOrtho(-2, 2, -2*(float)h/w, 2*(float)h/w, -2, 2);          //if width<=height, then adjust cube horizontally
  else
    glOrtho(-2*(float)w/h, 2*(float)w/h, -2, 2, -2, 2);          //if width>height, then adjust cube vertically
  glMatrixMode(GL_MODELVIEW);                 //Set matrix mode to modelview matrix
}

int main(int argc,char *argv[]) {
  glutInit(&argc,argv);
  init();
  glutDisplayFunc(display);
  glutIdleFunc(spinCube);             //Register idle function i.e spinning cube without user interaction
  glutMouseFunc(mouse);               //Register mouse callback to listen to mouse clicks
  glutReshapeFunc(reshape);           //Register reshape function
  glutMainLoop();
  return 0;
}
```

You can also write reshape as follows but it doesn't support window reshape:

```
void reshape(int w, int h) {
  glutReshapeWindow(500,500); glMatrixMode(GL_PROJECTION);
  glLoadIdentity(); glOrtho(-2,2,-2,2,-2,2); glMatrixMode(GL_MODELVIEW);
} //If you write this, you can't able to change window size. Execute & Check
```

Output:



***