# Animation

```
#include <GL/gl.h>
#include <GL/glut.h>
#include <math.h>
// global variable diclaration
int frameNumber = 0; // frame no
void drawWindmill() // Function to draw windmill
{

    int i;
    glColor3f(0.0, 0.0, 0.0); // red green blue
    glBegin(GL_POLYGON);
    glVertex2f(-0.05f, 0); // for drawing rectangular base part
    glVertex2f(-0.05f, 3);
    glVertex2f(0.05f, 3);
    glVertex2f(0.05f, 0);
    glEnd();
    glTranslatef(0, 3, 0); // x,y,z
    glColor3f(1.0, 0.0, 0.0); // red,green,blue (RED PLATES OF WINDMILL)
    glRotated(frameNumber * (180.0 / 45), 0, 0, 1); //(angle,x,y,z)
    for (i = 0; i < 4; i++) // LOOP TO DRAW FOUR PLATES
    {
        glRotated(90, 0, 0, 1); // 90,0,0,Z
        glBegin(GL_POLYGON);
        glVertex2f(0, 0); // FOR DRAWING TYIANGLULAR PLATE
        glVertex2f(1.0f, 0.2f);
        glVertex2f(1.0f, -0.2f);
        glEnd();
    }
}
void display() // DISPLAY FUNCTION
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity(); // TAKES IDENTITY MATRIX
    glPushMatrix(); // PUSH MATRIX
    glTranslated(2.2, 1.6, 0); // SET POSITION OF WINDMILL
    glScaled(0.4, 0.4, 1); // SCALLING WINDMILL WITH POINT (0.4,0.4,1)
    drawWindmill(); // FUNCTION CALL TO DRAW WINDMILL
    glPopMatrix(); // POP MATRIX
    glPushMatrix(); // PUSH MATRIX
    glTranslated(3.7, 0.8, 0); // SET POSITION OF WINDMILL
    glScaled(0.7, 0.7, 1); // SCALLING WINDMILL WITH POIN(0.7,0.7,1)
    drawWindmill(); // FUNCTION CALL TO DRAW WINDMILL
```

```
    glPopMatrix(); // POP MATRIX
    glutSwapBuffers(); // SWAP BUFFER
}

void doFrame(int v)
{
    frameNumber++; // INCREMENT FRAME NO
    glutPostRedisplay(); // POST REDISPLAY
    glutTimerFunc(10, doFrame, 0);
}

void init() // FUNCTION INITIALISATION
{
    glClearColor(1, 1, 1, 0);
    glMatrixMode(GL_PROJECTION); // MATRIX MODE  FOR PROJECTION
    glLoadIdentity(); // LOADS IDENTITY MATRIX
    glOrtho(0, 7, -1, 4, -1, 1); // MIN X,MAX X,MIN Y,MAX Y,MIN Z,MAX Z VALUE
    glMatrixMode(GL_MODELVIEW); // MATRIX MOD FOR MODEL VIEW
}

int main(int argc, char **argv) // MAIN FUNCTION
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(700, 500); // DEFINED WINDOW SIZE 700*500
    glutInitWindowPosition(100, 100); // DEFINED WINDOW POSITION 100,100
    glutCreateWindow("Windmill Animation"); // NAME OF WINDOW
    init(); // FIRSTLY CALL TO INTIALISE VALUE
    glutDisplayFunc(display); // DISPLAY
    glutTimerFunc(200, doFrame, 0); // TIMER FUNC
    glutMainLoop();
    return 0;
}
```

# Bangle

```
#include <iostream>
#include <GL/freeglut.h>
#include <GL/gl.h>
```

```
#include <math.h>
using namespace std;

int cx = 300;
int cy = 300;
int r = 70;
bool flag = 1;

void DisplayPoint(int x, int y)
{
   glColor3f(0.0, 0.0, 0.0);
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
   glFlush();
}

void initialize(void)
{
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(0, 600, 0, 600);
}

void octant(int xc, int yc, int x, int y)
{
   DisplayPoint(xc + x, yc + y);
   DisplayPoint(xc + y, yc + x);
   DisplayPoint(xc + y, yc - x);
   DisplayPoint(xc + x, yc - y);
   DisplayPoint(xc - x, yc - y);
   DisplayPoint(xc - y, yc - x);
   DisplayPoint(xc - y, yc + x);
   DisplayPoint(xc - x, yc + y);
}

void circleMP(int xc, int yc, int r)
{
   int p = 1 - r, x = 0, y = r;
   // loop til the x become y equal to radius (r,r)
   while (x < y)
   {
      octant(xc, yc, x, y);
      x++;
```

```
        if (p > 0)
        {
            // if p>0 decrement the y and 2(x-y)+1
            y--;
            p += 2 * (x - y) + 1;
        }
        else
        {
            // if p<=0 add 2x+1 to p
            p += 2 * x + 1;
        }
    }
}

// convert the rad to deg
double ang(int q)
{
    return (double)q * 3.142 / 180;
}

void drawcircles(int x, int y, int r)
{
    // circleMP(x,y,r);
    circleMP(x + 2 * r, y, 0.26 * r);
    circleMP(x - 2 * r, y, 0.26 * r);
    circleMP(x + 2 * r * cos(ang(90)), y + 2 * r * sin(ang(90)), 0.26 * r);
    circleMP(x + 2 * r * cos(ang(75)), y + 2 * r * sin(ang(75)), 0.26 * r);
    circleMP(x + 2 * r * cos(ang(60)), y + 2 * r * sin(ang(60)), 0.26 * r);
    circleMP(x + 2 * r * cos(ang(45)), y + 2 * r * sin(ang(45)), 0.26 * r);
    circleMP(x + 2 * r * cos(ang(30)), y + 2 * r * sin(ang(30)), 0.26 * r);
    circleMP(x + 2 * r * cos(ang(15)), y + 2 * r * sin(ang(15)), 0.26 * r);

    circleMP(x - 2 * r * cos(ang(75)), y + 2 * r * sin(ang(75)), 0.26 * r);
    circleMP(x - 2 * r * cos(ang(60)), y + 2 * r * sin(ang(60)), 0.26 * r);
    circleMP(x - 2 * r * cos(ang(30)), y + 2 * r * sin(ang(30)), 0.26 * r);
    circleMP(x - 2 * r * cos(ang(45)), y + 2 * r * sin(ang(45)), 0.26 * r);
    circleMP(x - 2 * r * cos(ang(15)), y + 2 * r * sin(ang(15)), 0.26 * r);

    circleMP(x - 2 * r * cos(ang(90)), y - 2 * r * sin(ang(90)), 0.26 * r);
    circleMP(x - 2 * r * cos(ang(75)), y - 2 * r * sin(ang(75)), 0.26 * r);
    circleMP(x - 2 * r * cos(ang(60)), y - 2 * r * sin(ang(60)), 0.26 * r);
    circleMP(x - 2 * r * cos(ang(45)), y - 2 * r * sin(ang(45)), 0.26 * r);
    circleMP(x - 2 * r * cos(ang(30)), y - 2 * r * sin(ang(30)), 0.26 * r);
    circleMP(x - 2 * r * cos(ang(15)), y - 2 * r * sin(ang(15)), 0.26 * r);
```

```
    circleMP(x + 2 * r * cos(ang(75)), y - 2 * r * sin(ang(75)), 0.26 * r);
    circleMP(x + 2 * r * cos(ang(60)), y - 2 * r * sin(ang(60)), 0.26 * r);
    circleMP(x + 2 * r * cos(ang(30)), y - 2 * r * sin(ang(30)), 0.26 * r);
    circleMP(x + 2 * r * cos(ang(45)), y - 2 * r * sin(ang(45)), 0.26 * r);
    circleMP(x + 2 * r * cos(ang(15)), y - 2 * r * sin(ang(15)), 0.26 * r);

    circleMP(x, y, 1.75 * r);
    circleMP(x, y, 1.6 * r);
}

void renderFunction(void)
{
    glColor3f(1, 1, 1);
    glPointSize(2);
    drawcircles(300, 300, 70);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 0);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Circle");
    initialize();
    glutDisplayFunc(renderFunction);
    glutMainLoop();
    return 0;
}
```

# Bresenham Monitor

```
#include <iostream>
#include <GL/glut.h>
using namespace std;

void init ()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);       // Set display-window color to white
```

```
    glMatrixMode(GL_PROJECTION);            // Set projection parameters
    gluOrtho2D(-360, 360, 270, -270);       // To define coordinate reference (LRBT) frame within
the display window
}

void displayPoint (float x, float y)
{
    glColor3f(0, 0, 0);
    glPointSize(2);
    glBegin(GL_POINTS);
        glVertex2f(x, y);
    glEnd();
}

void bresLine (float x1, float y1, float x2, float y2)
{
    float p, xinc, yinc, pinc1, pinc2;
    float dx = x2 - x1;
    float dy = y2 - y1;
    float x = x1;
    float y = y1;

    if (dx < 0)
        dx = -dx;
    if (dy < 0)
        dy = -dy;

    xinc = 1;
    if (x2 < x1)
        xinc = -1;
    yinc = 1;
    if (y2 < y1)
        yinc = -1;

    displayPoint(x, y);

    if (dx > dy)
    {
        p = 2 * dy - dx;
        pinc1 = 2 * (dy - dx);
        pinc2 = 2 * dy;

        for (int i = 0; i < (int)dx; i++)
        {
```

```
            if (p >= 0)
            {
                y = y + yinc;
                p = p + pinc1;
            }
            else
                p = p + pinc2;
            x = x + xinc;

            displayPoint(x, y);
        }
    }
    else    // abs(dx) <= abs (dy)
    {
        p = 2 * dx - dy;
        pinc1 = 2 * (dx - dy);
        pinc2 = 2 * dx;

        for (int i = 0; i < (int)dy; i++)
        {
            if (p >= 0)
            {
                x = x + xinc;
                p = p + pinc1;
            }
            else
                p = p + pinc2;
            y = y + yinc;

            displayPoint(x, y);
        }
    }

    glFlush();
}

void drawing ()
{
    glClear(GL_COLOR_BUFFER_BIT);           // Clear display window

    // Outer Rectangle
    bresLine(-220, -158, 220, -158);
    bresLine(220, -158, 220, 107);
    bresLine(220, 107, -220, 107);
```

```
    bresLine(-220, 107, -220, -158);

    // Inner Rectangle
    bresLine(-200, -138, 200, -138);
    bresLine(200, -138, 200, 87);
    bresLine(200, 87, -200, 87);
    bresLine(-200, 87, -200, -138);

    // Base Trapezium
    bresLine(-75, 107, 75, 107);
    bresLine(75, 107, 100, 157);
    bresLine(100, 157, -100, 157);
    bresLine(-100, 157, -75, 107);

    // Triangle-1
    bresLine(-200, 87, -120, 0);
    bresLine(-120, 0, -30, 87);
    bresLine(-30, 87, -200, 87);

    // Triangle-2
    bresLine(-30, 87, 60, -60);
    bresLine(60, -60, 200, 87);
    bresLine(200, 87, -30, 87);

    glFlush();                              // Process all OpenGL routines as quickly as possible (To
put changed pixel in frame buffer)
}

int main (int argc, char **argv)                // argc -> argument counter, argv -> argument vector
{
    glutInit(&argc, argv);                      // Initialize GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RED);    // Set display mode
    glutInitWindowPosition(50, 100);            // Set top-left display-window position
    glutInitWindowSize(720, 540);               // Set display-window width and height
    glutCreateWindow("CGL QB Q5");              // Create display window

    init();                                 // Execute initialization procedure
    glutDisplayFunc(drawing);                   // Send graphics to display window
    glutMainLoop();

    return 0;
}
```

# Bresenham staircase

```
#include <iostream>
#include <GL/glut.h>
using namespace std;

void init ()
{
   glClearColor(1.0, 1.0, 1.0, 0.0);       // Set display-window color to white
   glMatrixMode(GL_PROJECTION);            // Set projection parameters
   gluOrtho2D(-360, 360, 270, -270);       // To define coordinate reference (LRBT) frame within
the display window
}

void displayPoint (float x, float y)
{
   glColor3f(0, 0, 0);
   glPointSize(2);
   glBegin(GL_POINTS);
      glVertex2f(x, y);
   glEnd();
}

void bresLine (float x1, float y1, float x2, float y2)
{
   float p, xinc, yinc, pinc1, pinc2;
   float dx = x2 - x1;
   float dy = y2 - y1;
   float x = x1;
   float y = y1;

   if (dx < 0)
      dx = -dx;
   if (dy < 0)
      dy = -dy;
```

```
xinc = 1;
if (x2 < x1)
   xinc = -1;
yinc = 1;
if (y2 < y1)
   yinc = -1;

displayPoint(x, y);

if (dx > dy)
{
   p = 2 * dy - dx;
   pinc1 = 2 * (dy - dx);
   pinc2 = 2 * dy;

   for (int i = 0; i < (int)dx; i++)
   {
      if (p >= 0)
      {
         y = y + yinc;
         p = p + pinc1;
      }
      else
         p = p + pinc2;
      x = x + xinc;

      displayPoint(x, y);
   }
}
else    // abs(dx) <= abs (dy)
{
   p = 2 * dx - dy;
   pinc1 = 2 * (dx - dy);
   pinc2 = 2 * dx;

   for (int i = 0; i < (int)dy; i++)
   {
      if (p >= 0)
      {
         x = x + xinc;
         p = p + pinc1;
      }
      else
         p = p + pinc2;
```

```
            y = y + yinc;

            displayPoint(x, y);
        }
    }

    glFlush();
}

void drawing ()
{
    glClear(GL_COLOR_BUFFER_BIT);          // Clear display window

    // Staircase Pattern
    bresLine(-150, 150, -150, 100);
    bresLine(-150, 100, -100, 100);
    bresLine(-100, 100, -100, 50);
    bresLine(-100, 50, -50, 50);
    bresLine(-50, 50, -50, 0);
    bresLine(-50, 0, 0, 0);
    bresLine(0, 0, 0, -50);
    bresLine(0, -50, 50, -50);
    bresLine(50, -50, 50, -100);
    bresLine(50, -100, 100, -100);
    bresLine(100, -100, 100, -150);
    bresLine(100, -150, 150, -150);
    bresLine(150, -150, 150, 150);
    bresLine(150, 150, -150, 150);

    glFlush();                             // Process all OpenGL routines as quickly as possible (To
put changed pixel in frame buffer)
}

int main (int argc, char **argv)                // argc -> argument counter, argv -> argument vector
{
    glutInit(&argc, argv);                      // Initialize GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RED);    // Set display mode
    glutInitWindowPosition(50, 100);            // Set top-left display-window position
    glutInitWindowSize(720, 540);               // Set display-window width and height
    glutCreateWindow("CGL QB Q21");             // Create display window

    init();                                     // Execute initialization procedure
    glutDisplayFunc(drawing);                   // Send graphics to display window
    glutMainLoop();
```

```
    return 0;
}
```

# dashLine bresenham pattern

```
#include <iostream>
#include <GL/glut.h>
using namespace std;

void init ()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);       // Set display-window color to white
    glMatrixMode(GL_PROJECTION);            // Set projection parameters
    gluOrtho2D(-360, 360, -270, 270);       // To define coordinate reference (LRBT) frame within the display
window
}

void displayPoint (float x, float y)
{
    glColor3f(0, 0, 0);
    glPointSize(2);
    glBegin(GL_POINTS);
        glVertex2f(x, y);
    glEnd();
}

struct Point
{
    float x;
    float y;
};
Point P, Q, A1, A2, B1, B2, B3, B4, C1, C2, C3, C4;

void pointCalculations ()
{
    A1.x = P.x;
    A1.y = Q.y;
    A2.x = Q.x;
    A2.y = P.y;

    B1.x = (A1.x + P.x) / 2;
    B1.y = (A1.y + P.y) / 2;
```

```
    B2.x = (A1.x + Q.x) / 2;
    B2.y = (A1.y + Q.y) / 2;
    B3.x = (Q.x + A2.x) / 2;
    B3.y = (Q.y + A2.y) / 2;
    B4.x = (P.x + A2.x) / 2;
    B4.y = (P.y + A2.y) / 2;

    C1.x = (B1.x + B2.x) / 2;
    C1.y = (B1.y + B2.y) / 2;
    C2.x = (B2.x + B3.x) / 2;
    C2.y = (B2.y + B3.y) / 2;
    C3.x = (B3.x + B4.x) / 2;
    C3.y = (B3.y + B4.y) / 2;
    C4.x = (B4.x + B1.x) / 2;
    C4.y = (B4.y + B1.y) / 2;
}

void bresDashLine (float x1, float y1, float x2, float y2)
{
    float p, xinc, yinc, pinc1, pinc2;
    float dx = x2 - x1;
    float dy = y2 - y1;
    float x = x1;
    float y = y1;

    if (dx < 0)
        dx = -dx;
    if (dy < 0)
        dy = -dy;

    xinc = 1;
    if (x2 < x1)
        xinc = -1;
    yinc = 1;
    if (y2 < y1)
        yinc = -1;

    displayPoint(x, y);

    if (dx > dy)
    {
        p = 2 * dy - dx;
        pinc1 = 2 * (dy - dx);
        pinc2 = 2 * dy;

        for (int i = 0; i < (int)dx; i++)
        {
            if (p >= 0)
            {
```

```
            y = y + yinc;
            p = p + pinc1;
        }
        else
            p = p + pinc2;
        x = x + xinc;

        if (i%11 == 3 || i%11 == 4 || i%11 == 5 || i%11 == 6 || i%11 == 7 || i%11 == 8 || i%11 == 9 || i%11 ==
10)
        displayPoint(x, y);
      }
   }
   else    // (dx) <= (dy)
   {
      p = 2 * dx - dy;
      pinc1 = 2 * (dx - dy);
      pinc2 = 2 * dx;

      for (int i = 0; i < (int)dy; i++)
      {
         if (p >= 0)
         {
            x = x + xinc;
            p = p + pinc1;
         }
         else
            p = p + pinc2;
         y = y + yinc;

         if (i%11 == 3 || i%11 == 4 || i%11 == 5 || i%11 == 6 || i%11 == 7 || i%11 == 8 || i%11 == 9 || i%11 ==
10)
         displayPoint(x, y);
      }
   }

   glFlush();
}

void drawing ()
{
   glClear(GL_COLOR_BUFFER_BIT);          // Clear display window

   bresDashLine(A1.x, A1.y, Q.x, Q.y);
   bresDashLine(Q.x, Q.y, A2.x, A2.y);
   bresDashLine(A2.x, A2.y, P.x, P.y);
   bresDashLine(P.x, P.y, A1.x, A1.y);

   bresDashLine(B1.x, B1.y, B2.x, B2.y);
   bresDashLine(B2.x, B2.y, B3.x, B3.y);
```

```
    bresDashLine(B3.x, B3.y, B4.x, B4.y);
    bresDashLine(B4.x, B4.y, B1.x, B1.y);

    bresDashLine(C1.x, C1.y, C2.x, C2.y);
    bresDashLine(C2.x, C2.y, C3.x, C3.y);
    bresDashLine(C3.x, C3.y, C4.x, C4.y);
    bresDashLine(C4.x, C4.y, C1.x, C1.y);

    glFlush();                              // Process all OpenGL routines as quickly as possible (To put
changed pixel in frame buffer)
}

int main (int argc, char **argv)                    // argc -> argument counter, argv -> argument vector
{
    // Coordinate inputs
    cout << "\n[Note that the x values range from -360 to 360 and y values from -270 to 270]" << endl;
    cout << "\nEnter the float coordinates (x, y) for the point P (bottom-left corner): ";
    cin >> P.x >> P.y;
    cout << "Enter the float coordinates (x, y) for the point Q (top-right corner): ";
    cin >> Q.x >> Q.y;
    pointCalculations();

    glutInit(&argc, argv);                          // Initialize GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RED);    // Set display mode
    glutInitWindowPosition(50, 100);                // Set top-left display-window position
    glutInitWindowSize(720, 540);                   // Set display-window width and height
    glutCreateWindow("CGL QB Q16");                 // Create display window

    init();                                 // Execute initialization procedure
    glutDisplayFunc(drawing);                       // Send graphics to display window
    glutMainLoop();

    return 0;
}
```

# DDA monitor mountain

```
#include <iostream>
#include <GL/glut.h>
using namespace std;
```

```
void init ()
{
   glClearColor(1.0, 1.0, 1.0, 0.0);       // Set display-window color to white
   glMatrixMode(GL_PROJECTION);            // Set projection parameters
   gluOrtho2D(-360, 360, 270, -270);       // To define coordinate reference (LRBT) frame within
the display window
}

void displayPoint (float x, float y)
{
   glColor3f(0, 0, 0);
   glPointSize(2);
   glBegin(GL_POINTS);
      glVertex2f(x, y);
   glEnd();
}

void ddaLine (float x1, float y1, float x2, float y2)
{
   float step;

   float dx = x2 - x1;
   float dy = y2 - y1;

   if (abs(dx) > abs(dy))
      step = abs(dx);
   else
      step = abs(dy);

   float xinc = dx / (float)step;
   float yinc = dy / (float)step;
   float x = x1;
   float y = y1;

   for (int i = 0; i <= step; i++)
   {
      displayPoint(x, y);
      x = x + xinc;
      y = y + yinc;
   }
}

void drawing ()
{
```

```
    glClear(GL_COLOR_BUFFER_BIT);          // Clear display window

    // Outer Rectangle
    ddaLine(-220, -158, 220, -158);
    ddaLine(220, -158, 220, 107);
    ddaLine(220, 107, -220, 107);
    ddaLine(-220, 107, -220, -158);

    // Inner Rectangle
    ddaLine(-200, -138, 200, -138);
    ddaLine(200, -138, 200, 87);
    ddaLine(200, 87, -200, 87);
    ddaLine(-200, 87, -200, -138);

    // Base Trapezium
    ddaLine(-75, 107, 75, 107);
    ddaLine(75, 107, 100, 157);
    ddaLine(100, 157, -100, 157);
    ddaLine(-100, 157, -75, 107);

    // Triangle-1
    ddaLine(-200, 87, -120, 0);
    ddaLine(-120, 0, -30, 87);
    ddaLine(-30, 87, -200, 87);

    // Triangle-2
    ddaLine(-30, 87, 60, -60);
    ddaLine(60, -60, 200, 87);
    ddaLine(200, 87, -30, 87);

    glFlush();                              // Process all OpenGL routines as quickly as possible (To
put changed pixel in frame buffer)
}

int main (int argc, char **argv)                // argc -> argument counter, argv -> argument vector
{
    glutInit(&argc, argv);                      // Initialize GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RED);    // Set display mode
    glutInitWindowPosition(50, 100);            // Set top-left display-window position
    glutInitWindowSize(720, 540);               // Set display-window width and height
    glutCreateWindow("CGL QB Q1");              // Create display window

    init();                                 // Execute initialization procedure
    glutDisplayFunc(drawing);                   // Send graphics to display window
```

```
    glutMainLoop();

    return 0;
}
```

# DDA line

```
#include <iostream>
#include <GL/glut.h>
using namespace std;

void init ()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);       // Set display-window color to white
    glMatrixMode(GL_PROJECTION);            // Set projection parameters
    gluOrtho2D(-360, 360, -270, 270);       // To define coordinate reference (LRBT) frame within
the display window
}

void displayPoint (float x, float y)
{
    glColor3f(0, 0, 0);
    glPointSize(2);
    glBegin(GL_POINTS);
        glVertex2f(x, y);
    glEnd();
}

void ddaLine (float x1, float y1, float x2, float y2)
{
    float step;

    float dx = x2 - x1;
    float dy = y2 - y1;

    if (abs(dx) > abs(dy))
        step = abs(dx);
```

```
    else
        step = abs(dy);

    float xinc = dx / (float)step;
    float yinc = dy / (float)step;
    float x = x1;
    float y = y1;

    for (int i = 0; i <= step; i++)
    {
        displayPoint(x, y);
        x = x + xinc;
        y = y + yinc;
    }
}

void ddaDashDotLine (float x1, float y1, float x2, float y2)
{
    float step;

    float dx = x2 - x1;
    float dy = y2 - y1;

    if (abs(dx) > abs(dy))
        step = abs(dx);
    else
        step = abs(dy);

    float xinc = dx / (float)step;
    float yinc = dy / (float)step;
    float x = x1;
    float y = y1;

    for (int i = 0; i <= step; i++)
    {
        // In a group of 15 pixels, 8 will form a line (dash), 1 will form a dot,
        // remaining 6 are off and hence provide space between dash and dot.
        // Here, i%15 == 1, 2, 3, 12, 13, 14 are off pixels adjecent to i%15 == 0 pixel.
        if (i%15 == 4 || i%15 == 5 || i%15 == 6 || i%15 == 7 || i%15 == 8 || i%15 == 9 || i%15 == 10
|| i%15 == 11)
            displayPoint(x, y);
        else if (i%15 == 0)
            displayPoint(x, y);
        x = x + xinc;
```

```
        y = y + yinc;
    }
}

void ddaDashLine (float x1, float y1, float x2, float y2)
{
    float step;

    float dx = x2 - x1;
    float dy = y2 - y1;

    if (abs(dx) > abs(dy))
        step = abs(dx);
    else
        step = abs(dy);

    float xinc = dx / (float)step;
    float yinc = dy / (float)step;
    float x = x1;
    float y = y1;

    for (int i = 0; i <= step; i++)
    {
        // Here, we only need line (dash) and spaces.
        // Therefore in a group of 11 pixels, i%11 == 0, 1, 2 are three off pixels
        // and the remaining form a straigh line (dash).
        if (i%11 == 3 || i%11 == 4 || i%11 == 5 || i%11 == 6 || i%11 == 7 || i%11 == 8 || i%11 == 9 ||
i%11 == 10)
            displayPoint(x, y);
        x = x + xinc;
        y = y + yinc;
    }
}

float x01, y01, x02, y02;
int choice;
void drawing ()
{
    glClear(GL_COLOR_BUFFER_BIT);           // Clear display window

    // Dividing the screen in four quadrants by drawning
    // two intersecting axes
    glColor3f(0.0, 1.0, 0.0);               // Coloring the axes Green
    glBegin(GL_LINES);
```

```
        glVertex2f(-360, 0);
        glVertex2f(360, 0);
    glEnd();
    glBegin(GL_LINES);
        glVertex2f(0, 270);
        glVertex2f(0, -270);
    glEnd();

    switch (choice)
    {
        case 1:
            ddaDashDotLine(x01, y01, x02, y02);
            break;
        case 2:
            ddaDashLine(x01, y01, x02, y02);
            break;
        default:
            break;
    }

    glFlush();                              // Process all OpenGL routines as quickly as possible (To
put changed pixel in frame buffer)
}

int main (int argc, char **argv)                    // argc -> argument counter, argv -> argument vector
{
    // Choosing Dash-Dot-Dash or Dash Line
    cout << "Dash-Dot-Dash / Dash Line using DDA" << endl;
    cout << "Enter 1 for Dash-Dot-Dash Line";
    cout << "\nEnter 2 for Dash Line" << endl;
    cin >> choice;

    // Coordinate inputs
    cout << "\nEnter the float coordinates of the points P1 and P2" << endl;
    cout << "Note that the x values range from -360 to 360 and y values from -270 to 270" <<
endl;
    cout << "(x, y) for P1: ";
    cin >> x01 >> y01;
    cout << "(x, y) for P2: ";
    cin >> x02 >> y02;

    glutInit(&argc, argv);                          // Initialize GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RED);    // Set display mode
    glutInitWindowPosition(50, 100);                // Set top-left display-window position
```

```
    glutInitWindowSize(720, 540);                // Set display-window width and height
    glutCreateWindow("CGL QB Q3");               // Create display window

    init();                                      // Execute initialization procedure
    glutDisplayFunc(drawing);                    // Send graphics to display window
    glutMainLoop();

    return 0;
}
```

# DDA staircase

```
#include <iostream>
#include <GL/glut.h>
using namespace std;

void init ()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);       // Set display-window color to white
    glMatrixMode(GL_PROJECTION);            // Set projection parameters
    gluOrtho2D(-360, 360, 270, -270);       // To define coordinate reference (LRBT) frame within
the display window
}

void displayPoint (float x, float y)
{
    glColor3f(0, 0, 0);
    glPointSize(2);
    glBegin(GL_POINTS);
        glVertex2f(x, y);
    glEnd();
}

void ddaLine (float x1, float y1, float x2, float y2)
{
    float step;

    float dx = x2 - x1;
    float dy = y2 - y1;
```

```
    if (abs(dx) > abs(dy))
        step = abs(dx);
    else
        step = abs(dy);

    float xinc = dx / (float)step;
    float yinc = dy / (float)step;
    float x = x1;
    float y = y1;

    for (int i = 0; i <= step; i++)
    {
        displayPoint(x, y);
        x = x + xinc;
        y = y + yinc;
    }
}

void drawing ()
{
    glClear(GL_COLOR_BUFFER_BIT);          // Clear display window

    // Staircase Pattern
    ddaLine(-150, 150, -150, 100);
    ddaLine(-150, 100, -100, 100);
    ddaLine(-100, 100, -100, 50);
    ddaLine(-100, 50, -50, 50);
    ddaLine(-50, 50, -50, 0);
    ddaLine(-50, 0, 0, 0);
    ddaLine(0, 0, 0, -50);
    ddaLine(0, -50, 50, -50);
    ddaLine(50, -50, 50, -100);
    ddaLine(50, -100, 100, -100);
    ddaLine(100, -100, 100, -150);
    ddaLine(100, -150, 150, -150);
    ddaLine(150, -150, 150, 150);
    ddaLine(150, 150, -150, 150);

    glFlush();                             // Process all OpenGL routines as quickly as possible (To
put changed pixel in frame buffer)
}

int main (int argc, char **argv)                  // argc -> argument counter, argv -> argument vector
{
```

```
    glutInit(&argc, argv);                          // Initialize GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RED);    // Set display mode
    glutInitWindowPosition(50, 100);                // Set top-left display-window position
    glutInitWindowSize(720, 540);                   // Set display-window width and height
    glutCreateWindow("CGL QB Q19");                 // Create display window

    init();                                 // Execute initialization procedure
    glutDisplayFunc(drawing);                       // Send graphics to display window
    glutMainLoop();

    return 0;
}
```

# Koch curve

```
#include <iostream>
#include <GL/glut.h>
#include <math.h>
using namespace std;

void init ()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.0, 0.0, 0.0);
    gluOrtho2D(-360, 360, -270, 270);
    glClear(GL_COLOR_BUFFER_BIT);
};

// Converting the degree angle to radian
float ang (float d)
{
    return (float)d * 3.142/180;
}

void draw();

void draw_koch(float xa, float ya, float xb, float yb, int n)
{
    float xc, xd, yc, yd, midx, midy;
```

```
    // C and D points trisect the A-B line
    xc = (2 * xa + xb) / 3;
    yc = (2 * ya + yb) / 3;
    xd = (2 * xb + xa) / 3;
    yd = (2 * yb + ya) / 3;

    // (Mid) point is made by the curve continuation of C and D
    midx = xc + ((xd - xc) * cos(ang(60))) + ((yd - yc) * sin(ang(60)));
    midy = yc - ((xd - xc) * sin(ang(60))) + ((yd - yc) * cos(ang(60)));

    if (n > 0)
    {
        draw_koch(xa, ya, xc, yc, n - 1);
        draw_koch(xc, yc, midx, midy, n - 1);
        draw_koch(midx, midy, xd, yd, n - 1);
        draw_koch(xd, yd, xb, yb, n - 1);
    }

    else
    {
        glVertex2f(xa, ya);
        glVertex2f(xc, yc);

        glVertex2f(xc, yc);
        glVertex2f(midx, midy);

        glVertex2f(midx, midy);
        glVertex2f(xd, yd);

        glVertex2f(xd, yd);
        glVertex2f(xb, yb);
    }
}

void draw (int n)
{
    glBegin(GL_LINES);
    draw_koch(0, -200, 173.40, 100, n);
    draw_koch(173.40, 100, -173.40, 100, n);
    draw_koch(-173.40, 100, 0, -200, n);
    glEnd();
    glFlush();
}
```

```
int main(int argc, char **argv)
{
   cout << "\n[Koch Curve]" << endl;
   int n;
   cout << "Enter the number of iterations (n): ";
   cin >> n;
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition(50, 100);
   glutInitWindowSize(720, 540);
   glutCreateWindow("Koch Curve");
   init();
   draw(n);
   glutMainLoop();
   return 0;
}
```

# Many circle

```
#include <iostream>
#include <math.h>
#include <GL/glut.h>
using namespace std;

void init ()
{
   glClearColor(1.0, 1.0, 1.0, 0.0);       // Set display-window color to white
   glMatrixMode(GL_PROJECTION);            // Set projection parameters
   gluOrtho2D(-360, 360, -270, 270);       // To define coordinate reference (LRBT) frame within
the display window
}

void displayPoint (float x, float y)
{
   glColor3f(0, 0, 0);
   glPointSize(2);
```

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

void drawCircle(float xc, float yc, float x, float y)
{
    displayPoint(xc + x, yc + y);
    displayPoint(xc + x, yc - y);
    displayPoint(xc - x, yc + y);
    displayPoint(xc - x, yc - y);
    displayPoint(xc + y, yc + x);
    displayPoint(xc + y, yc - x);
    displayPoint(xc - y, yc + x);
    displayPoint(xc - y, yc - x);
}

void bresCircle (float xc, float yc, float r)
{
    float x = 0, y = r;
    float d = 3 - 2*r;

    drawCircle(xc, yc, x, y);
    while (y >= x)
    {
        x++;
        if (d >= 0)
        {
            y--;
            d = d + 4*(x-y) + 10;
        }
        else
            d = d + 4*x + 6;
        drawCircle(xc, yc, x, y);
    }

    glFlush();
}

// Converting the degree angle to radian
float ang(float d)
{
    return (float)d*3.142/180;
}
```

```
float xc0, yc0, r0;
void drawing ()
{
   glClear(GL_COLOR_BUFFER_BIT);          // Clear display window

   bresCircle(xc0, yc0, r0);

   bresCircle(xc0 + r0*sin(ang(0)), yc0 + r0*cos(ang(0)), r0);
   bresCircle(xc0 + r0*sin(ang(45)), yc0 + r0*cos(ang(45)), r0);
   bresCircle(xc0 + r0*sin(ang(90)), yc0 + r0*cos(ang(90)), r0);
   bresCircle(xc0 + r0*sin(ang(135)), yc0 + r0*cos(ang(135)), r0);
   bresCircle(xc0 + r0*sin(ang(180)), yc0 + r0*cos(ang(180)), r0);
   bresCircle(xc0 + r0*sin(ang(225)), yc0 + r0*cos(ang(225)), r0);
   bresCircle(xc0 + r0*sin(ang(270)), yc0 + r0*cos(ang(270)), r0);
   bresCircle(xc0 + r0*sin(ang(315)), yc0 + r0*cos(ang(315)), r0);

   bresCircle(xc0, yc0, 2*r0);

   glFlush();                          // Process all OpenGL routines as quickly as possible (To put
changed pixel in frame buffer)
}

int main (int argc, char **argv)                  // argc -> argument counter, argv -> argument vector
{
   cout << "\n[Note that the x values range from -360 to 360 and y values from -270 to 270]" <<
endl;
   cout << "\nEnter the centre (x, y) of the central circle: ";
   cin >> xc0 >> yc0;
   cout << "\nEnter the radius of the central circle: ";
   cin >> r0;

   glutInit(&argc, argv);                        // Initialize GLUT
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RED);    // Set display mode
   glutInitWindowPosition(50, 100);               // Set top-left display-window position
   glutInitWindowSize(720, 540);                  // Set display-window width and height
   glutCreateWindow("CGL QB Q28");                // Create display window

   init();                                  // Execute initialization procedure
   glutDisplayFunc(drawing);                     // Send graphics to display window
   glutMainLoop();

   return 0;
}
```

# Necklace Pattern

```
#include <iostream>
#include <math.h>
#include <GL/glut.h>
using namespace std;

void init ()
{
   glClearColor(1.0, 1.0, 1.0, 0.0);       // Set display-window color to white
   glMatrixMode(GL_PROJECTION);            // Set projection parameters
   gluOrtho2D(-360, 360, -270, 270);       // To define coordinate reference (LRBT) frame within
the display window
}

void displayPoint (float x, float y)
{
   glColor3f(0, 0, 0);
   glPointSize(1);
   glBegin(GL_POINTS);
      glVertex2f(x, y);
   glEnd();
}

void drawCircle(float xc, float yc, float x, float y)
{
   displayPoint(xc + x, yc + y);
   displayPoint(xc + x, yc - y);
   displayPoint(xc - x, yc + y);
   displayPoint(xc - x, yc - y);
   displayPoint(xc + y, yc + x);
   displayPoint(xc + y, yc - x);
   displayPoint(xc - y, yc + x);
   displayPoint(xc - y, yc - x);
}

void bresCircle (float xc, float yc, float r)
```

```
{
   float x = 0, y = r;
   float d = 3 - 2*r;

   drawCircle(xc, yc, x, y);
   while (y >= x)
   {
      x++;
      if (d >= 0)
      {
         y--;
         d = d + 4*(x-y) + 10;
      }
      else
         d = d + 4*x + 6;
      drawCircle(xc, yc, x, y);
   }

   glFlush();
}

// Converting the degree angle to radian
float ang(float d)
{
   return (float)d * 3.142/180;
}

float xc0, yc0;
void drawing ()
{
   glClear(GL_COLOR_BUFFER_BIT);          // Clear display window

   bresCircle(xc0, yc0, 40);

   bresCircle(xc0 + 64, yc0 + 14, 25);
   bresCircle(xc0 + 106, yc0 + 42, 25);
   bresCircle(xc0 + 141, yc0 + 78, 25);
   bresCircle(xc0 + 158, yc0 + 126, 25);
   bresCircle(xc0 + 160, yc0 + 177, 25);

   bresCircle(xc0 - 64, yc0 + 14, 25);
   bresCircle(xc0 - 106, yc0 + 42, 25);
   bresCircle(xc0 - 141, yc0 + 78, 25);
   bresCircle(xc0 - 158, yc0 + 126, 25);
```

```
    bresCircle(xc0 - 160, yc0 + 177, 25);

    bresCircle(xc0, yc0 - 66, 25);
    bresCircle(xc0 + 46, yc0 - 46, 25);
    bresCircle(xc0 - 46, yc0 - 46, 25);
    bresCircle(xc0, yc0 - 117, 25);

    glFlush();                          // Process all OpenGL routines as quickly as possible (To put
changed pixel in frame buffer)
}

int main (int argc, char **argv)                    // argc -> argument counter, argv -> argument vector
{
    cout << "\n[Note that the x values range from -360 to 360 and y values from -270 to 270]" <<
endl;
    cout << "\nEnter the centre (x, y) of the middle bead of the necklace: ";
    cin >> xc0 >> yc0;

    glutInit(&argc, argv);                          // Initialize GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RED);    // Set display mode
    glutInitWindowPosition(50, 100);                // Set top-left display-window position
    glutInitWindowSize(720, 540);                   // Set display-window width and height
    glutCreateWindow("CGL QB Q23");                 // Create display window

    init();                                     // Execute initialization procedure
    glutDisplayFunc(drawing);                       // Send graphics to display window
    glutMainLoop();

    return 0;
}
```