**Kathmandu University**

**Department of Computer Science and Engineering Dhulikhel, Kavre**



**Mini Report on**

**“Lab 3”**

**[Course Code: COMP 342]**

**(For partial fulfillment of III Year/ I Semester in Computer Science)**

# Submitted By

**Nisham Ghimire (17)**

# Submitted To

**Mr. Dhiraj Shrestha**

**Department of Computer Science and Engineering**

# Submission Date

**14th January, 2024**

1. Write a Program to implement mid- point Circle Drawing Algorithm. Ans:

**Algorithm**

1. Start
2. Input the coordinates of two-line endpoints.
3. Store the left endpoint in (x0, y0).
4. Plot First Point:
5. Load (x0, y0) into the frame buffer (plot the first point).
6. Calculate Constants:
   * dx (change in x)
   * dy (change in y)
   * 2dy (twice the change in y)
   * 2dy - 2dx
7. Obtain the initial value for the decision parameter:
   * P0 = 2dy - dx
8. Point Calculation Loop:
   * For each xk along the line starting at k=0, perform the following tests:
   * If Pk < 0, then the next point to plot is (xk + 1, yk), and:
   * Update Pk+1 = Pk + 2dy
   * Otherwise, if Pk >= 0, the next point to plot is (xk + 1, yk + 1), and:
   * Update Pk+1 = Pk + 2dy - 2dx
9. Repeat Loop:
   * Repeat the above steps ∆x times.

Stop

**Source Code:**

import ctypes

import numpy as np

from OpenGL.GL import \*

from OpenGL.GLU import \*

import pygame as pg

from pygame.locals import \*

*# Define Shaders*

vertexShader = """

attribute vec2 position;

void main()

{

  gl\_Position = vec4(position, 0.0, 1.0);

}

"""

fragmentShader = """

void main()

{

  gl\_FragColor = vec4(0.0,1.0,0.0,1.0);

}

"""

def normalize(xList, yList, resolution):

    xList = [x / resolution for x in xList]

    yList = [y / resolution for y in yList]

    coordinateList = np.zeros((len(xList), 2))

    i = 0

    for \_ in xList:

        coordinateList[i] = [xList[i], yList[i]]

        i += 1

    return coordinateList

def midpoint\_circle(x\_center, y\_center, r, res):

    x = r

    y = 0

*# Printing the initial point the*

*# axes after translation*

    x\_coordinates = np.array([])

    y\_coordinates = np.array([])

    x\_coordinates = np.append(x\_coordinates, x + x\_center)

    y\_coordinates = np.append(y\_coordinates, y + y\_center)

*# When radius is zero only a single*

*# point be printed*

    if r > 0:

        x\_coordinates = np.append(x\_coordinates, x + x\_center)

        x\_coordinates = np.append(x\_coordinates, y + x\_center)

        x\_coordinates = np.append(x\_coordinates, -y + x\_center)

        y\_coordinates = np.append(y\_coordinates, -y + y\_center)

        y\_coordinates = np.append(y\_coordinates, x + y\_center)

        y\_coordinates = np.append(y\_coordinates, x + y\_center)

    P = 1 - r

    while x > y:

        y += 1

        if P <= 0:

            P = P + 2 \* y + 1

        else:

            x -= 1

            P = P + 2 \* y - 2 \* x + 1

        if x < y:

            break

        x\_coordinates = np.append(x\_coordinates, x + x\_center)

        x\_coordinates = np.append(x\_coordinates, -x + x\_center)

        x\_coordinates = np.append(x\_coordinates, x + x\_center)

        x\_coordinates = np.append(x\_coordinates, -x + x\_center)

        y\_coordinates = np.append(y\_coordinates, y + y\_center)

        y\_coordinates = np.append(y\_coordinates, y + y\_center)

        y\_coordinates = np.append(y\_coordinates, -y + y\_center)

        y\_coordinates = np.append(y\_coordinates, -y + y\_center)

        if x != y:

            x\_coordinates = np.append(

                x\_coordinates,

                y + x\_center,

            )

            x\_coordinates = np.append(x\_coordinates, -y + x\_center)

            x\_coordinates = np.append(x\_coordinates, y + x\_center)

            x\_coordinates = np.append(x\_coordinates, -y + x\_center)

            y\_coordinates = np.append(y\_coordinates, x + y\_center)

            y\_coordinates = np.append(y\_coordinates, x + y\_center)

            y\_coordinates = np.append(y\_coordinates, -x + y\_center)

            y\_coordinates = np.append(y\_coordinates, -x + y\_center)

    return normalize(x\_coordinates, y\_coordinates, res)

tempData = midpoint\_circle(0, 0, 500, 1000)

data = np.zeros(int(len(tempData)), [("position", np.float32, 2)])

data["position"] = tempData

def compileShader(source, type):

    shader = glCreateShader(type)

    glShaderSource(shader, source)

    glCompileShader(shader)

    if not glGetShaderiv(shader, GL\_COMPILE\_STATUS):

        error = glGetShaderInfoLog(shader).decode()

        print(error)

        raise RuntimeError(f"{source} shader compilation error")

    return shader

def createProgram(vertex, fragment):

    program = glCreateProgram()

    glAttachShader(program, vertex)

    glAttachShader(program, fragment)

    glLinkProgram(program)

    if not glGetProgramiv(program, GL\_LINK\_STATUS):

        print(glGetProgramInfoLog(program))

        raise RuntimeError("Error Linking program")

    glDetachShader(program, vertex)

    glDetachShader(program, fragment)

    return program

def main():

    running = True

    while running:

        width, height = 800, 800

        pg.init()

        pg.display.set\_mode((width, height), DOUBLEBUF | OPENGL | GL\_RGBA)

        pg.display.set\_caption("Midpoint Circle - Lab 3 | Nisham Ghimire")

        glViewport(0, 0, width, height)

*# here inti()*

        glClear(GL\_COLOR\_BUFFER\_BIT)

        glClearColor(0.0, 0.0, 0.0, 1.0)

        glLoadIdentity()

        program = createProgram(

            compileShader(vertexShader, GL\_VERTEX\_SHADER),

            compileShader(fragmentShader, GL\_FRAGMENT\_SHADER),

        )

        glUseProgram(program)

        buffer = glGenBuffers(1)

        glBindBuffer(GL\_ARRAY\_BUFFER, buffer)

        stride = data.strides[0]

        offset = ctypes.c\_void\_p(0)

        loc = glGetAttribLocation(program, "position")

        glEnableVertexAttribArray(loc)

        glBindBuffer(GL\_ARRAY\_BUFFER, buffer)

        glVertexAttribPointer(loc, 3, GL\_FLOAT, False, stride, offset)

        glBufferData(GL\_ARRAY\_BUFFER, data.nbytes, data, GL\_STATIC\_DRAW)

        glDrawArrays(GL\_POINTS, 0, len(data))

        pg.display.flip()

        for event in pg.event.get():

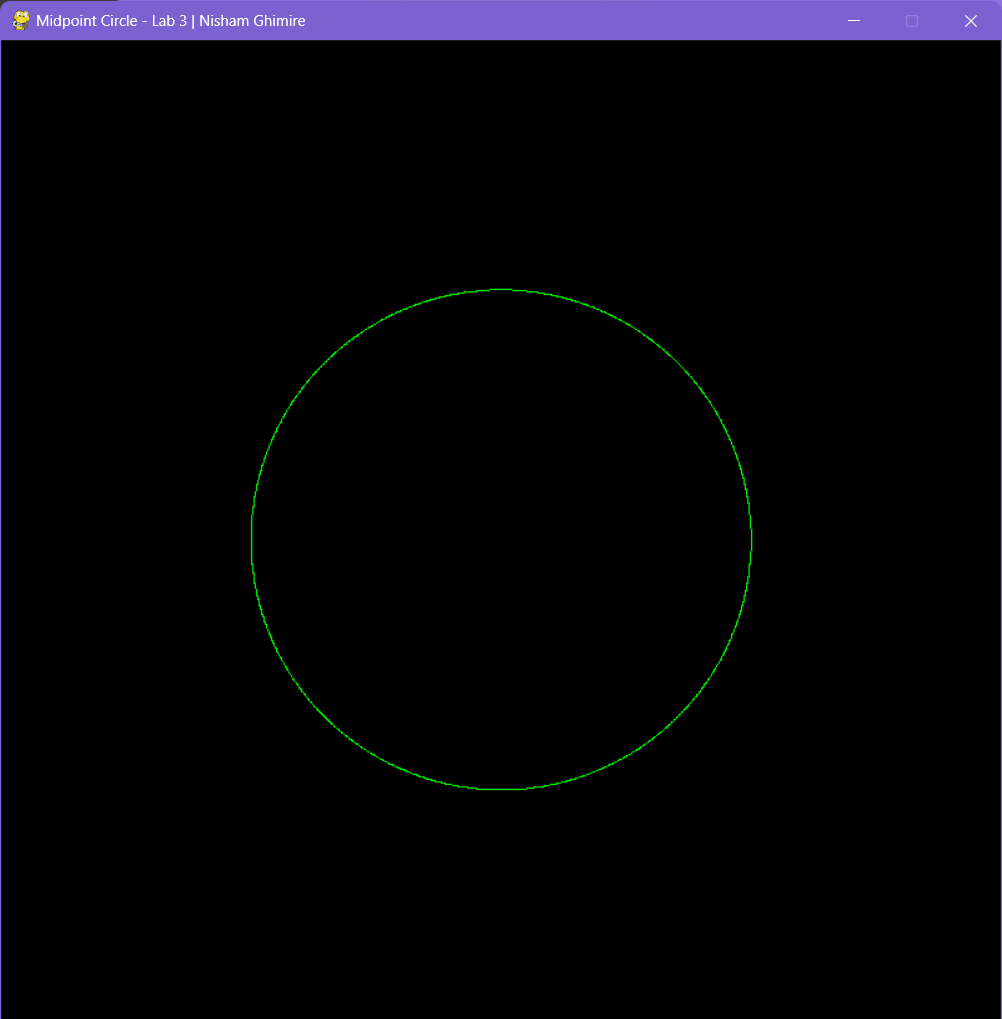
            if event.type == pg.QUIT:

                running = False

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output:



1. Write a Program to implement mid- point Ellipse Drawing Algorithm Ans:

**Algorithm**

1. Take input radius along x axis and y axis and obtain center of ellipse.
2. Initially, we assume ellipse to be centered at origin and the first point as : (x, y0)= (0, ry). 3.Obtain the initial decision parameter for region 1 as: p10=ry2+1/4rx2-rx 2ry
3. For every xk position in region 1 :
   * If p1k<0 then the next point along the is (xk+1 , yk) and p1k+1=p1k+2ry2xk+1+ry2
   * Else, the next point is (xk+1, yk-1 ) 5.And p1k+1=p1k+2ry2xk+1 – 2rx2yk+1+ry2
4. Obtain the initial value in region 2 using the last point (x0, y0) of region 1 as: p20=ry2(x0+1/2)2+rx2 (y0-1)2-rx2ry2
5. At each yk in region 2 starting at k =0 perform the following task.
   * If p2k>0 the next point is (xk, yk-1) and p2k+1=p2k-2rx2yk+1+rx2
   * Else, the next point is (xk+1, yk -1) and p2k+1=p2k+2ry2xk+1 -2rx2yk+1+rx2
6. Now obtain the symmetric points in the three quadrants and plot the coordinate value as: x=x+xc, y=y+yc
7. Repeat the steps for region 1 until 2ry2x>=2rx2y

**Source Code:**

import ctypes

import numpy as np

from OpenGL.GL import \*

from OpenGL.GLU import \*

import pygame as pg

from pygame.locals import \*

*# Define Shaders*

vertexShader = """

attribute vec2 position;

void main()

{

  gl\_Position = vec4(position, 0.0, 1.0);

}

"""

fragmentShader = """

void main()

{

  gl\_FragColor = vec4(1.0,0.0,0.0,1.0);

}

"""

def normalize(xList, yList, resolution):

    xList = [x / resolution for x in xList]

    yList = [y / resolution for y in yList]

    coordinateList = np.zeros((len(xList), 2))

    i = 0

    for \_ in xList:

        coordinateList[i] = [xList[i], yList[i]]

        i += 1

    return coordinateList

def midpoint\_ellipse(rx, ry, xc, yc, res):

    x = 0

    y = ry

    x\_coordinates = np.array([])

    y\_coordinates = np.array([])

*# Initial decision parameter of region 1*

    d1 = (ry \* ry) - (rx \* rx \* ry) + (0.25 \* rx \* rx)

    dx = 2 \* ry \* ry \* x

    dy = 2 \* rx \* rx \* y

*# For region 1*

    while dx < dy:

        x\_coordinates = np.append(x\_coordinates, x + xc)

        x\_coordinates = np.append(x\_coordinates, -x + xc)

        x\_coordinates = np.append(x\_coordinates, x + xc)

        x\_coordinates = np.append(x\_coordinates, -x + xc)

        y\_coordinates = np.append(y\_coordinates, y + yc)

        y\_coordinates = np.append(y\_coordinates, y + yc)

        y\_coordinates = np.append(y\_coordinates, -y + yc)

        y\_coordinates = np.append(y\_coordinates, -y + yc)

        if d1 < 0:

            x += 1

            dx = dx + (2 \* ry \* ry)

            d1 = d1 + dx + (ry \* ry)

        else:

            x += 1

            y -= 1

            dx = dx + (2 \* ry \* ry)

            dy = dy - (2 \* rx \* rx)

            d1 = d1 + dx - dy + (ry \* ry)

*# Decision parameter of region 2*

    d2 = (

        ((ry \* ry) \* ((x + 0.5) \* (x + 0.5)))

        + ((rx \* rx) \* ((y - 1) \* (y - 1)))

        - (rx \* rx \* ry \* ry)

    )

*# Plotting points of region 2*

    while y >= 0:

        x\_coordinates = np.append(x\_coordinates, x + xc)

        x\_coordinates = np.append(x\_coordinates, -x + xc)

        x\_coordinates = np.append(x\_coordinates, x + xc)

        x\_coordinates = np.append(x\_coordinates, -x + xc)

        y\_coordinates = np.append(y\_coordinates, y + yc)

        y\_coordinates = np.append(y\_coordinates, y + yc)

        y\_coordinates = np.append(y\_coordinates, -y + yc)

        y\_coordinates = np.append(y\_coordinates, -y + yc)

        if d2 > 0:

            y -= 1

            dy = dy - (2 \* rx \* rx)

            d2 = d2 + (rx \* rx) - dy

        else:

            y -= 1

            x += 1

            dx = dx + (2 \* ry \* ry)

            dy = dy - (2 \* rx \* rx)

            d2 = d2 + dx - dy + (rx \* rx)

    return normalize(x\_coordinates, y\_coordinates, res)

tempData = midpoint\_ellipse(800, 400, 0, 0, 1000)

data = np.zeros(int(len(tempData)), [("position", np.float32, 2)])

data["position"] = tempData

def compileShader(source, type):

    shader = glCreateShader(type)

    glShaderSource(shader, source)

    glCompileShader(shader)

    if not glGetShaderiv(shader, GL\_COMPILE\_STATUS):

        error = glGetShaderInfoLog(shader).decode()

        print(error)

        raise RuntimeError(f"{source} shader compilation error")

    return shader

def createProgram(vertex, fragment):

    program = glCreateProgram()

    glAttachShader(program, vertex)

    glAttachShader(program, fragment)

    glLinkProgram(program)

    if not glGetProgramiv(program, GL\_LINK\_STATUS):

        print(glGetProgramInfoLog(program))

        raise RuntimeError("Error Linking program")

    glDetachShader(program, vertex)

    glDetachShader(program, fragment)

    return program

def main():

    running = True

    while running:

        width, height = 800, 800

        pg.init()

        pg.display.set\_mode((width, height), DOUBLEBUF | OPENGL | GL\_RGBA)

        pg.display.set\_caption("Midpoint Ellipse - Lab 3 | Nisham Ghimire")

        glViewport(0, 0, width, height)

*# here inti()*

        glClear(GL\_COLOR\_BUFFER\_BIT)

        glClearColor(0.0, 0.0, 0.0, 1.0)

        glLoadIdentity()

        program = createProgram(

            compileShader(vertexShader, GL\_VERTEX\_SHADER),

            compileShader(fragmentShader, GL\_FRAGMENT\_SHADER),

        )

        glUseProgram(program)

        buffer = glGenBuffers(1)

        glBindBuffer(GL\_ARRAY\_BUFFER, buffer)

        stride = data.strides[0]

        offset = ctypes.c\_void\_p(0)

        loc = glGetAttribLocation(program, "position")

        glEnableVertexAttribArray(loc)

        glBindBuffer(GL\_ARRAY\_BUFFER, buffer)

        glVertexAttribPointer(loc, 3, GL\_FLOAT, False, stride, offset)

        glBufferData(GL\_ARRAY\_BUFFER, data.nbytes, data, GL\_STATIC\_DRAW)

        glDrawArrays(GL\_POINTS, 0, len(data))

        pg.display.flip()

        for event in pg.event.get():

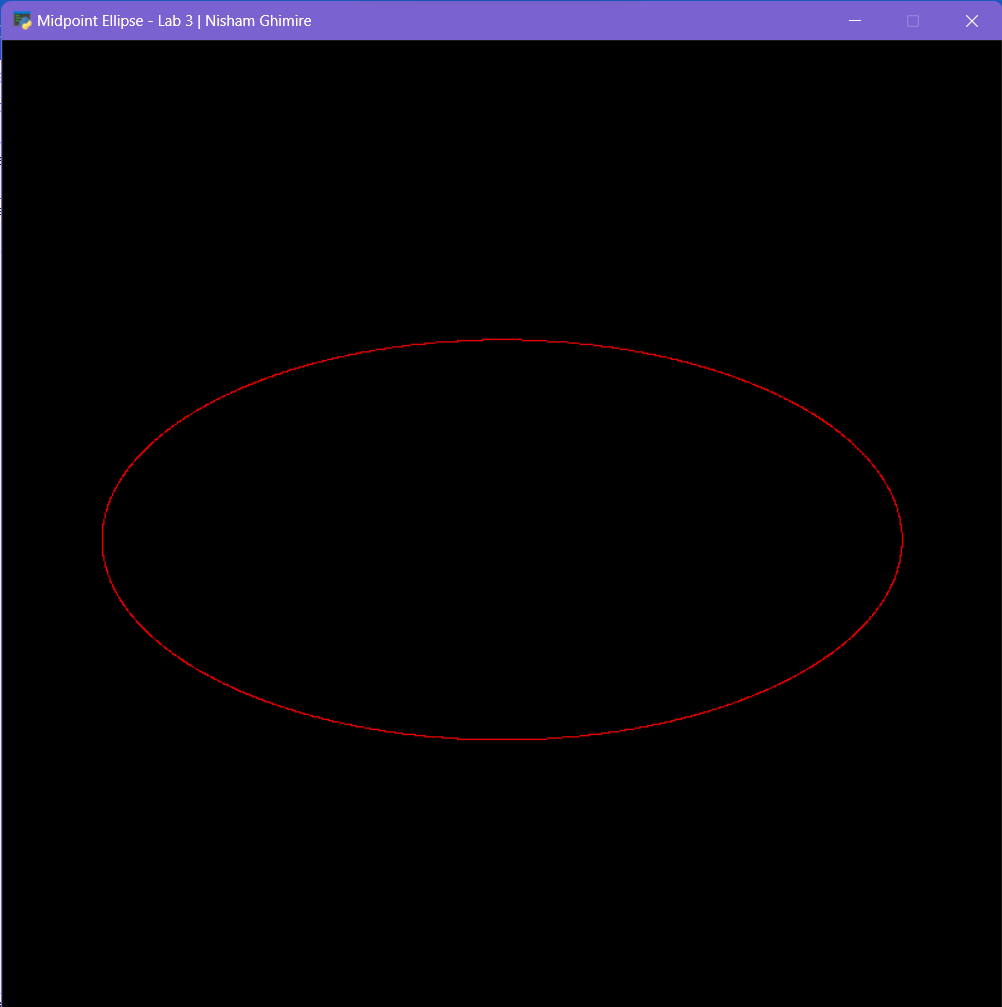
            if event.type == pg.QUIT:

                running = False

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output**:



**Conclusion:**

After the completion of this lab, I learned how to draw a circle using mid-point circle algorithm and ellipse using mid-point ellipse algorithm by the use of python, Opengl APIs for python, and pygame for window creation.