# Kathmandu University Department of Computer Science and Engineering

**Dhulikhel, Kavre**



# Mini Report on

**“Lab 5”**

# [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**

**LAB 6**

1. *Implement Cohen Sutherland Line Clipping algorithm.*

**Ans**

**Algorithm**

1) Assign the region codes to both endpoints.

2) Perform OR operation on both of these endpoints.

3) if OR = 0000,

then it is completely visible (inside the window).

* Else

Perform AND operation on both these endpoints.

* if AND ? 0000,

then the line is invisible and not inside the window. Also, it can’t be considered for clipping.

* else

AND = 0000, the line is partially inside the window and considered for clipping.

4) After confirming that the line is partially inside the window, then we find the intersection with the boundary of the window. By using the following formula:-

* Slope:- m= (y2-y1)/(x2-x1)

a) If the line passes through top or the line intersects with the top boundary of the window.

* x = x + (y\_wmax – y)/m
* y = y\_wmax

b) If the line passes through the bottom or the line intersects with the bottom boundary of the window.

* x = x + (y\_wmin – y)/m
* y = y\_wmin

c) If the line passes through the left region or the line intersects with the left boundary of the window.

* y = y+ (x\_wmin – x)\*m
* x = x\_wmin

d) If the line passes through the right region or the line intersects with the right boundary of the window.

* y = y + (x\_wmax -x)\*m
* x = x\_wmax

5) Now, overwrite the endpoints with a new one and update it.

6) Repeat the 4th step till your line doesn’t get completely clipped

**Source Code:**

import pygame

from pygame.locals import \*

from OpenGL.GL import \*

from OpenGL.GLU import \*

*# Define region codes*

INSIDE = 0  *# 0000*

LEFT = 1    *# 0001*

RIGHT = 2   *# 0010*

BOTTOM = 4  *# 0100*

TOP = 8     *# 1000*

*# Define window boundaries*

xmin, ymin = 50, 50

xmax, ymax = 100, 100

*# Define a function to compute the region code for a point (x, y)*

def compute\_code(x, y):

    code = INSIDE

    if x < xmin:

        code |= LEFT

    elif x > xmax:

        code |= RIGHT

    if y < ymin:

        code |= BOTTOM

    elif y > ymax:

        code |= TOP

    return code

*# Define Cohen-Sutherland line clipping algorithm*

def cohen\_sutherland\_line\_clip\_and\_draw(x0, y0, x1, y1):

*# Compute outcodes*

    outcode0 = compute\_code(x0, y0)

    outcode1 = compute\_code(x1, y1)

    accept = False

    while True:

        if not (outcode0 | outcode1):  *# If logical OR is 0, then both points are inside the clip rectangle*

            accept = True

            break

        elif outcode0 & outcode1:  *# If logical AND is not 0, then both points are outside the clip rectangle*

            break

        else:

*# Failed both tests, so calculate the line segment to clip*

*# from an outside point to an intersection with clip edge*

            x, y = 0, 0  *# Initialize coordinates for intersection*

*# At least one endpoint is outside the clip rectangle; pick it*

            outcode\_out = outcode0 if outcode0 else outcode1

*# Find intersection point*

            if outcode\_out & TOP:  *# Point is above the clip rectangle*

                x = x0 + (x1 - x0) \* (ymax - y0) / (y1 - y0)

                y = ymax

            elif outcode\_out & BOTTOM:  *# Point is below the clip rectangle*

                x = x0 + (x1 - x0) \* (ymin - y0) / (y1 - y0)

                y = ymin

            elif outcode\_out & RIGHT:  *# Point is to the right of the clip rectangle*

                y = y0 + (y1 - y0) \* (xmax - x0) / (x1 - x0)

                x = xmax

            elif outcode\_out & LEFT:  *# Point is to the left of the clip rectangle*

                y = y0 + (y1 - y0) \* (xmin - x0) / (x1 - x0)

                x = xmin

*# Now we move outside point to intersection point to clip*

*# and get ready for next pass*

            if outcode\_out == outcode0:

                x0, y0 = x, y

                outcode0 = compute\_code(x0, y0)

            else:

                x1, y1 = x, y

                outcode1 = compute\_code(x1, y1)

    if accept:

*# Draw the clipped line*

        glColor3f(0.0, 1.0, 0.0)  *# Green color*

        glBegin(GL\_LINES)

        glVertex2f(x0, y0)

        glVertex2f(x1, y1)

        glEnd()

def main():

    pygame.init()

    display = (500, 500)

    pygame.display.set\_mode(display, DOUBLEBUF|OPENGL)

    gluOrtho2D(0, 500, 0, 500)

    while True:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                quit()

        glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT)

        glColor3f(1.0, 0.0, 0.0)  *# Red color*

*# Draw the line with red color*

        glBegin(GL\_LINES)

        glVertex2f(120, 10)

        glVertex2f(40, 130)

        glEnd()

*# Draw the clipping window with green color*

        glColor3f(0.0, 1.0, 0.0)

        glBegin(GL\_LINE\_LOOP)

        glVertex2f(xmin, ymin)

        glVertex2f(xmax, ymin)

        glVertex2f(xmax, ymax)

        glVertex2f(xmin, ymax)

        glEnd()

*# Perform line clipping and draw the result*

        cohen\_sutherland\_line\_clip\_and\_draw(120, 10, 40, 130)

        pygame.display.flip()

        pygame.time.wait(10)

main()

**Output:**

A screenshot of a computer

Description automatically generated

1. *Implement Sutherland Hodgemann polygon clipping algorithm.*

Ans

**Algorithm**

1. Start

2. Read coordinates of the clipping window

3. Consider the left edge of the window

4. Compare the vertices of each edge of the polygon, individually with the clipping plane

5. Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary discussed earlier.

6. Repeat steps 4 and 5 for remaining edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.

7. Stop.

**Source Code:**

import pygame

from pygame.locals import \*

from OpenGL.GL import \*

from OpenGL.GLUT import \*

SCREEN\_WIDTH = 1000

SCREEN\_HEIGHT = 800

*# Constants defining the region codes*

INSIDE = 0

LEFT = 1

RIGHT = 2

BOTTOM = 4

TOP = 8

def lineDDA(x0, y0, xEnd, yEnd):

    dx = xEnd - x0

    dy = yEnd - y0

    x = x0

    y = y0

    if abs(dx) > abs(dy):

        steps = abs(dx)

    else:

        steps = abs(dy)

    xIncrement = float(dx) / float(steps)

    yIncrement = float(dy) / float(steps)

    glBegin(GL\_POINTS)

    for \_ in range(int(steps) + 1):

        glVertex2d(round(x), round(y))

        x += xIncrement

        y += yIncrement

    glEnd()

def calculate\_intersection(p1, p2, p3, p4):

    x1, y1 = p1

    x2, y2 = p2

    x3, y3 = p3

    x4, y4 = p4

    denominator = ((x1 - x2) \* (y3 - y4)) - ((y1 - y2) \* (x3 - x4))

*# Check if the lines are parallel or coincident*

    if denominator == 0:

        return None

    px = (((x1 \* y2) - (y1 \* x2)) \* (x3 - x4) - (x1 - x2) \* ((x3 \* y4) - (y3 \* x4))) / denominator

    py = (((x1 \* y2) - (y1 \* x2)) \* (y3 - y4) - (y1 - y2) \* ((x3 \* y4) - (y3 \* x4))) / denominator

    return px, py

def sutherland\_hodgman(subject\_polygon, clip\_polygon):

    output\_list = subject\_polygon[:]

    clip\_edges = len(clip\_polygon)

    result = []

    for i in range(clip\_edges):

        input\_list = output\_list[:]

        output\_list.clear()

        edge\_start = clip\_polygon[i]

        edge\_end = clip\_polygon[(i + 1) % clip\_edges]

        for j in range(len(input\_list)):

            current\_point = input\_list[j]

            previous\_point = input\_list[(j - 1) % len(input\_list)]

*# Check if the current point is inside or outside the clipping edge*

            if (edge\_end[0] - edge\_start[0]) \* (current\_point[1] - edge\_start[1]) - (edge\_end[1] - edge\_start[1]) \* (

                    current\_point[0] - edge\_start[0]) >= 0:

                if (edge\_end[0] - edge\_start[0]) \* (previous\_point[1] - edge\_start[1]) - (

                        edge\_end[1] - edge\_start[1]) \* (previous\_point[0] - edge\_start[0]) < 0:

*# Calculate intersection point and add it to the output list*

                    intersection = calculate\_intersection(edge\_start, edge\_end, previous\_point, current\_point)

                    if intersection:

                        output\_list.append(intersection)

                output\_list.append(current\_point)

            elif (edge\_end[0] - edge\_start[0]) \* (previous\_point[1] - edge\_start[1]) - (

                    edge\_end[1] - edge\_start[1]) \* (previous\_point[0] - edge\_start[0]) >= 0:

*# Calculate intersection point and add it to the output list*

                intersection = calculate\_intersection(edge\_start, edge\_end, previous\_point, current\_point)

                if intersection:

                    output\_list.append(intersection)

    result = output\_list

    pygame.init()

    pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT), DOUBLEBUF | OPENGL)

    glViewport(0, 0, SCREEN\_WIDTH, SCREEN\_HEIGHT)

    glMatrixMode(GL\_PROJECTION)

    glLoadIdentity()

    glOrtho(-SCREEN\_WIDTH / 2, SCREEN\_WIDTH / 2, -SCREEN\_HEIGHT / 2, SCREEN\_HEIGHT / 2, -1, 1)

    glMatrixMode(GL\_MODELVIEW)

    while True:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                quit()

        glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

        glLoadIdentity()

*# Draw the original subject polygon*

        glColor3f(0.0, 0.0, 1.0)

        glBegin(GL\_LINE\_LOOP)

        for vertex in subject\_polygon:

            glVertex2f(vertex[0], vertex[1])

        glEnd()

*# Draw the clipping window*

        glColor3f(1.0, 0.0, 0.0)

        glBegin(GL\_LINE\_LOOP)

        for vertex in clip\_polygon:

            glVertex2f(vertex[0], vertex[1])

        glEnd()

*# Draw the resulting clipped polygon*

        glColor3f(0.0, 1.0, 0.0)

        glBegin(GL\_LINE\_LOOP)

        for vertex in result:

            glVertex2f(vertex[0], vertex[1])

        glEnd()

        pygame.display.flip()

subject\_polygon = [(50, 150), (200, 50), (350, 150), (350, 300), (250, 300), (200, 250), (150, 350), (100, 350), (100, 200)]

clip\_polygon = [(100, 100), (300, 100), (300, 300), (100, 300)]

sutherland\_hodgman(subject\_polygon, clip\_polygon)

**Output:**

**A black background with a black square with red and green lines

Description automatically generated**

1. *Write a Program to Implement:*
   1. *3D Translation*
   2. *3D Rotation*
   3. *3D Scaling*

*(Consider any three-dimensional shapes given by your graphics and library and Perform these Transformations)*

**Ans:**

**Algorithm for 3D translation**

1. Start
2. Initialize the graphics mode.
3. Draw a 3D object.
4. Translation
   * Get the translation value tx, ty
   * Move the object with tx, ty (x’=x+tx,y’=y+ty, z’=z+tz)
   * Plot (x’, y’)
5. Stop

**Source Code:**

import pygame

from OpenGL.GL import \* from OpenGL.GLUT import \* from OpenGL.GLU import \*

# cube's vertices cube\_vertices = [

[-1, -1, -1], [1, -1, -1], [1, 1, -1], [-1, 1, -1], # Bottom face

[-1, -1, 1], [1, -1, 1], [1, 1, 1], [-1, 1, 1] # Top face

]

# cube's edges using vertex indices cube\_edges = [

faces

]

def draw\_cube(vertices): glBegin(GL\_LINES)

for edge in cube\_edges:

for vertex\_index in edge:

import pygame

from OpenGL.GL import \*

from OpenGL.GLUT import \*

from OpenGL.GLU import \*

*# Cube's vertices*

cube\_vertices = [

    [-1, -1, -1], [1, -1, -1], [1, 1, -1], [-1, 1, -1],  *# Bottom face*

    [-1, -1, 1], [1, -1, 1], [1, 1, 1], [-1, 1, 1]  *# Top face*

]

*# Cube's edges using vertex indices*

cube\_edges = [

    [0, 1], [1, 2], [2, 3], [3, 0],  *# Bottom face edges*

    [4, 5], [5, 6], [6, 7], [7, 4],  *# Top face edges*

    [0, 4], [1, 5], [2, 6], [3, 7]  *# Vertical edges connecting top and bottom faces*

]

def draw\_cube(vertices):

    glBegin(GL\_LINES)

    for edge in cube\_edges:

        for vertex\_index in edge:

            glVertex3fv(vertices[vertex\_index])

    glEnd()

def main():

    pygame.init()

    display = (800, 600)

    pygame.display.set\_mode(display, pygame.OPENGL | pygame.DOUBLEBUF)

    gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)

    glTranslatef(0.0, 0.0, -5)  *# Initial camera position*

    clock = pygame.time.Clock()

    while True:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                quit()

        glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

*# Original Cube*

        glColor3f(0.0, 1.0, 0.0)  *# Green color*

        draw\_cube(cube\_vertices)

*# Translated Cube*

        glPushMatrix()

        glTranslatef(1, 0, 0)  *# Translate by (1, 0, 0)*

        glColor3f(1.0, 0.0, 0.0)  *# Red color*

        draw\_cube(cube\_vertices)

        glPopMatrix()

        pygame.display.flip()

        clock.tick(60)

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

    main()

**Output:**

A green and red cube

Description automatically generated

**Algorithm for 3D Rotation**

1. Start
2. Initialize the graphics mode.
3. Draw a 3D object.
4. Rotation
   1. Get the Rotation angle
   2. Rotate the object by the angle ф
      * x’=x cos ф - y sin ф
      * y’=x sin ф - y cosф
   3. Plot (x’,y’)
5. Stop

**Source Code:**

import pygame

from OpenGL.GL import \*

from OpenGL.GLUT import \*

from OpenGL.GLU import \*

*# The cube's vertices*

cube\_vertices = [

    [-1, -1, -1], [1, -1, -1], [1, 1, -1], [-1, 1, -1],  *# Bottom face*

    [-1, -1, 1], [1, -1, 1], [1, 1, 1], [-1, 1, 1]  *# Top face*

]

*# The cube's edges using vertex indices*

cube\_edges = [

    [0, 1], [1, 2], [2, 3], [3, 0],  *# Bottom face edges*

    [4, 5], [5, 6], [6, 7], [7, 4],  *# Top face edges*

    [0, 4], [1, 5], [2, 6], [3, 7]  *# Vertical edges connecting top and bottom faces*

]

def draw\_cube(vertices):

    glBegin(GL\_LINES)

    for edge in cube\_edges:

        for vertex\_index in edge:

            glVertex3fv(vertices[vertex\_index])

    glEnd()

def main():

    pygame.init()

    display = (800, 600)

    pygame.display.set\_mode(display, pygame.OPENGL | pygame.DOUBLEBUF)

    gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)

    glTranslatef(0.0, 0.0, -5)  *# Initial camera position*

    clock = pygame.time.Clock()

    while True:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                quit()

        glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

*# Original Cube*

        glColor3f(0.0, 1.0, 0.0)  *# Green color*

        draw\_cube(cube\_vertices)

*# Rotated Cube*

        glPushMatrix()

        glRotatef(5, 5, 5, 5)  *# Rotate by 5 degrees on all axes*

        glColor3f(1.0, 0.0, 0.0)  *# Red color*

        draw\_cube(cube\_vertices)

        glPopMatrix()

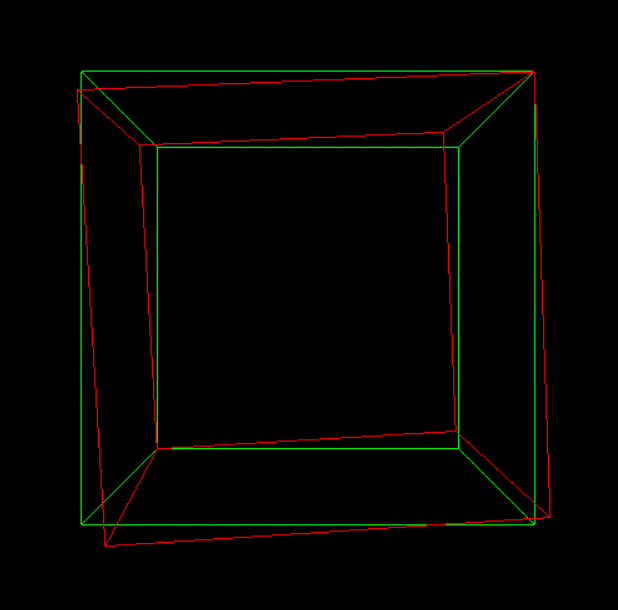
        pygame.display.flip()

        clock.tick(60)

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

    main()

**Output:**



**Algorithm for 3D scaling**

1. Start
2. Initialize the graphics mode.
3. Draw a 3D object.
4. Scaling
   * Get the scaling value Sx,Sy
   * Resize the object with Sx,Sy (x’=x\*Sx,y’=y\*Sy)
   * Plot (x’,y’)
5. Stop

**Source Code:**

import pygame

from OpenGL.GL import \*

from OpenGL.GLUT import \*

from OpenGL.GLU import \*

*# Cube's vertices*

cube\_vertices = [

    [-1, -1, -1], [1, -1, -1], [1, 1, -1], [-1, 1, -1],  *# Bottom face*

    [-1, -1, 1], [1, -1, 1], [1, 1, 1], [-1, 1, 1]  *# Top face*

]

*# Cube's edges using vertex indices*

cube\_edges = [

    [0, 1], [1, 2], [2, 3], [3, 0],  *# Bottom face edges*

    [4, 5], [5, 6], [6, 7], [7, 4],  *# Top face edges*

    [0, 4], [1, 5], [2, 6], [3, 7]  *# Vertical edges connecting top and bottom faces*

]

def draw\_cube(vertices):

    glBegin(GL\_LINES)

    for edge in cube\_edges:

        for vertex\_index in edge:

            glVertex3fv(vertices[vertex\_index])

    glEnd()

def main():

    pygame.init()

    display = (800, 600)

    pygame.display.set\_mode(display, pygame.OPENGL | pygame.DOUBLEBUF)

    gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)

    glTranslatef(0.0, 0.0, -5)  *# Initial camera position*

    clock = pygame.time.Clock()

    while True:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                quit()

        glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

*# Original Cube*

        glColor3f(0.0, 1.0, 0.0)  *# Green color*

        draw\_cube(cube\_vertices)

*# Scaled Cube*

        glPushMatrix()

        glScalef(0.5, 0.5, 0.5)  *# Scale by factors (0.5, 0.5, 0.5)*

        glColor3f(1.0, 0.0, 0.0)  *# Red color*

        draw\_cube(cube\_vertices)

        glPopMatrix()

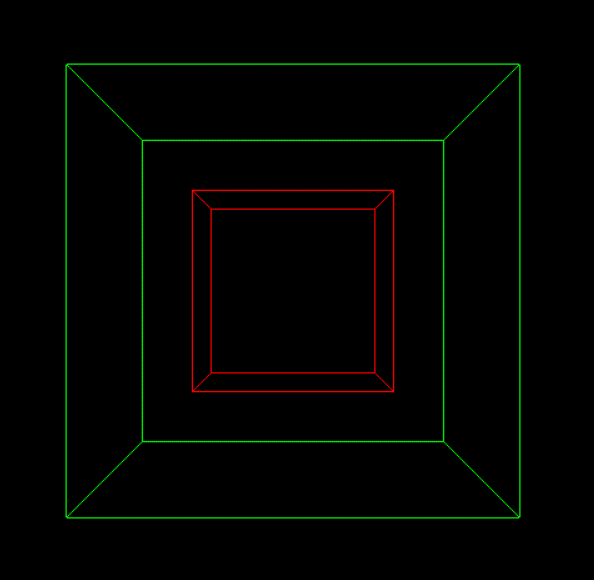
        pygame.display.flip()

        clock.tick(60)

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

    main()

**Output:**



# Conclusion:

After the completion of this lab, I learned how to transform shapes in three dimensions by using homogeneous coordinate system and transformation matrices by the use of python, Opengl APIs for python, and pygame for window creation.