Computer Graphics

Karthik S
LIT2021012
5th Semester - Information Technology
Mail: lit2021012@iiitl.ac.in

Table of Contents

> All the code to this lab can be found in my github repository : https://github.com/KarthikS373/graphics

Lab 04

Write a program to implement Liang Barsky line clipping algorithm

AIM

The aim of this lab is to implement the Liang-Barsky line clipping algorithm, a fundamental algorithm used in computer graphics for line segment clipping against a rectangular window

Introduction:

The Liang-Barsky line clipping algorithm is a widely used technique for efficiently clipping line segments against a rectangular region. It is essential in computer graphics and is used to ensure that only the visible portions of a line segment are rendered on the screen, optimizing the rendering process

Submission

How to Run

- Install poetry from [here] if not already installed
- Clone the project and install the dependencies using poetry install
- Run the project using `poetry run python path_to_main.py`

How to Use

- Launch the application
- Select the drawing algorithm by clicking the respective buttons
- Input the center coordinates and radius of the circle in the provided entry fields
- Click the "Submit" button to execute the selected algorithm and draw the circle
- View the calculation logs displayed on the GUI

Implementation Details:

- Input: The program takes as input the
  - Coordinates of the line segment endpoints P1(x_1, y_1) and P2(x_2, y_2)
  - Coordinates of the rectangular window (W_{xmin}, W_{ymin}, W_{xmax}, W_{ymax})
- Clipping: The algorithm calculates the intersection points of the line with the four sides of the window (left, right, top, and bottom) using parametric equations
- Classification: It classifies based on the intersection points, the line segment as
  - completely inside
  - completely outside
  - partially inside the window
- Output: The program outputs the clipped line segment or a message indicating that the line segment is completely outside the window

Algorithm

```
Input:
  - (x1, y1): Coordinates of the line segment's starting point (float)
  - (x2, y2): Coordinates of the line segment's ending point (float)
  - clip_xmin: Minimum X-coordinate of the clipping window (float)
  - clip_ymin: Minimum Y-coordinate of the clipping window (float)
  - clip_xmax: Maximum X-coordinate of the clipping window (float)
  - clip_ymax: Maximum Y-coordinate of the clipping window (float)

Output:
  - clipped_line: Coordinates of the clipped line segment (x1_clip, y1_clip, x2_clip, y2_clip) (tuple of floats)

Algorithm:
  Calculate the parameters p and q:
    p = [-1 * (x2 - x1), x2 - x1, -1 * (y2 - y1), y2 - y1]
    q = [x1 - clip_xmin, clip_xmax - x1, y1 - clip_ymin, clip_ymax - y1]

  Initialize variables u1 and u2 to 0.0 and 1.0, respectively

  For each of the four boundary sides (i = 0 to 3):
  - Check if p[i] is equal to 0. If true, check if q[i] is less than 0. If true, return None (line is outside and parallel to the clipping edge)
  - If p[i] is not equal to 0:
    Calculate t = q[i] / p[i].
    If p[i] is less than 0, update u1 as max(u1, t) this indicates an intersection at outside to inside for the boundary
    If p[i] is greater than or equal to 0, update u2 as min(u2, t) this indicates an intersection at inside to outside for the boundary
  - Check if u1 is greater than u2. If true, return None (line is completely outside)
```

```
Calculate the coordinates of the clipped line segment:

  - x1_clip = x1 + u1 * (x2 - x1)

  - y1_clip = y1 + u1 * (y2 - y1)

  - x2_clip = x1 + u2 * (x2 - x1)

  - y2_clip = y1 + u2 * (y2 - y1)


Return the clipped_line (x1_clip, y1_clip, x2_clip, y2_clip)
```

Code

```
def liang_barsky_algorithm(x1, y1, x2, y2, clip_xmin, clip_ymin, clip_xmax, clip_ymax):
    logs = []
    p = [-1 * (x2 - x1), x2 - x1, -1 * (y2 - y1), y2 - y1]
    q = [x1 - clip_xmin, clip_xmax - x1, y1 - clip_ymin, clip_ymax - y1]
    u1 = 0.0
    u2 = 1.0

    for i in range(4):
        if p[i] == 0:
            logs.append(f"Line is parallel to one of the clipping boundary")
            if q[i] < 0:
                return None, logs # Line is outside and parallel to the clipping edge
        else:
            t = q[i] / p[i]
            if p[i] < 0:
                u1 = max(u1, t)
            logs.append(
                f"Intersection at outside to inside for boundary {i+1}: u1 = {u1}")
```

```
        else:
            u2 = min(u2, t)
            logs.append(
                f"Intersection at inside to outside for boundary {i+1}: u2 = {u2}")
        if u1 > u2:
            return None, logs # Line is outside

    logs.append(f"Line is partially inside with u1 = {u1} and u2 = {u2}")
    clipped_line = (
        x1 + u1 * (x2 - x1),
        y1 + u1 * (y2 - y1),
        x1 + u2 * (x2 - x1),
        y1 + u2 * (y2 - y1)
    )
return clipped_line, logs
```

Screenshots