CG assignment 7

Jeferson Morales Mariciano, Martin Lettry

November 22, 2023

Ex 1 [7 points]

$$image\_size = 10 * 10$$

$$p1 = (-1, -1, 2)$$

$$p2 = (7, 1, 10)$$

Computing the intersection of p1 and p2 with the image plane, we get the below result:

$$q1 = \frac{p1}{p1_z} = (-0.5, -0.5)$$

$$q2 = \frac{p2}{p2_z} = (0.7, 0.1)$$

As we assume that the top-left corner has coordinates (1, 1) and the bottom-right one is at (10, 10), we need to convert q1 and q2 to image coordinates and round down:

$$q1 = [(-0.5 + 1) \cdot \frac{10}{2} + 1, (1 - (-0.5)) \cdot \frac{10}{2} + 1] = [3.5, 8.5] \approx [3, 8]$$

$$q2 = [(0.7 + 1) \cdot \frac{10}{2} + 1, (1 - 0.1) \cdot \frac{10}{2} + 1] = [9.5, 5.5] \approx [9, 5]$$

Computing the midpoint decider:

```
def compute_midpoint():
    x1, y1 = 3, 8
    x2, y2 = 9, 5

    x, y = x1, y1

    dx = x2 - x1
    dy = abs(y2 - y1)

    f = -2 * dy + dx
```

```
for _ in range(dx + 1):
    print(f"({x},{y})")
    x += 1
    if f < 0:
        y -= 1
        f += 2 * dx
    f -= 2 * dy
```

This gives us the following values:

(3,8), (4,8), (5,7), (6,7), (7,6), (8,6), (9,5)

So the middle value is: (6,7)
p_z can hence be calculated the following way:

$$p_z = \frac{1}{(1-\lambda)\cdot\frac{1}{p_{1z}} + \lambda\cdot\frac{1}{p_{2z}}}$$

as (6, 7) is exactly in the middle of the line between q1 and q2, we get:

$$\lambda = \frac{1}{2}$$

Hence:

$$p_z = \frac{1}{(1-\frac{1}{2})\cdot\frac{1}{2} + \frac{1}{2}\cdot\frac{1}{10}} = \frac{10}{3}$$

So z-value associated with the middle pixel (along the horizontal direction) of the line is $\frac{10}{3}$.

$$A_1 = rgb(1,0,0)$$
$$A_2 = rgb(0,1,0)$$

The rgb values of the new point will be:

$$\begin{aligned}
A &= \frac{(1-\lambda)\frac{A_1}{p_1^z} + \lambda\frac{A_2}{p_2^z}}{\frac{1}{p^z}} \\
&= \frac{(1-\frac{1}{2})\frac{\begin{bmatrix}1&0&0\end{bmatrix}}{2} + \frac{1}{2}\frac{\begin{bmatrix}0&1&0\end{bmatrix}}{10}}{\frac{1}{\frac{10}{3}}} \\
&= \frac{\frac{1}{2}\begin{bmatrix}\frac{1}{2}&0&0\end{bmatrix} + \frac{1}{2}\begin{bmatrix}0&\frac{1}{10}&0\end{bmatrix}}{\frac{3}{10}} \\
&= \begin{bmatrix}\frac{1}{4} & \frac{1}{20} & 0\end{bmatrix}\cdot\frac{10}{3} \\
&= \begin{bmatrix}\frac{10}{12} & \frac{1}{6} & 0\end{bmatrix} \\
&= \begin{bmatrix}\frac{5}{6} & \frac{1}{6} & 0\end{bmatrix}
\end{aligned}$$

Ex 2 [8 points]

Given 3 vertices, we can divide the triangle mesh into 2 separate subtriangles such that the main triangle is cut through p1.

We can now create a bounding box for each of the subtriangles and iterate over the pixels from left to right and from bottom to top.

We can use the following Barycentric properties:
- The sum of all barycentric coordinates is 1
- The ratio of the areas of the subtriangles formed by points inside or outside the mesh remains constant.

Focusing on the horizontal edge of the triangle, the left vertex has barycentric coordinates (1, 0, 0), while the right vertex has coordinates (0, 0, 1).

By employing a linear approach, we can determine the barycentric coordinates of pixels in between, utilizing the width of the bounding box. It's important to note that the barycentric coordinate associated with the top vertex remains 0.

When computing each pixel, we can calculate the ratio based on the pixel's position relative to the bounding box width/height. Subsequently, we can use this ratio to appropriately adjust the barycentric coordinates.

```
def rasterize_triangle(p1, p2, p3):
    # Sort vertices in counter-clockwise order
    vertices = sort_vertices(p1, p2, p3)

    # Divide mesh into two subtriangles
    subtriangles = divide_mesh(vertices)

    for triangle in subtriangles:
        # Construct bounding box for the subtriangle
        bounding_box = calculate_bounding_box(triangle)

        # Iterate pixels from left to right, bottom to top
        for x in range(bounding_box.left, bounding_box.right + 1):
            for y in range(bounding_box.bottom, bounding_box.top + 1):
                # Calculate barycentric coordinates
                barycentric_coords = calculate_barycentric_coords(triangle,

                # Check if the pixel center lies inside the triangle
                if is_inside_triangle(barycentric_coords):
                    # Color the pixel
                    color_pixel(x, y)
```