CS 184: Computer Graphics and Imaging, Spring 2022

Project 1: Rasterizer

Jacob Hsiung, CS184-??

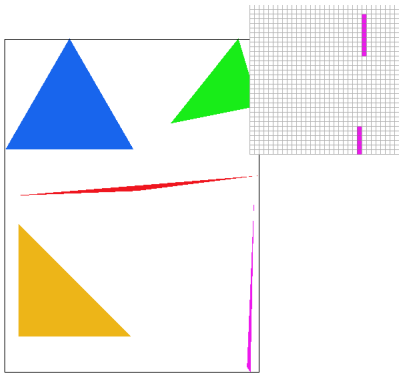
Overview

Give a high-level overview of what you implemented in this project. Think about what you've built as a whole. Share your thoughts on what interesting things you've learned from completing the project.

Section I: Rasterization

Part 1: Rasterizing single-color triangles

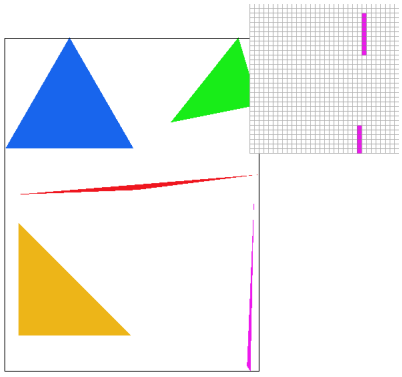
To rasterize the traingle, I first found the upper bound and lower bound of x, y. For all the pixel in the given range, we perform three line test to determine if the point is inside or outside the traingle. If all of the line tests are all positive or negative than the point is said to be within the traingle. The pixel is then sampled. This algorithim is no worse than one that checks each sample within the bounding box of the traingle because this is exactly what my algorithim does. It loop over every pixel in the bounding box and check if the pixel is inside or outside of the traingle.



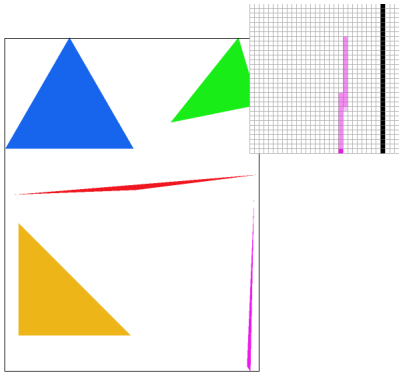
In this thin corner of triangle, the edge is disconnected

Part 2: Antialiasing triangles

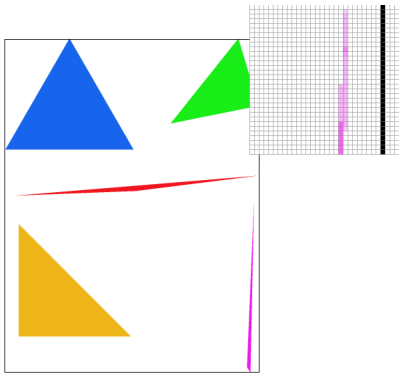
The reason why supersampling is useful because we don't need higher resolution picture, but we are sampling pixels as if they are in higher resolution. But this benifits come with the cost of higher computation and memory requirement. The algorithim is very simliar to that of part 1. I modified the algorithim by adding two more inner loop. So the algorithim now divide pixels into subpixels according to the sample rate. For each subpixels, it performs the same three line tests to determine if the subpixel is within the triangle, and color it accordinly. The main difference with super sampling is that, since we are sampling at a higher frequency, we need a bigger sample buffer to contain the result of super sampling. Hence, we need to adjust the memory size needed according to the sample rate. Also, at the end of sampling, instead of writing directly from sample buffer to framebuffer, we need to perform downsampling by averaging the value of the subpixels. Because of these sequence of caclulation, we can antialias our triangles.



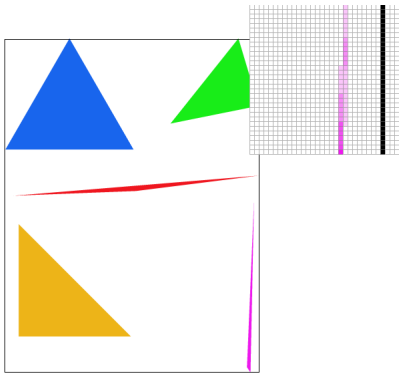
No supersampling



Supersampling rate: 4



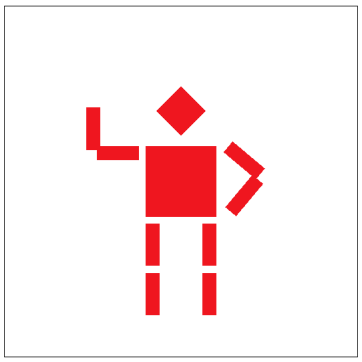
Supersampling rate: 9



Supersampling rate: 16

We can observe the difference of these zoom in area as we increase the supersampling rate. When no supersampling was performed, there was clearly a gap in the thin corner of the traingle, and the pixels we sampled just happen to be outside the traingle. However, as we start performing supersampling, this corner started to blurr so there are no obcious discontinuity in this corner. The change in the intensity of the color is because we are averaging the color over the area of this thin triangle. And since the traingle was only partially colored, so this would result in a lighter color.

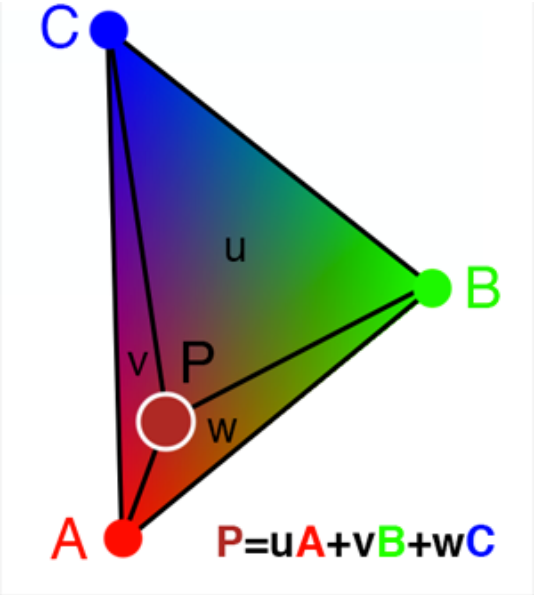
Part 3: Transforms

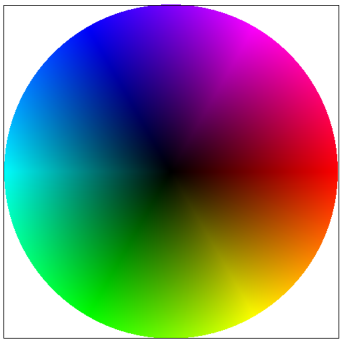


The cube man is trying to wave its right hand while putting left hand in his pocket

Section II: Sampling

Part 4: Barycentric coordinates





The best explanation of barycentric coordinate can be visualized as the triangle shown above. The three vertex can represent anything not just only color. And we can calculate the value at any point inside the triangle using interpolation. In other words, we are essentially adding the three vertex together with different weights, but the value of the three weights have to sum up to 1

Part 5: "Pixel sampling" for texture mapping

Pixel sampling is a method we can apply to map texture onto our frame buffer. Because the coordinate we used, x and y, in frame buffer and texture, u and v, is different, we need a way of mapping from texture to screen space. There are two ways of doing this, one is called nearest pixel sampling, and the other is called bilinear pixel sampling. Firstly, for nearest pixel sampling, we need to convert our x, y, coordinates into u, v coordinates. Even though the x, y, value may be discrete, the u, v, coordinates may not be. Hence, we choose the nearest texel and use that color value as our sample value. For bilinear pixel sampling, instead of directly using the nearest pixel, we use linear interpolation to obtain the texel. To begin with, we chose the 4 nearest texels. Presumably, our sample point would lie somewhere in the middle of these four points. let's say it is distance s away from the left and t away from the bottom. With the top 2 and bottom 2 points, we can use linear interpolation with parameter s to calculate the 2 texels value. With these 2 texels value, we can, again, use linear interpolation but with parameter t to obtain the desired texel value. Becuase of the effect of linear interpolation, bilinear sampling tends to produce smoother images. Below is an example where bilinear sampling clearly produce a better image than nearest sampling.

Nearest Sampling

Nearest Sampling with Supersampling

Bilinear Sampling

Bilinear Sampling with Supersampling

The difference is huge when we didn't perform supersampling. If we take a look at the zoom in portion of the, we can see that bilinear sampling produces a much smoother color transition. However, when we use supersampling, the difference between nearest sampling and bilinear sampling decreases.

Part 6: "Level sampling" with mipmaps for texture mapping

The idea of level sampling is somewhat similar to that of pixel sampling. The main difference was that, in the past, we are using nearest pixel or bilinear interpolation between pixels. In level sampling, we create levels of texture and do sampling using these levels. To be more precise, when we are trying to rasterize our scene, we often come across situations where different part of the picture need different levels of detail. This is where level sampling come into play. We create different "level" of texture where the subsequent level is the lower resolution of former level. By using different levels at different pixels, we can have fine details in closer object, and adequate detail in distant object.

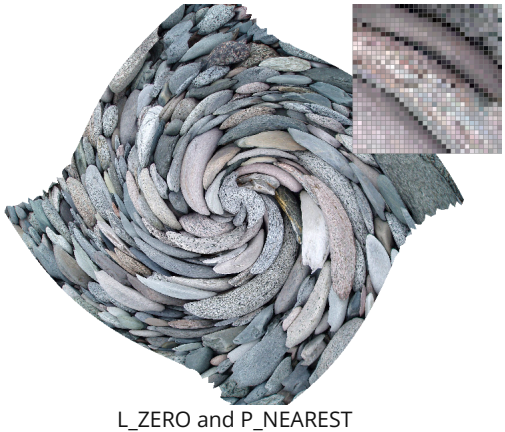
Level sampling can be further seperated into three kinds: level zero, nearest sampling, and bilinear sampling.

For level 0 sampling, we don't need to calculate any additional levels so this is the fastest implementation in level sampling, and it doesn't require additional memory other than the original texture data. All we do is just to calculate the u,v coordinates apply the texture. However, the cost of fast computation time is that we may notice that the picture has some poorly aliasing parts.

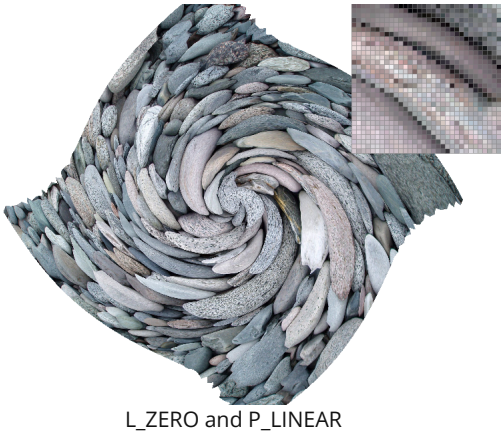
The other alternative is nearest level sampling. In this method, we do need to calculate all the levels of texture and store it so this methods would require more memory than L_ZERO sampling. After calculating the level of the pixel, we round it to the closest level and use the texture specified at that level at the u,v coordinates. With this implementation, we do have better texture mapping, but since we are choosing the nearest level, there might exist the same problem as nearest pixel sampling where the produced image has sharp texture changes

The last option is called bilinear level sampling. We also need to calculate all the levels and store it so it doesn't require more memory compared to nearest level sampling. But the big difference is that after we calculate the level instead of round to the closest level, we do a linear interpolation using the difference between levels. This way we can achieve smoother transitions but the tradeoff is this would require more computational power than nearest level sampling

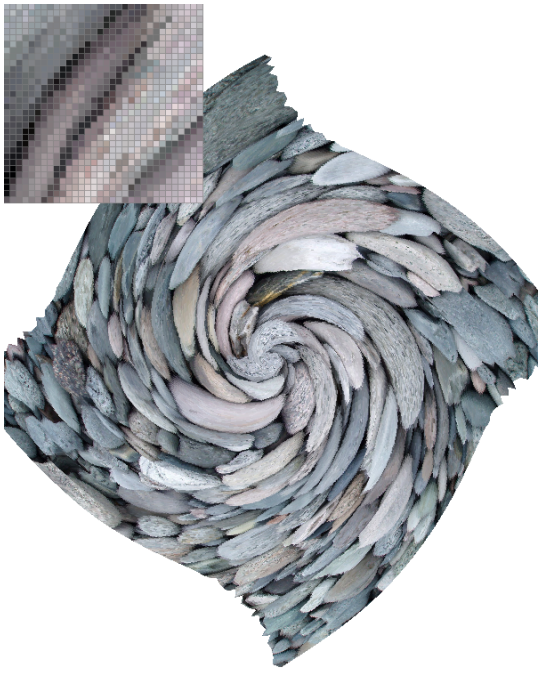
To implement level sampling, we have to consider all the different combination of pixel sampling and level sampling. So after calculating the level of the pixel we either use the nearest level or we interpolate our texture value between the levels. After deciding our level, we sample the texture value using the chosen pixel sampling methods. Within the level,if we are using nearest pixel sampling, we just pick the nearest value and that would be the return color for our pixel. If the method is bilinear pixel sampling, then we interpolate the texture value between the closest 2 pixel value.



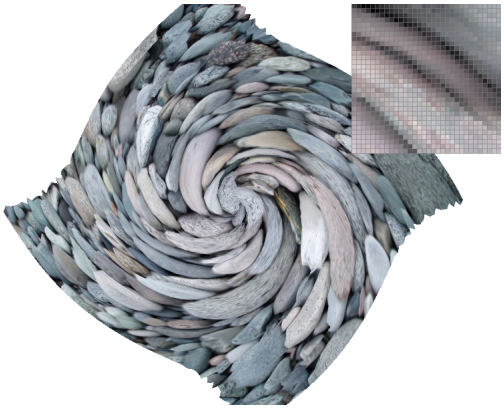
L_ZERO and P_NEAREST



L_ZERO and P_LINEAR



L_NEAREST and P_NEAREST



L_NEAREST and P_LINEAR

Section III: Art Competition

If you are not participating in the optional art competition, don't worry about this section!

Part 7: Draw something interesting!