Task1:

This task helps me fully understand of how does the rasterizing work instead of conceptually. I rasterize the triangles by sampling each pixel within the range of x\_values and y\_values, by choosing the center of each pixel and sample it. Then I applied the barycentric coordinates to solve the issue of winding order, because barycentric coordinates are always nonnegative for points inside the triangle, regardless of the winding order of the vertices. Finally, if the alpha, beta, gamma are greater or equal to zero, x and y are greater or equal to zero and within the bounding box, we can fill color for this pixel.

2. Efficiency of Algorithm:

The algorithm is efficient because it limits the point-in-triangle tests to the pixels within the bounding box of the triangle, rather than checking every pixel on the screen. By doing this, the number of checks is reduced to only those necessary, since pixels outside the bounding box cannot be inside the triangle.

A screenshot of a computer

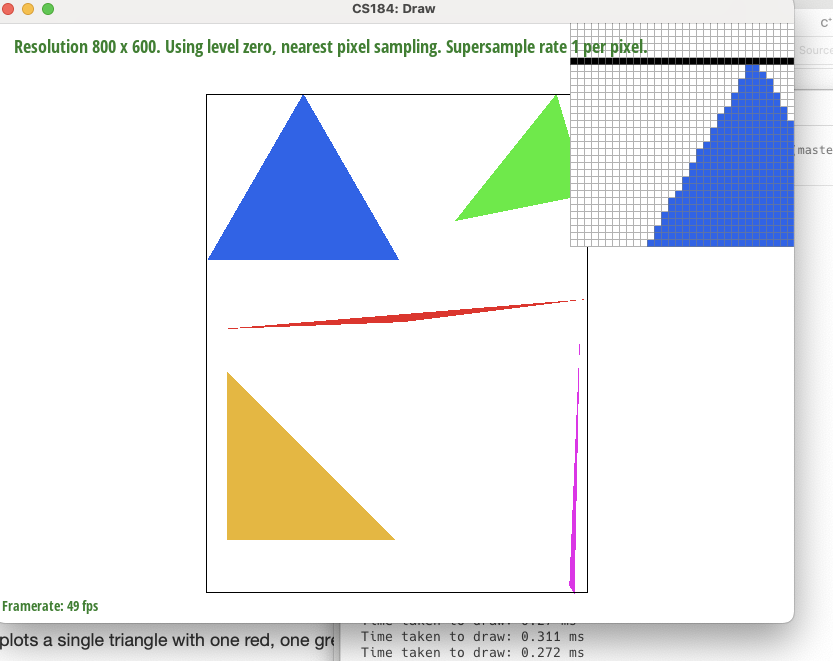
Description automatically generated

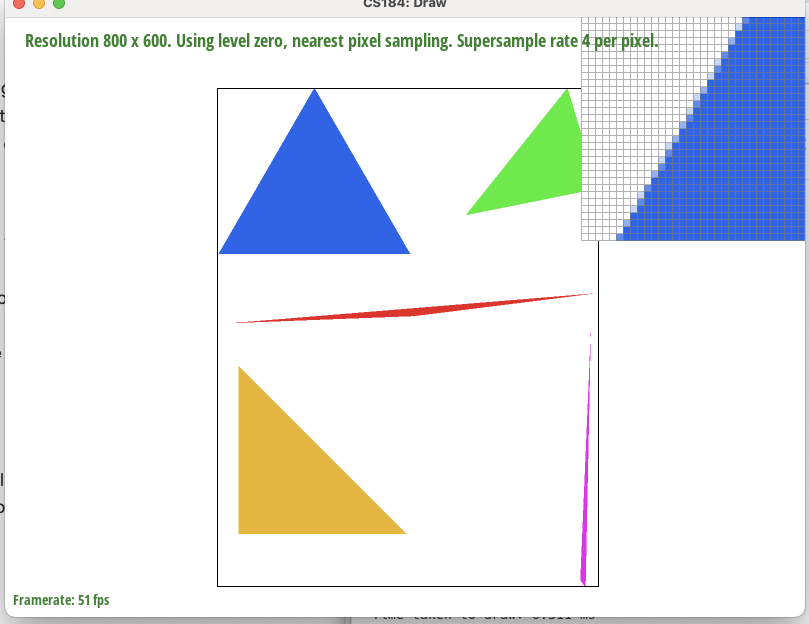
Task2:

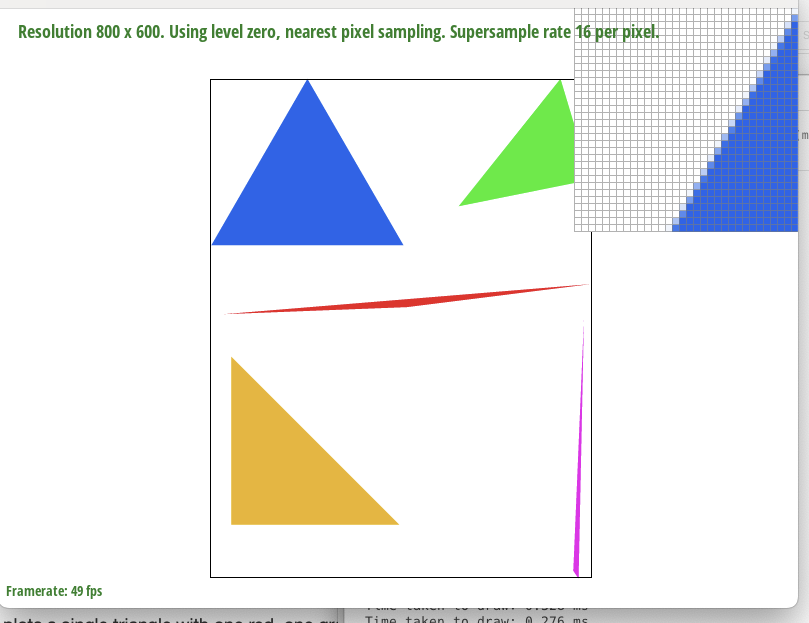
Supersampling is an antialiasing technique. It’s useful since it averages out the samples to determine the color of each pixel from higher resolution image to a low resolution image. It smoothes out the edges to make it look less jaggy. In the rasterization method, the sample buffer is a dynamic sized(determined by the sample\_rate) vector that stores diff color values for each supersample. And the buffer is resized to get high num of pixels(w\*h\*sample\_rate). And correspondingly, fill\_pixel, rasterize\_triangle, these methods are adapted to work with the supersampled buffer to calculate the position and color for each supersample in the area.

. So the set\_sample\_rate and set\_framebuffer\_target methods ensure that the sample buffer is at the correct size based on the supersampling rate and the dimension of the framebuffer. In the rasterize\_triangle method, it iterates over the bounding box of triangles at the supersampled resolution, and used barycentric coordinates to determine which supersamples are in range and should be colored. And after rendering we used the resolve\_to\_frambuffer to average out the color values of supersamples for each pixel.

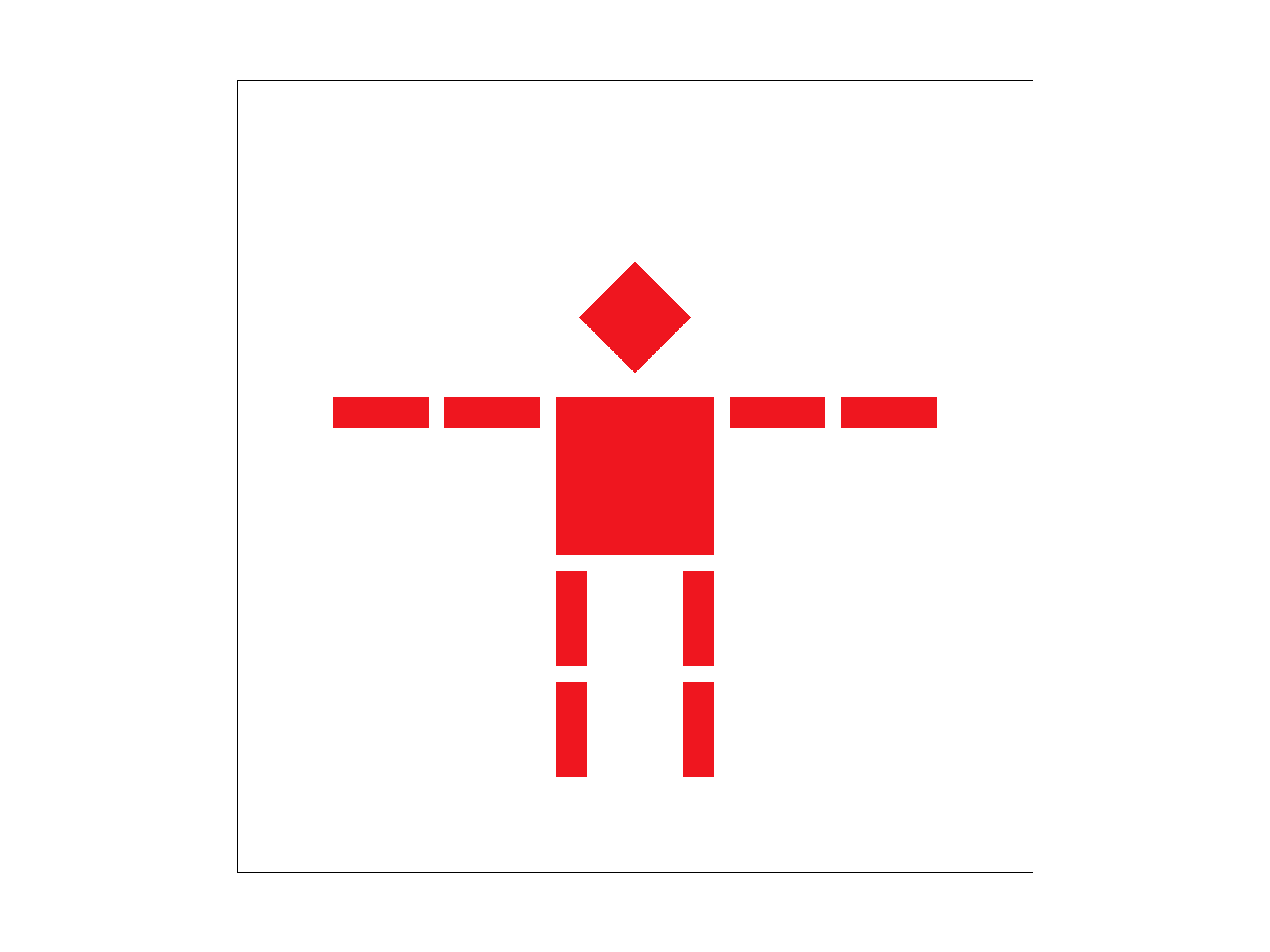
If you look at the three images below with different supersample rate. At sample 1, you should see very clear jaggies around the edges of the triangle. At the sample rate4(2\*2 supersampling), you should observed that the edges become smoother and the triangle should have a more slowly transition in terms of colors. And in the sample rate 16 (4\*4), the antialiasing effect would be even more apparent. All the edges and corners are more smoother.







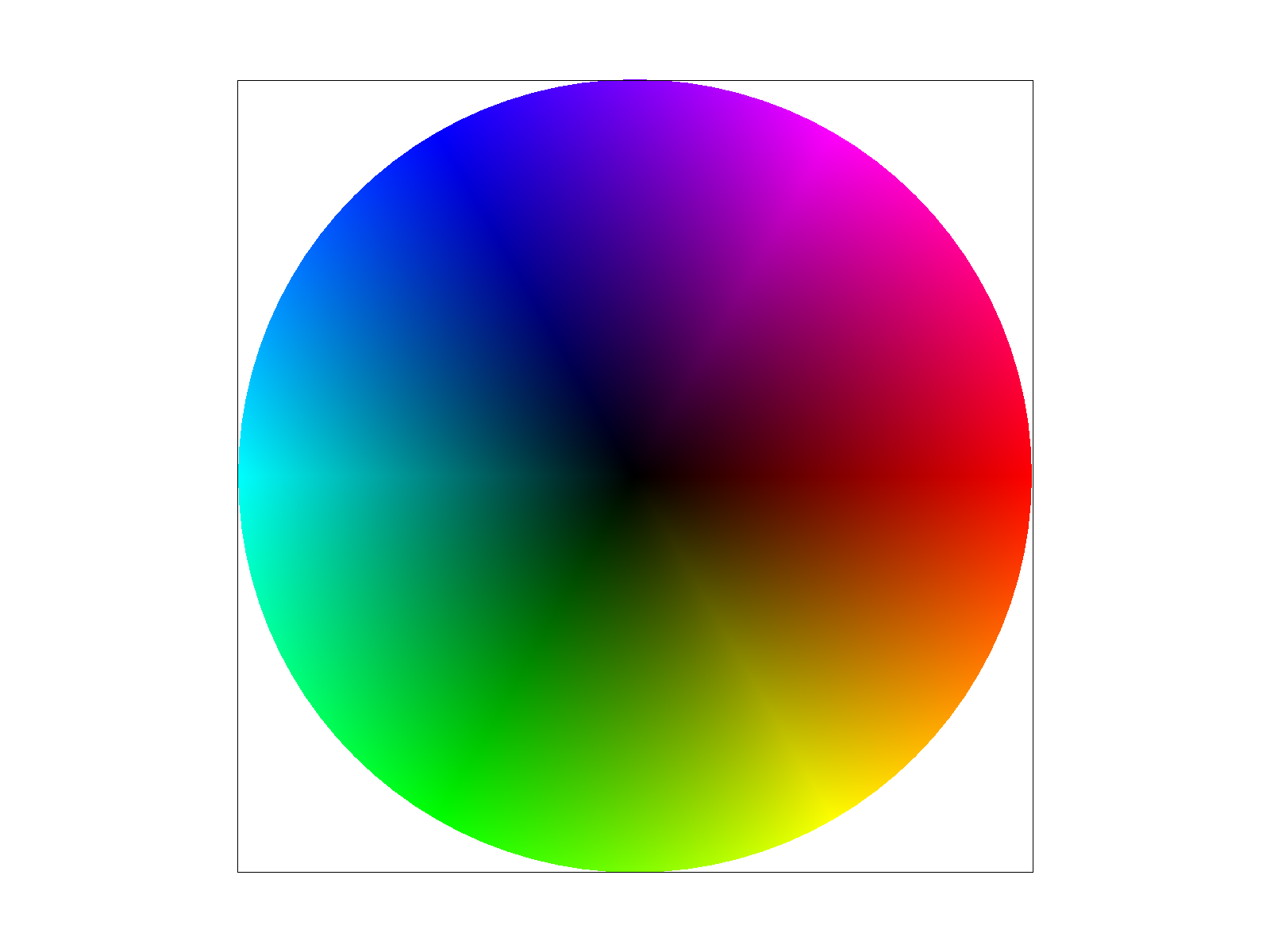
Task3:



Task4:

Based on my understanding, Barycentric coordinates are the coordinates that we calculate the proportional distances from three points, and we use the proportion of each direction as the proportion for our coordinate, then we apply the same idea for the color interpolation, use the proportional distances as the proportion of each different color proportion.

(see image below)



Task5:

Pixel sampling is the process of selecting a color value from a texture image to apply to a pixel on the screen. The way I did the texture mapping is by finding the closest grid of a point then find the texel of that point.

Nearest-Neighbor Sampling: This method is the simplest form of texture filtering. It selects the color of the nearest texel to the specified texture coordinate. Since texture coordinates are usually floating-point numbers but texels are discrete, the nearest texel is the one with the closest integer coordinates to the texture coordinate. This approach is straightforward and fast, but it can produce a blocky or pixelated image, especially when the texture is magnified or viewed at a steep angle.

Bilinear Sampling: Bilinear sampling considers the colors of the four nearest texels that surround the specified texture coordinate. It then uses a weighted average of these colors, with the weights determined by the distance of the texture coordinate from each texel's center (similar to supersampling). This method involves two steps of linear interpolation: first horizontally between the top and bottom pairs of texels, and then vertically between the results of the first interpolation. This method produces smoother images with more gradual transitions between texels.

See the below images of Nearest, bilinear texture sampling methods’ screenshots, we can see the bilinear filtering have more detailing to the picture (some orange color around the coastline), while the nearest sampling has more unclear image, and the color is not colorful as the bilinear filtering.

The largest differences between nearest-neighbor and bilinear sampling will be seen when textures are being scaled significantly up or down. Nearest-neighbor will maintain the sharpness of the original texels, but at the cost of potentially introducing blocky or aliased visuals. Bilinear filtering smooths out the transitions between texels, leading to a more visually better image when textures are scaled, but can make textures look blurry, especially when viewed up close.

Bilinear 1

Bilinear16A close-up of a globe

Description automatically generatedA close-up of a globe

Description automatically generated

Nearest1

A map of the world

Description automatically generated

Nearest16A close-up of a map

Description automatically generated

Task6: Level sampling is basically selecting the right level of detail from textures/mipmaps when rendering an object. And it’s adjusted based off the derivative of the texture coordinates to get rid of blurs and aliasing. And we use mipmap this method to make sure the image looks clear regardless of its distance by applying different layers of resolutions. So in our implementation we used du/dx and dv/dy to represent how the texture coordinates changed across the surface of the object. Then inside the get\_level method we calculated the difference of coordinates between pixels to get the rate of change. After that we scaled it by the actual dimensions to get texture’s size. Then we calculate the length of the vectors and choose the max length to represent the most rapid change in coordinates. And the final step is to calculate the mipmap level within the range.

So L\_Zero uses the base lv mipmap without any interpolation; L\_Nearest finds the nearest mipmap lv for sampling; L\_Linear interpolates between two mipmap level to get a mixed result which can be smoother. Sample\_nearest can be fast since it returns the color thats closest to the coordinates but it can turn out blurry while sample\_bilinear grabs the four closest coordinates to get a smoother image which can require more memory, power and time to compute. So overall, nearest sampling can turnout fast but very pixeled image. While more samples per pix and bilinear can be slower but comes with higher quality. And in terms of memory, Higher level of details can consume more memories and using nearest levels can save memories. Antialiasing power is very effective on bilinear since with higher sample rate reduces the jaggies but can cost more memory.