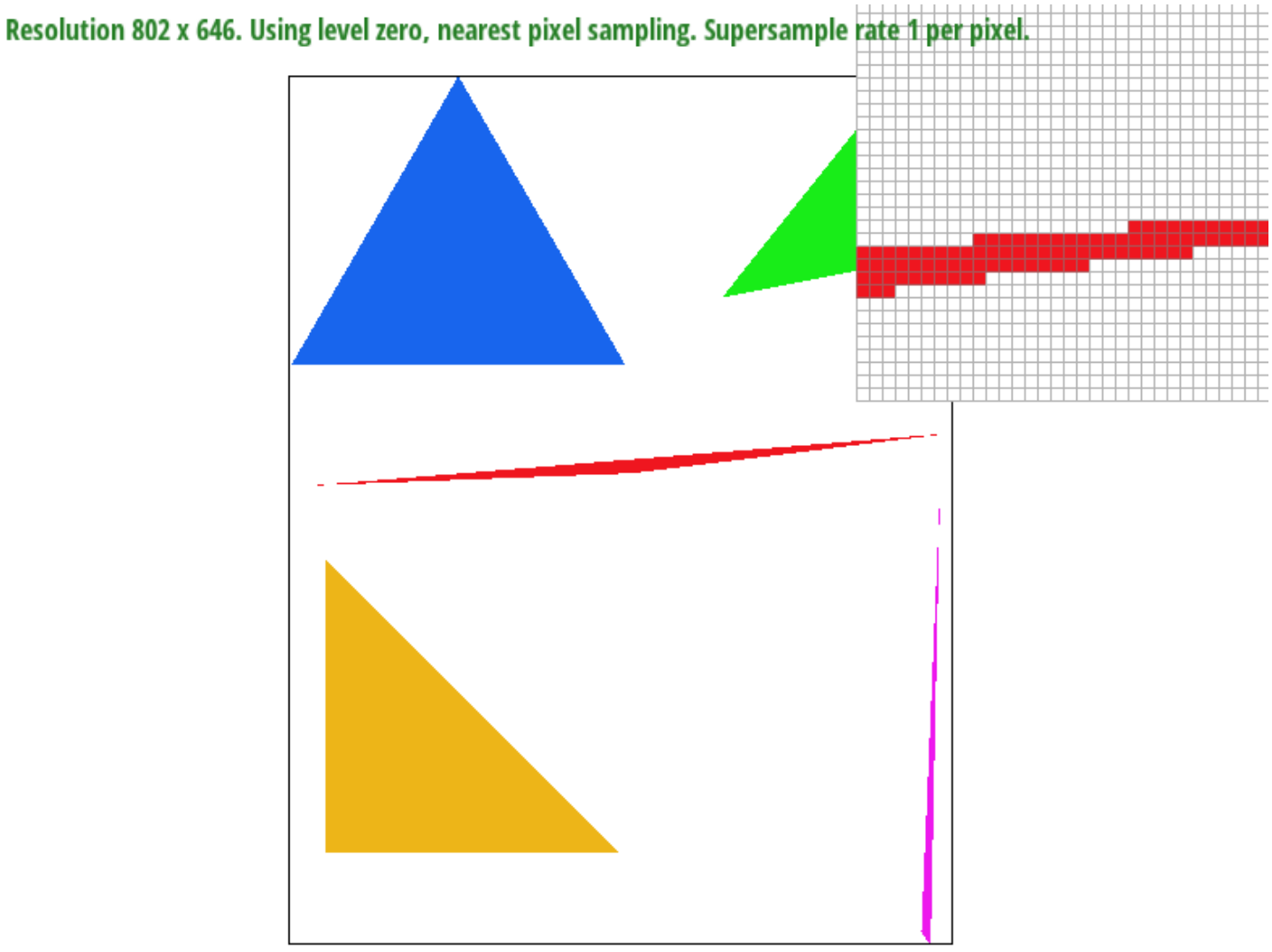
Overview

Overall in this project, I implemented some basic graphics rendering techniques from rasterizing simple triangles all the way to implementing trilinear texture map interpolation. I've learned that though the concepts of computer graphics aren't super complicated, implementing them can be quite tricky. Specifically, it's quite interesting how the slightest error in implementing computer graphics can create graphics that are totally weird and unexpected. Normally, when code isn't written correctly, it usually crashes but with computer graphics sometimes you just simply get abstract art and I thought that was very cool.

Task 1

I rasterized my triangles by first determining the bounding box of pixel locations that are both within the triangle bounding box and also within the image frame. Because of this, my algorithm is no worse than one that checks each sample within the bounding box of the triangle. My algorithm checks equal to or less than the number of samples within the bounding box of the triangle. Once the sampling population is determined, I simply iterate through each point and check to see if the point is within the triangle and then fill the pixel if it is. To make this check, my algorithm uses the which-side-of-plane check on each of the triangle edges and returns true iff the three signs of the checks are equal (IE if they are all positive or all negative).

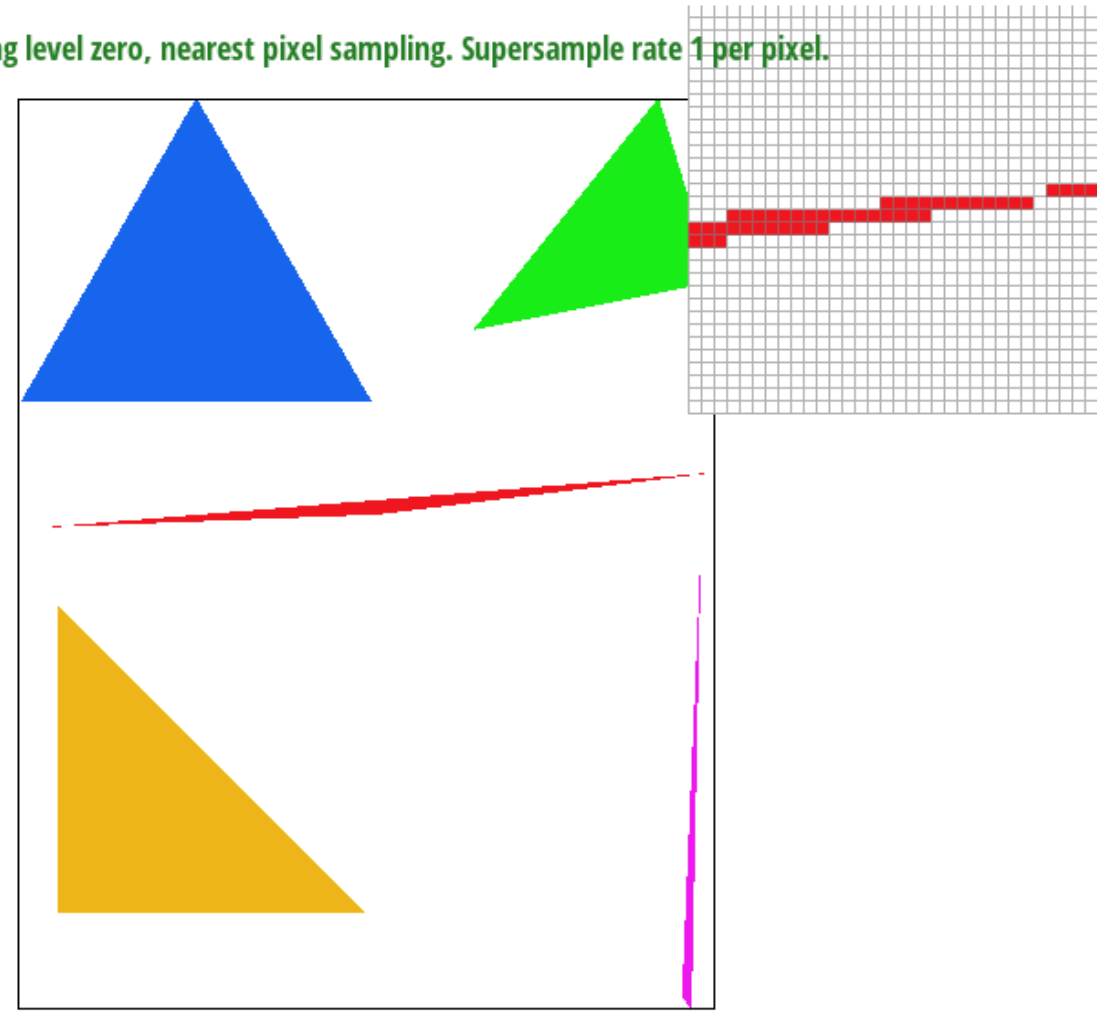


Task 2

For my supersampling algorithm, I increased the sample buffer size to include every supersampled pixel. To do this, I dynamically changed the size of the sample buffer to be width * height * sample_rate every time the frame size was changed or the sample rate was changed. I then also created a local variable called sqrt_sample_rate and that's basically

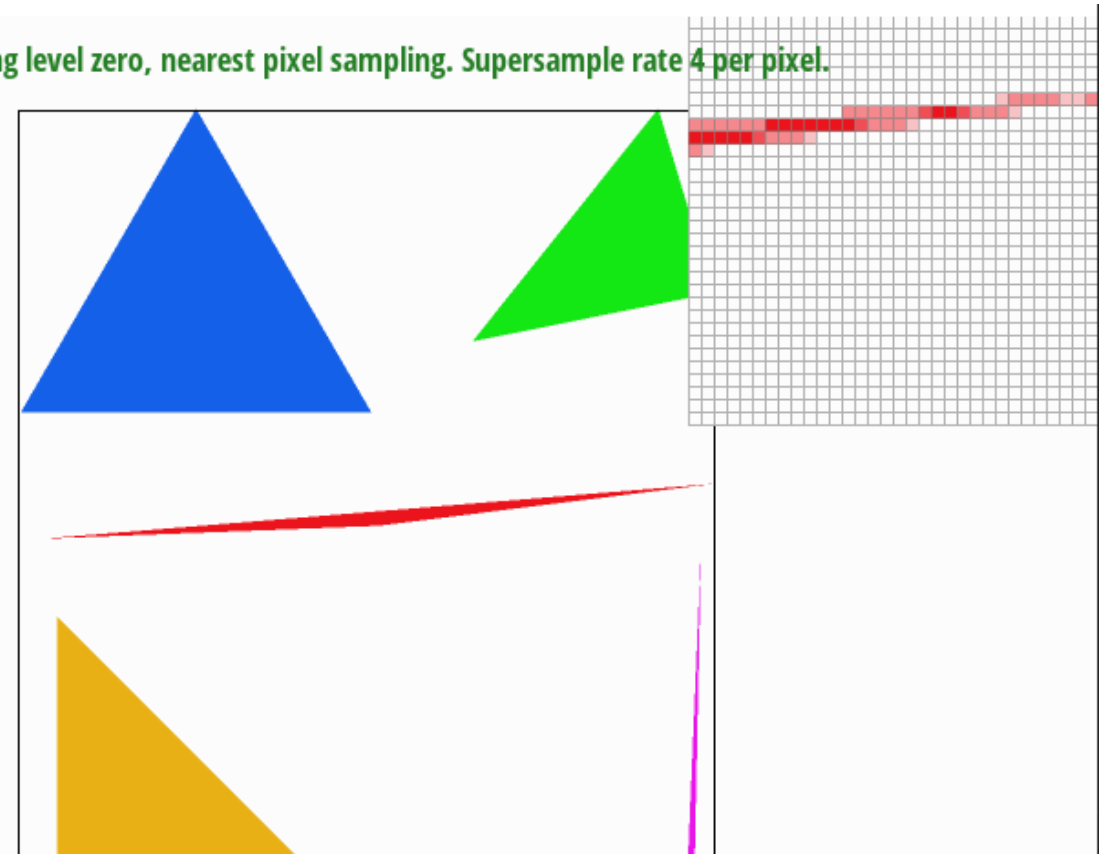
how many times per pixel per direction the image was sampled. I would then populate the sample buffer one supersample at a time storing the pixels into the sample buffer by row. Then in resolve_to_frame_buffer, I would simply iterate through the frame buffer by looping through the supersampled pixels and then I would update the rgb buffer for the pixel corresponding with the floor of the x and y position of the supersampled position divided by the sqrt_sample_rate.

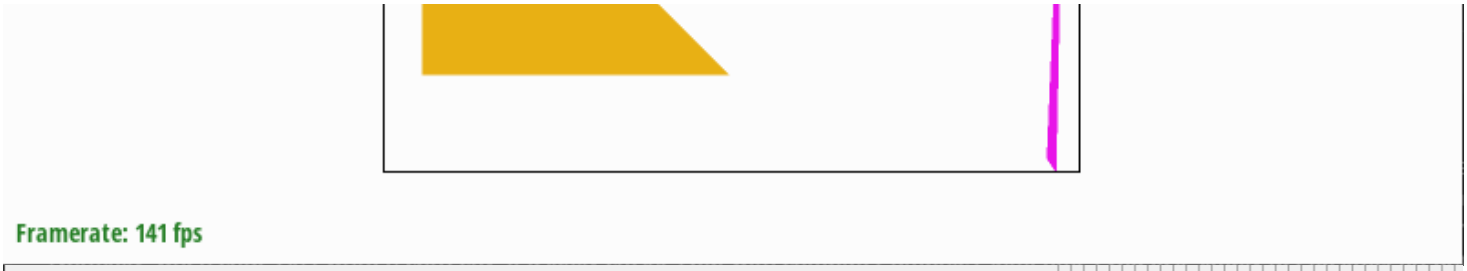
Resolution 800 x 600. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.



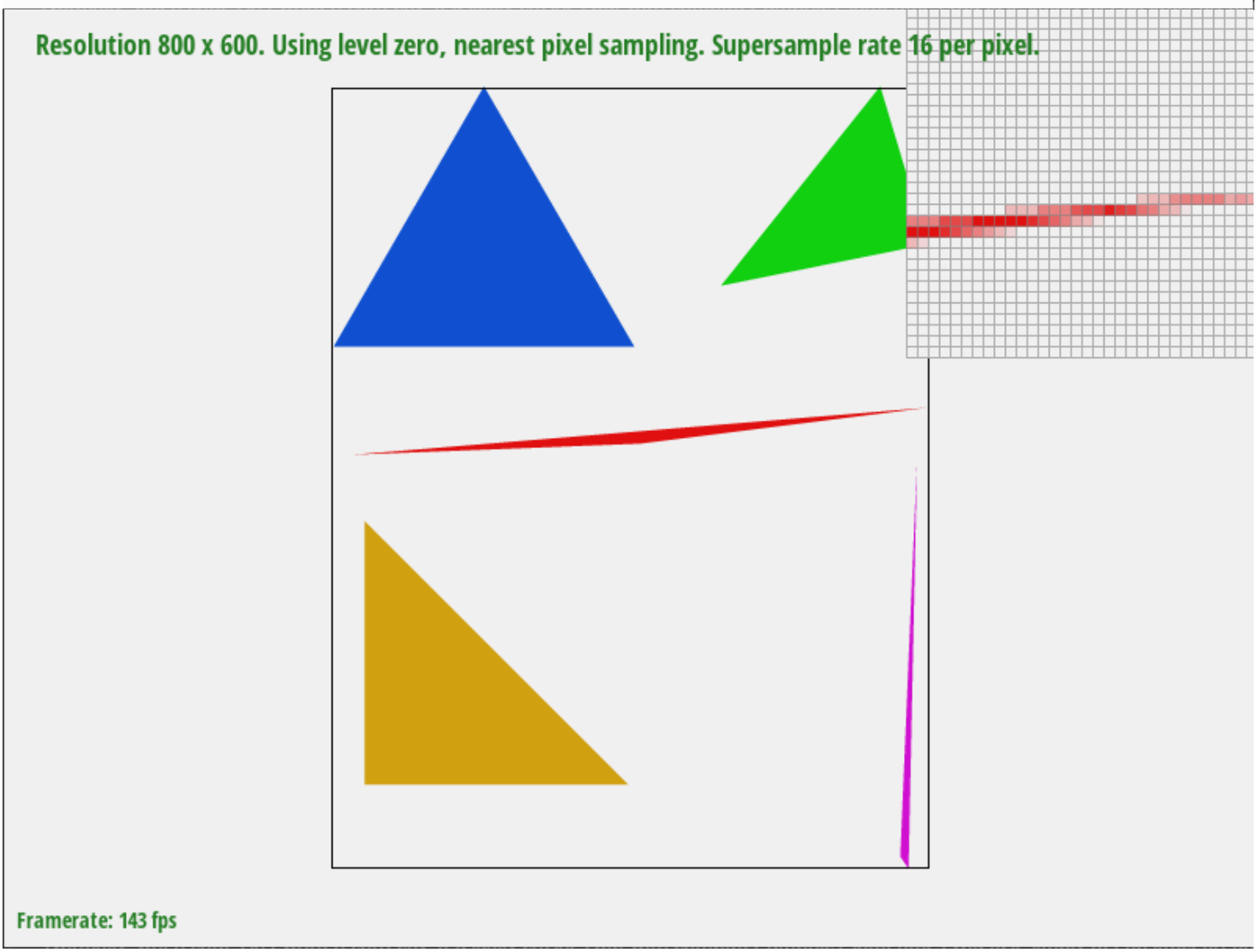
Framerate: 143 fps

Resolution 800 x 600. Using level zero, nearest pixel sampling. Supersample rate 4 per pixel.





Framerate: 141 fps

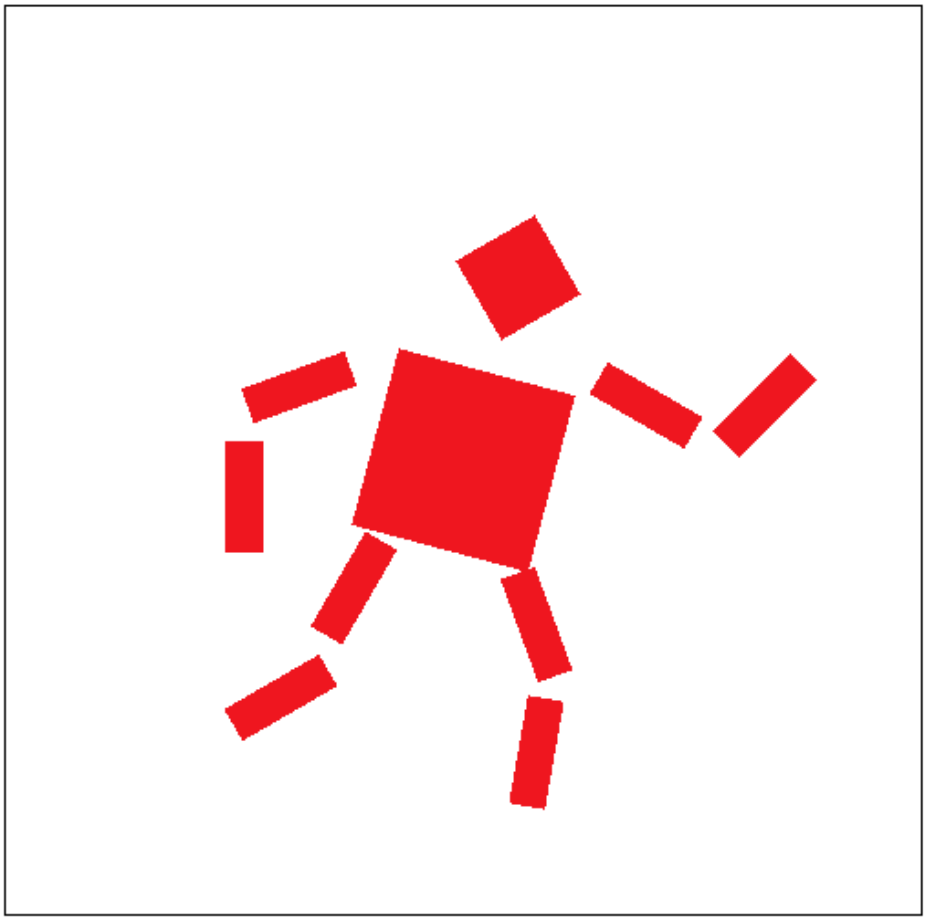


As the supersampling rate increases, we we more and more blurry edges. This is because we are taking the average value of the pixel at more places in the same spot on the image. Because the zoomed in area is on an edge of the triangle, as the sampling rate increases, that means in the areas that were once white, we are sampling more points within that pixel on the triangle and for spots that used to be red, we are now sampling more points within that pixel that aren't on the triangle.

Task 3

The cubeman is running. I angled the torso and head a little and the I positioned the legs and arms into positions that would make it look like it was mid run.

Resolution 800 x 600. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.

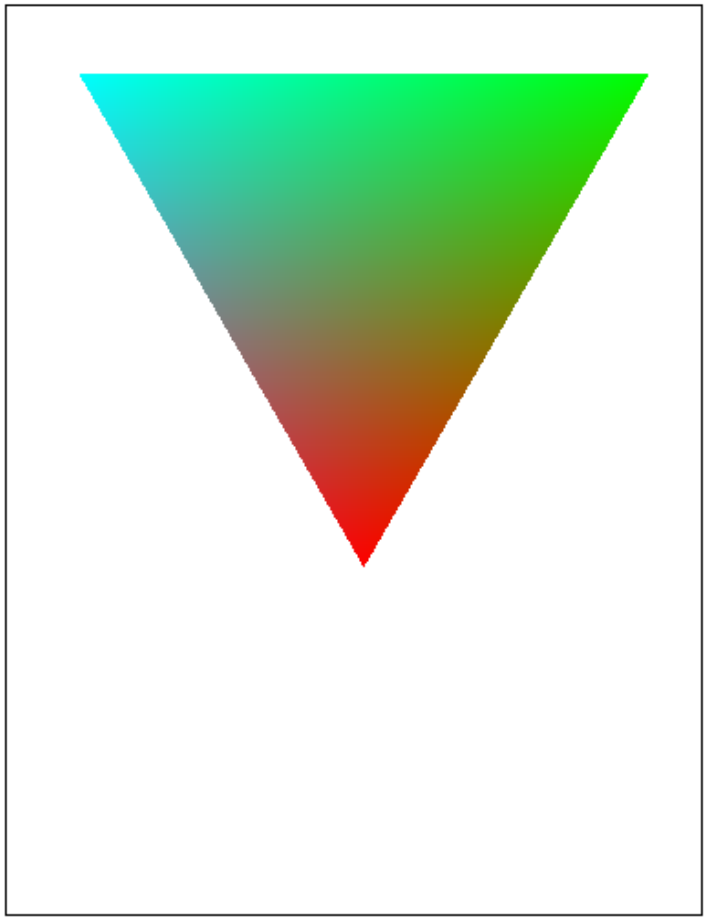


Framerate: 143 fps

Task 4

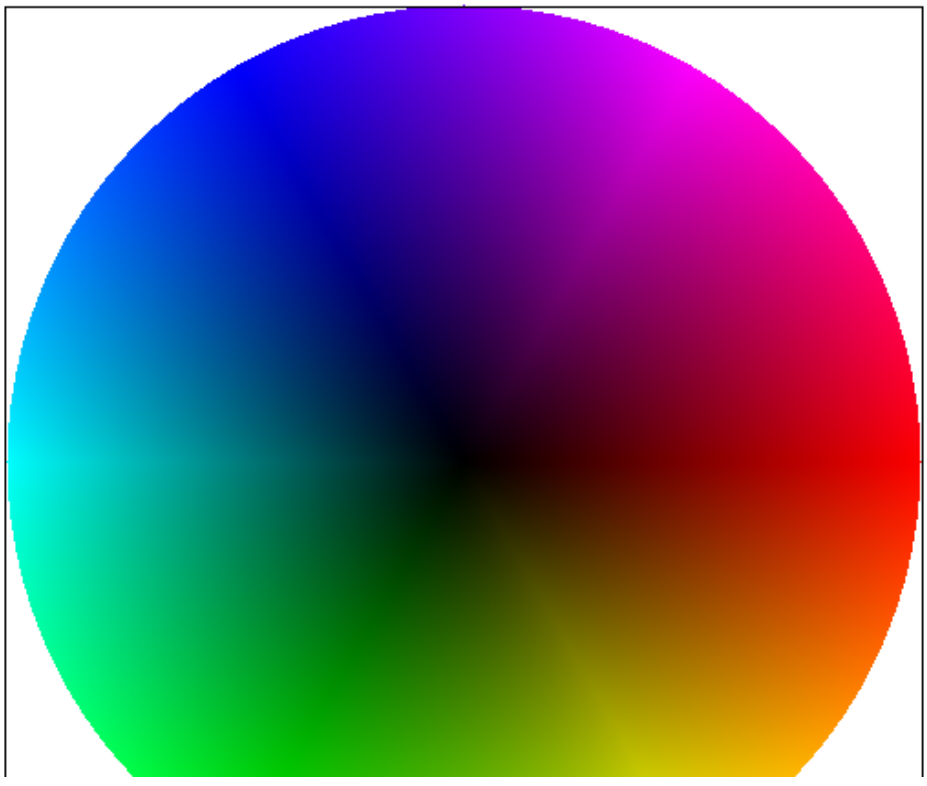
Given a triangle's three vertices and three analog values associated with those vertices, barycentric coordinates essentially give you the analog value in any place within the triangle where that analog value is proportional to its distances from each of the three vertices. The analog value should reflect the proportional distance of the point within the triangle with each of the triangle's vertices. For example, in the image below, the points closest to the blue vertex are more blue but as the points get closer to the red vertex the color becomes more red.

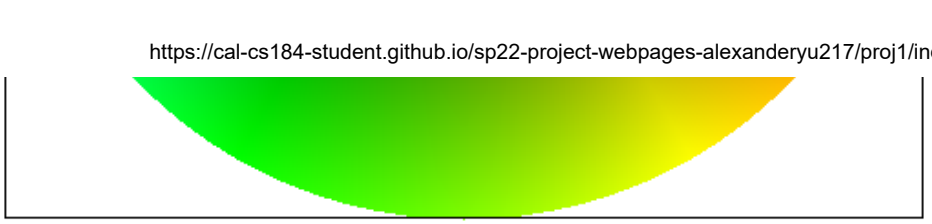
Resolution 800 x 600. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.



Framerate: 143 fps

Resolution 800 x 600. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.



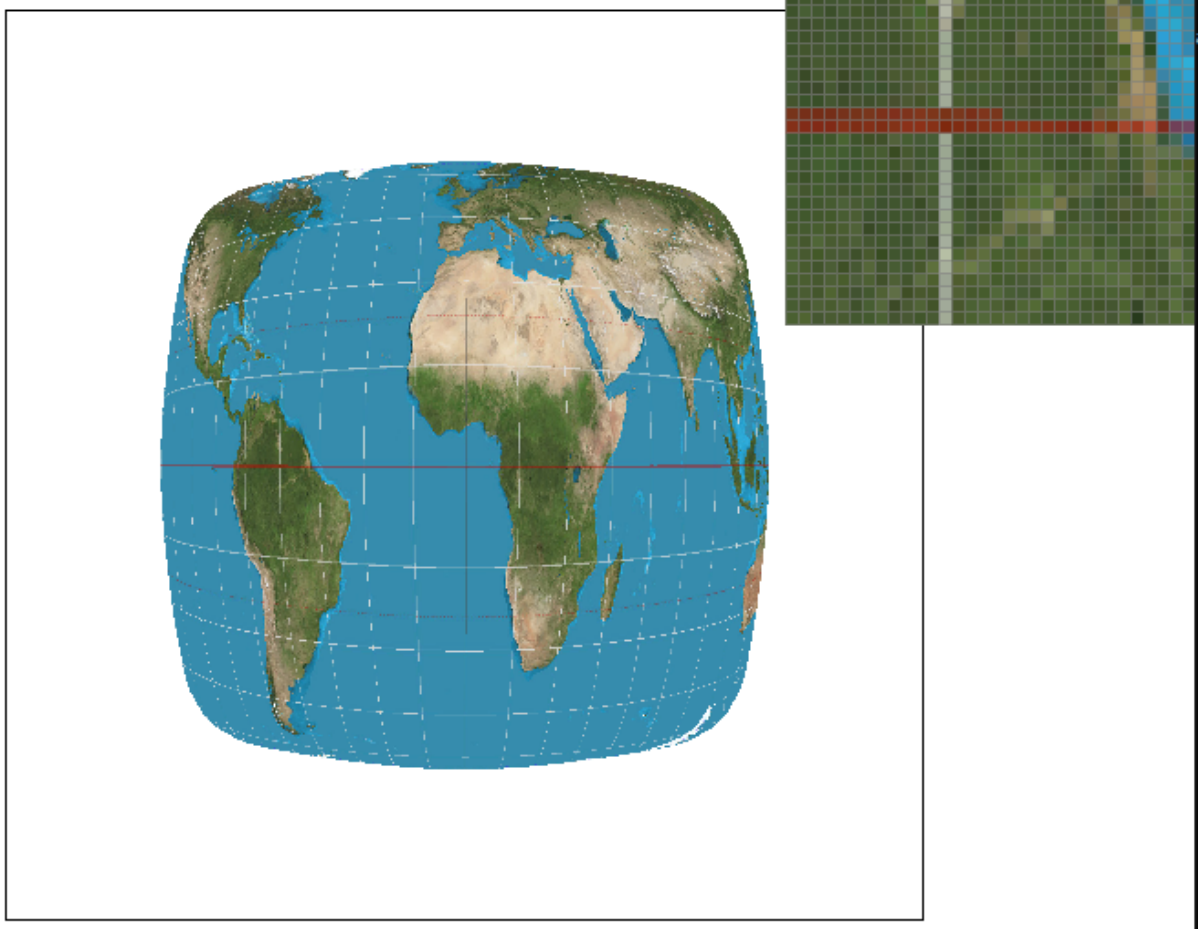


Framerate: 140 fps

Task 5

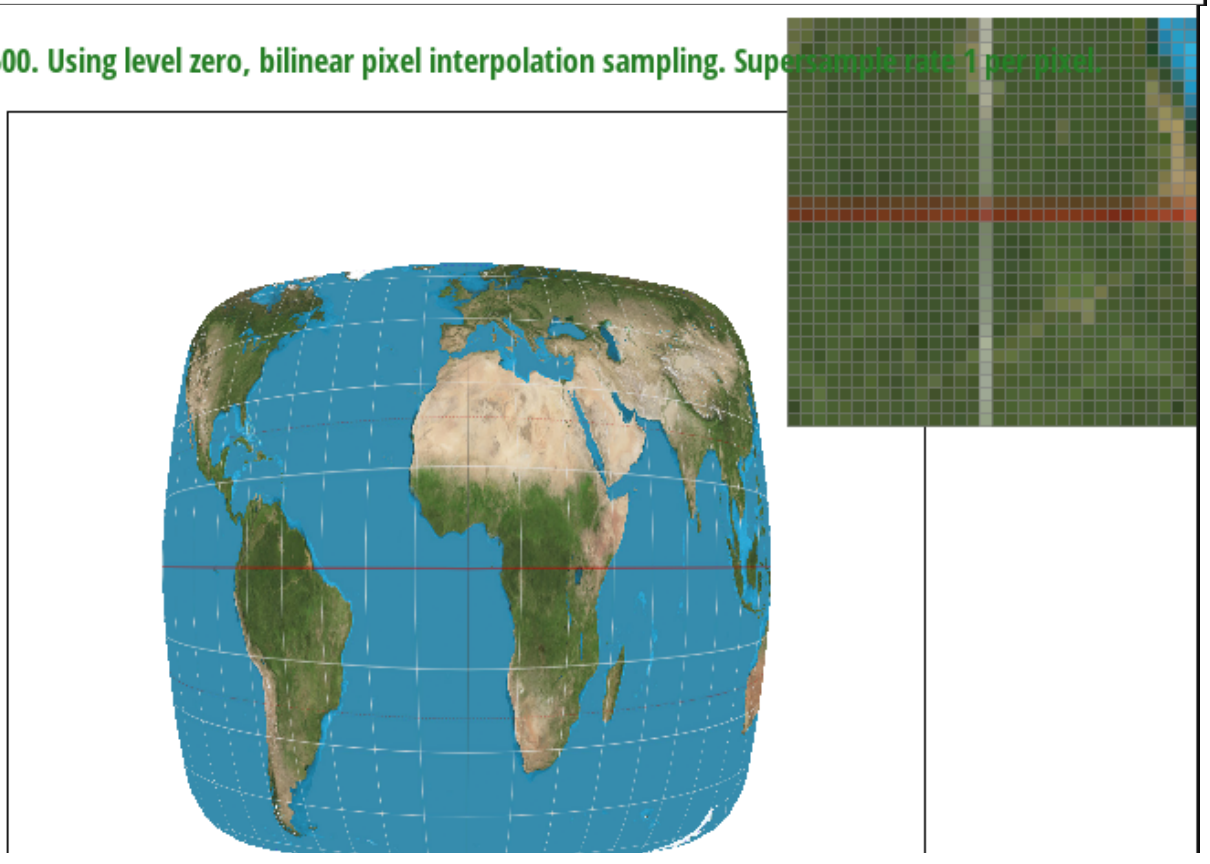
Sometimes, the pixel coordinate you are given isn't exactly where a pixel is. As a result, you need a way to determine how to sample that point and what color value to give it. This is exactly what happened when I applied textures to my image. My coordinates in the texture image space didn't necessarily correspond with a texel position exactly so I implemented nearest and bilinear interpolation sampling methods. For the nearest sampling method, you simply just pick the pixel/texel that is closest to you given coordinates. For the bilinear sampling method, you pick the four closest pixel/texel positions and interpolate your color value from the four closest pixel/texel values.

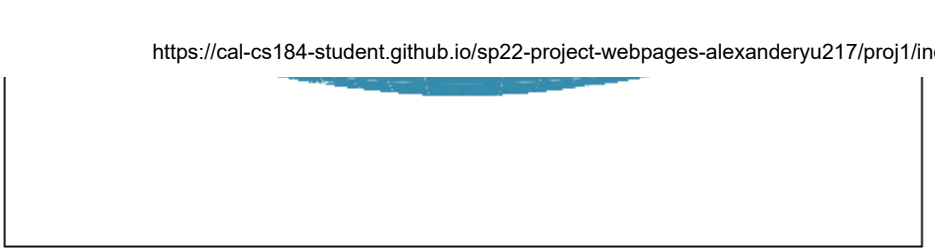
Resolution 800 x 600. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.



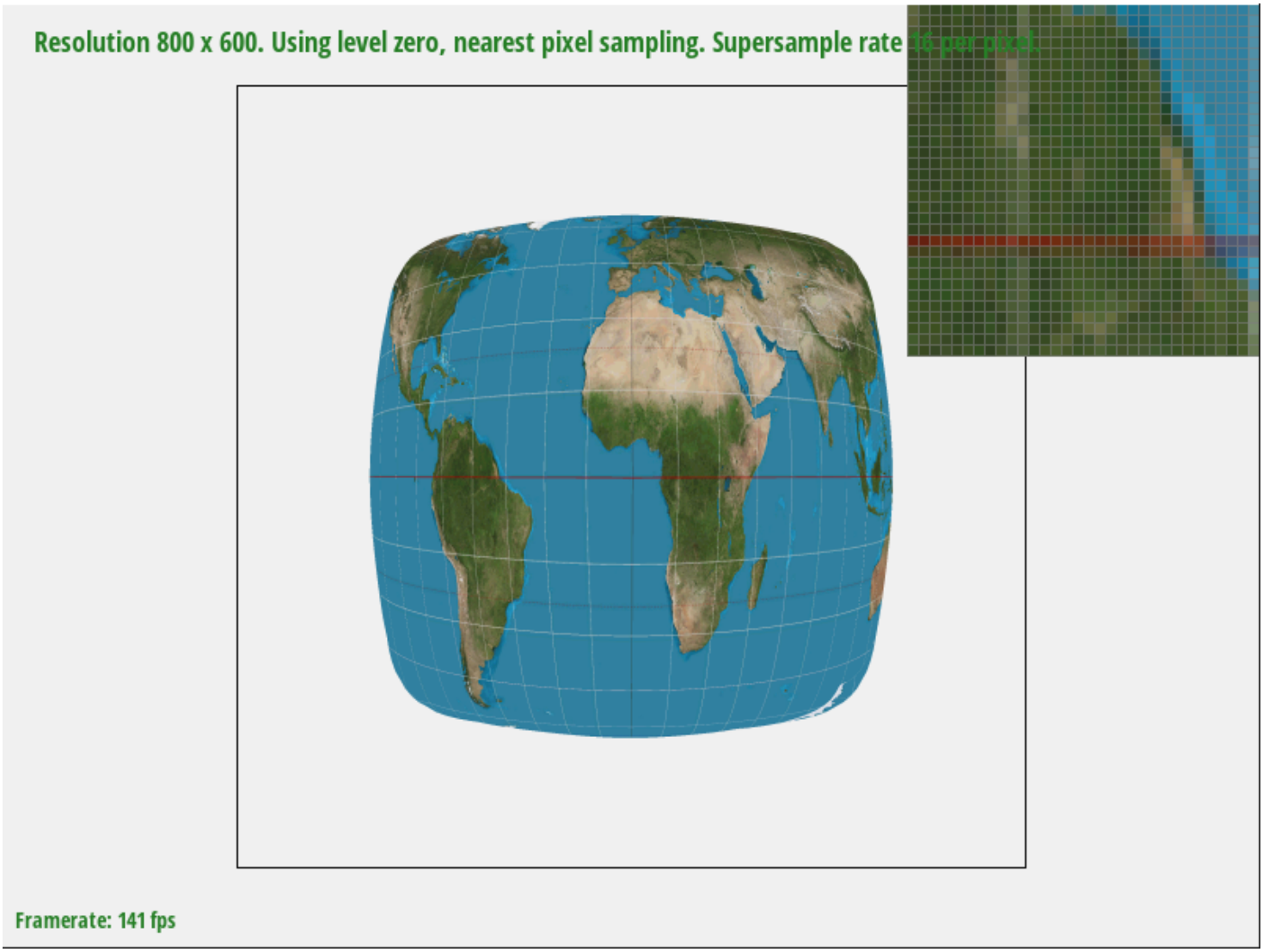
Framerate: 143 fps

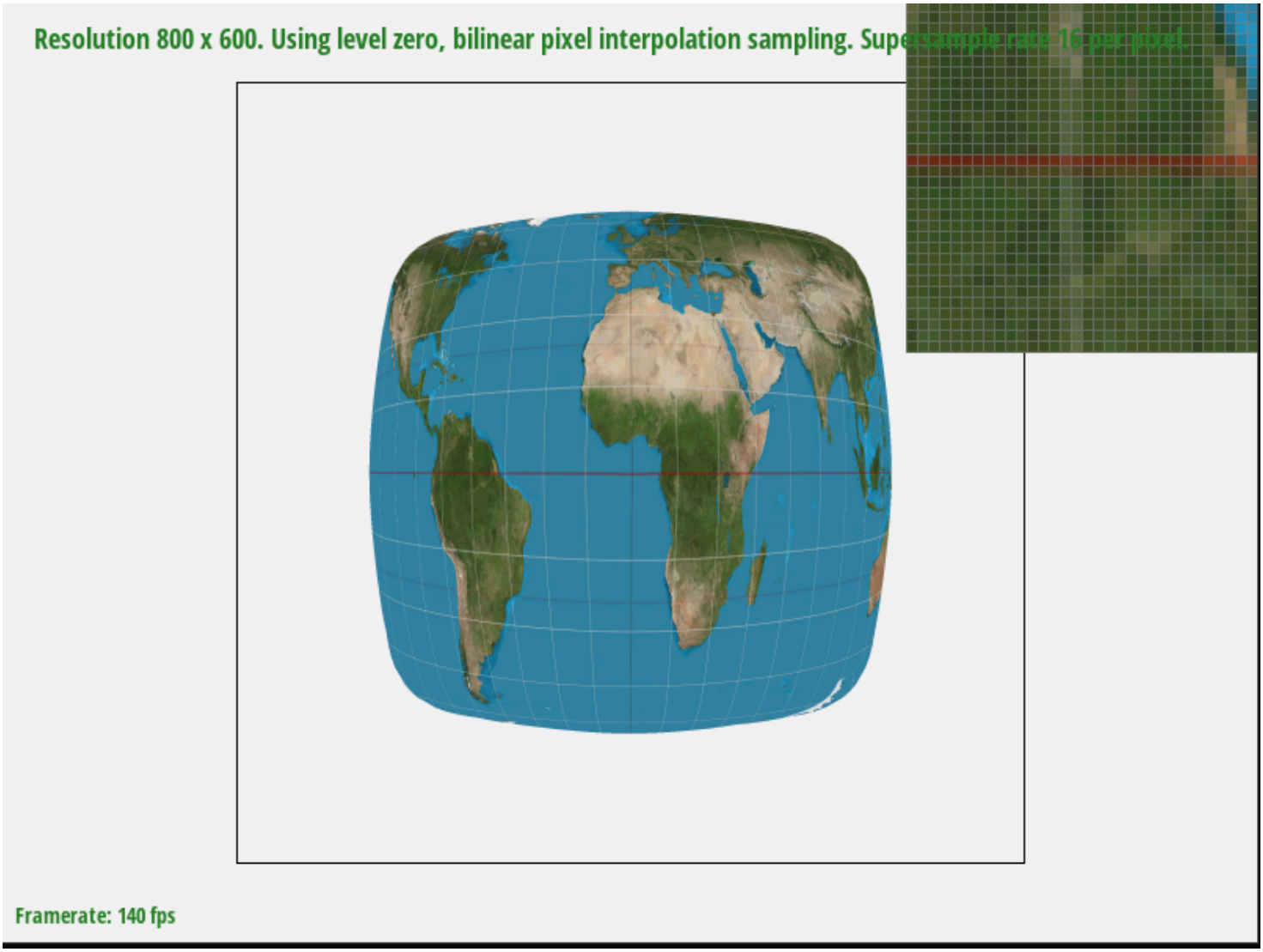
Resolution 800 x 600. Using level zero, bilinear pixel interpolation sampling. Supersample rate 1 per pixel.





Framerate: 143 fps


Resolution 800 x 600. Using level zero, nearest pixel sampling. Supersample rate 16 per pixel.
Framerate: 141 fps



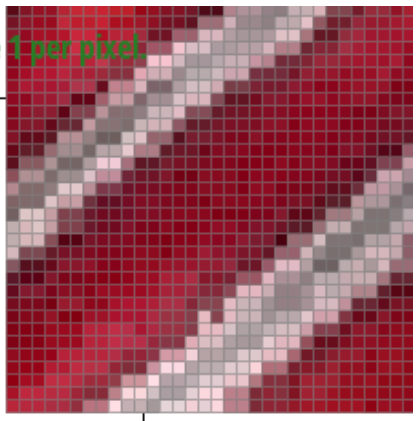
In the 1 sample rate images, you can clearly see that for nearest sampling, the equator line looks like it jumps from one pixel to the other as it goes around the globe. However, in the bilinearly interpolated image, you can see that the equator line is much more smooth and doesn't have the jumpy effect that was found in the nearest sampling method. In addition, when the images were supersampled at 16 samples per pixel, there was not a significant difference in the image quality across the two pixel sampling methods. In both the 16 sample rate images, the lines became softer and more continuous than in the 1 sample rate images. Thus, the largest difference between the two methods will come when the sampling rate is low because in bilinear interpolation, you are already taking the average of several pixel values to perform some anti aliasing on the image.

Task 6

When you have differently sized triangles in the image and you apply a texture over them, your sampling rate across the texture image will change. As a result, you sometimes may wish to sample from texture maps of different resolutions to avoid aliasing problems. To do this, I calculated rate of change of the u and v coordinates in the texture space given a point in the image frame space. Using these rates, I performed a simple mathematical calculation to determine which level I should be sampling my textures at. I implemented three ways of level sampling. First, I implemented zero level sampling which is essentially just simply sampling the zeroth level texture map. Then I implemented nearest level and used the level closest to that returned by my mathematical calculation. Finally, I implemented linear interpolation texture mapping which is just applying linear interpolation to two points with one sampled from the lower level and one sample from the higher level texture map. The lower and upper level are just the floor and ceiling of the floating point value returned from the texture map level calculation. In terms of speed, level sampling is the fastest method, pixel sampling is the second fastest, and supersampling is the slowest. In terms of memory usage, it's the same order. However, for antialiasing power, I would say that supersampling comes in first, level sampling comes in second, and then pixel sampling comes in last. No matter the other antialiasing methods, when the image gets supersampled sufficiently, I've found that the quality of the images don't differ much. When using level sampling, I've noticed that the image gets rid of aliasing artifacts by getting blurrier overall.

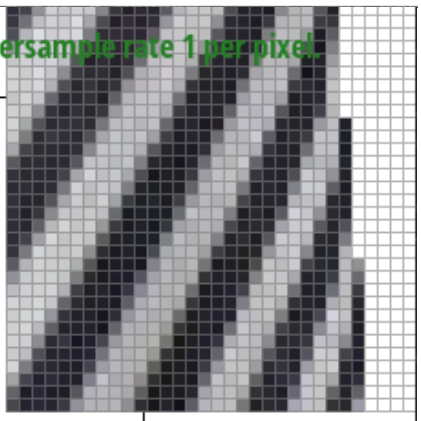
For pixel sampling, there's a small but noticeable difference when you use bilinear filtering vs nearest pixel filtering as edges get smoother and you don't get as many jaggies.

Resolution 800 x 600. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.



Framerate: 139 fps

Resolution 800 x 600. Using level zero, bilinear pixel interpolation sampling. Supersample rate 1 per pixel.

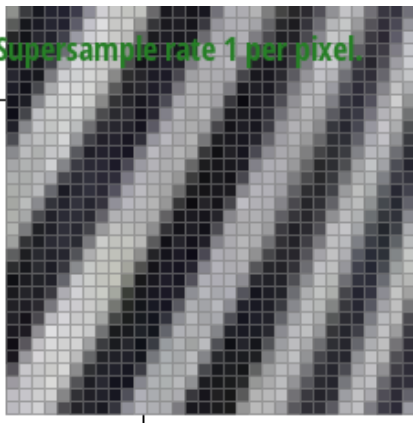


Framerate: 141 fps

Resolution 800 x 600. Using nearest level, nearest pixel sampling. Supersample rate 1 per pixel.

Framerate: 140 fps

Resolution 800 x 600. Using nearest level, bilinear pixel interpolation sampling. Supersample rate 1 per pixel.



Framerate: 142 fps

https://cal-cs184-student.github.io/sp22-project-webpages-alexanderyu217/proj1/index.html