Project 1 Write-Up

Task 1

Walk Through:

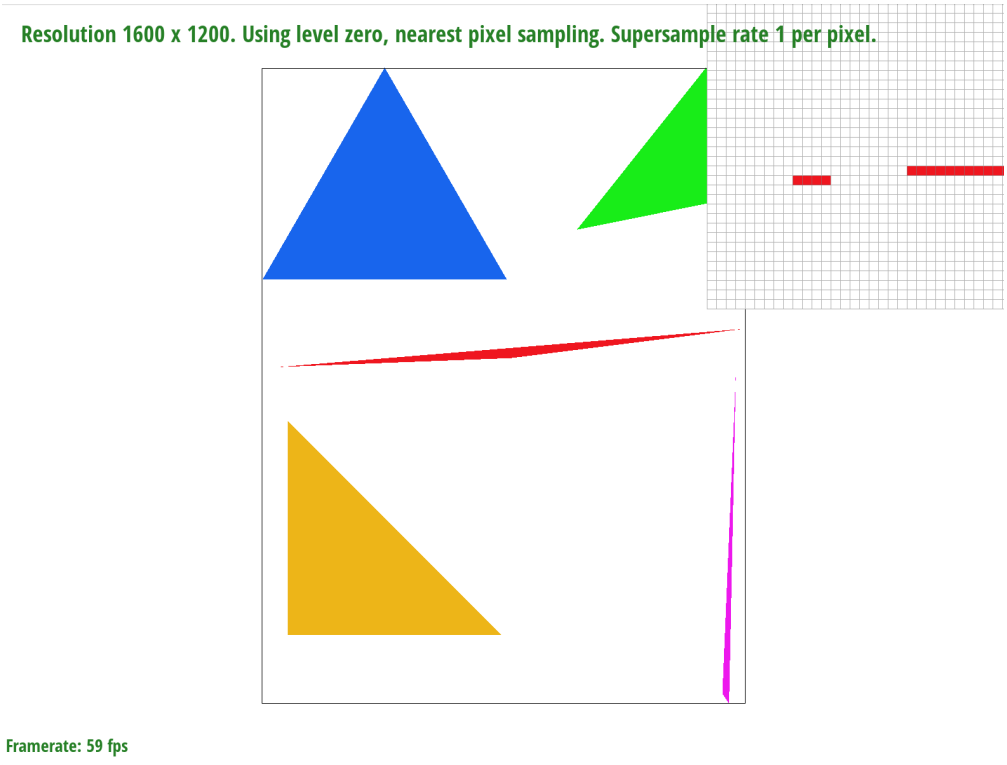
I rasterized triangles using the point in triangle sampling method. The idea is to, for every pixle, check that the center of the pixel is within the traingle. If it is, we fill it with color.

I start by setting up a double for loop to check every single pixel. The pixel location being checked is x + 0.5 and y + 0.5 - the center of the pixel. Then, using the point in triangle formula, I do some math to calculate whether the center point is in the pixel. There needs to be a check for both directions of navigating the trangles edges. Then if the values are greater than zero, the pixel is in the triangle and it gets colored.

Explain how your algorithm is no worse than one that checks each sample within the bounding box of the triangle.

My algorithm is no worse than one that checks each sample within the bounding box because the bulk of the work happens when the pixel is in the triangle. Since the bounding box will always be larger than the triangle, my algorithm will always compute less pixels that whats in the box.

Interesting Part of SVG4

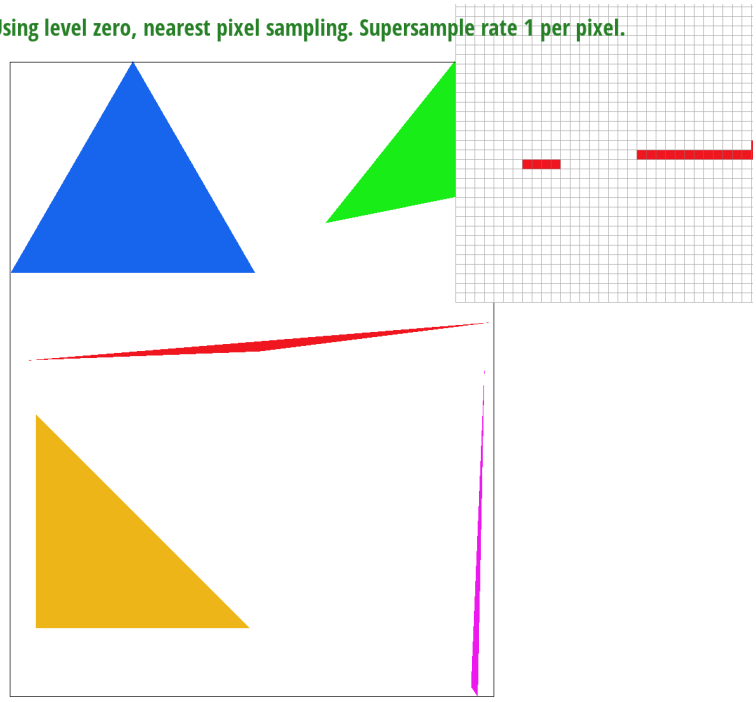


Task 2

Super Sampling Algorithm

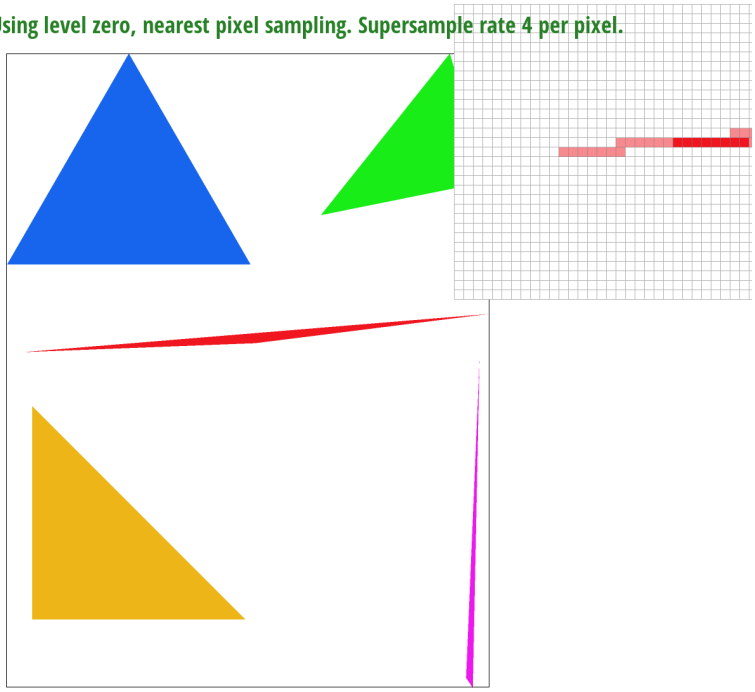
For the super sampling algorithm, I modified my algorithm from task 1 to incorporate it. Similar to task 1, i sampled at every pixel however each pixel needed to be sampled sample rate times at specific locations. So I added a second double for loop to sample sample rate locations per pixel and write directly to sample_buffer. In resolve_to_framebuffer, I averaged out every sample_rate number of values to write to the frame buffer. So every color rendered is the average of sample_rate values. I changed how fill_pixel operates since the sample_buffer size gets changed relative to sample_rate. So fill_pixel now writes sample_rate number of values into sample_buffer for each point or line. Supersampling is useful

Resolution 1600 x 1200. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.



Framerate: 59 fps

Resolution 1600 x 1200. Using level zero, nearest pixel sampling. Supersample rate 4 per pixel.

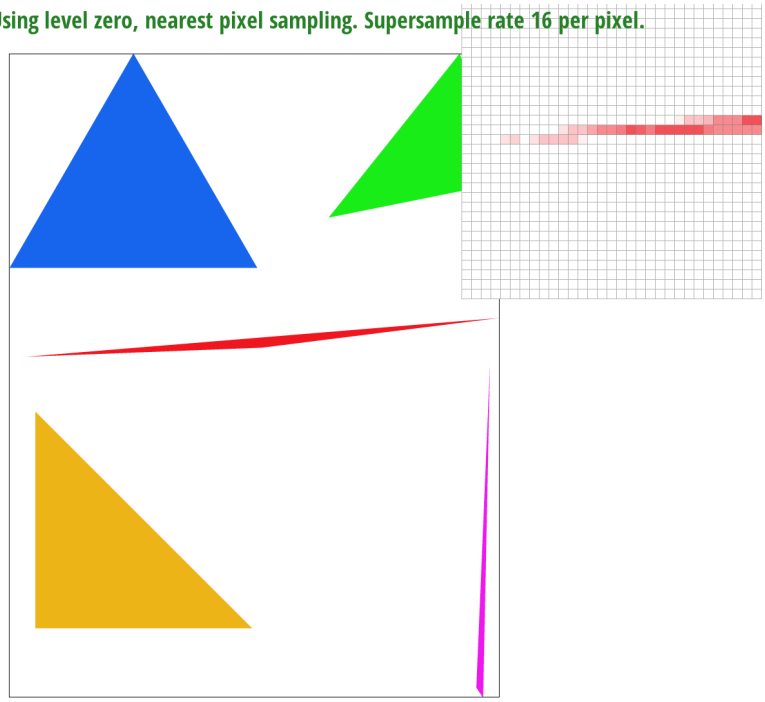


Framerate: 59 fps

Resolution 1600 x 1200. Using level zero, nearest pixel sampling. Supersample rate 9 per pixel.

Framerate: 59 fps

Resolution 1600 x 1200. Using level zero, nearest pixel sampling. Supersample rate 16 per pixel.



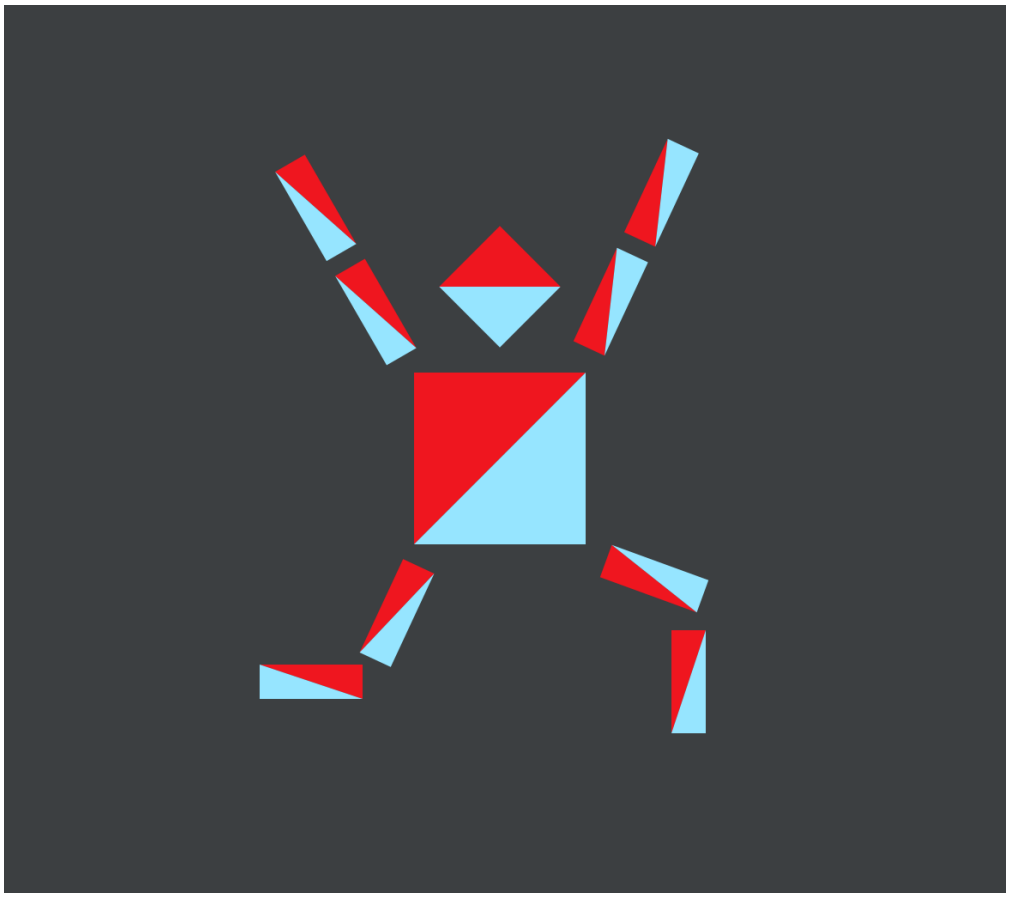
ramerate: 58 fps

Task 3

For my cubeman, I tried to do something funny and make it look like cubeman is running with his arms up. The steps I took to make cubeman look funnier was to include rotations. Specifically, I added rotations to the specific upper leg parts on both right and left legs. THen I added rotation transforms on the lower legs. Once rotated, I edited the translation transforms already in place for the body parts to line up.

For the arms, I did a similar thing. Instead, I added rotations equivalently starting at the shoulder so the entire arm would be affected. After rotating, I edited the translations so that the arms attached at a correct looking location on the cube man. Hence this crazy looking run.

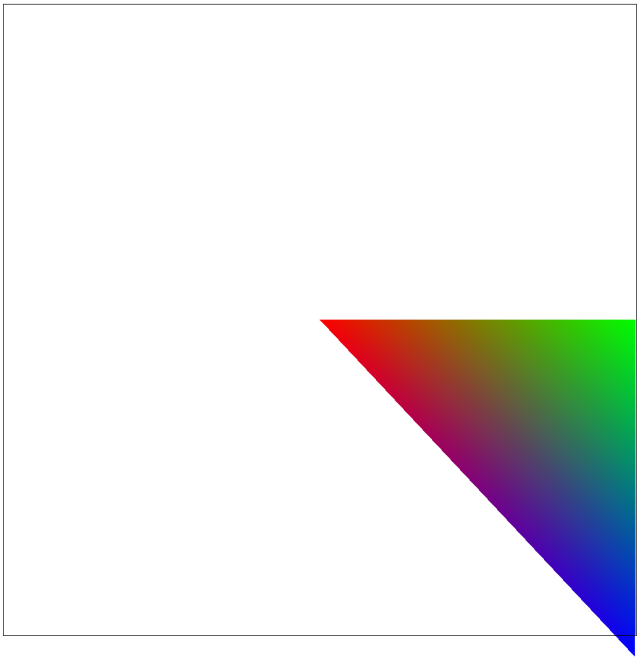
As a final touch, I decided to modify some colors to give it some swag.



Task 4

Image aid!

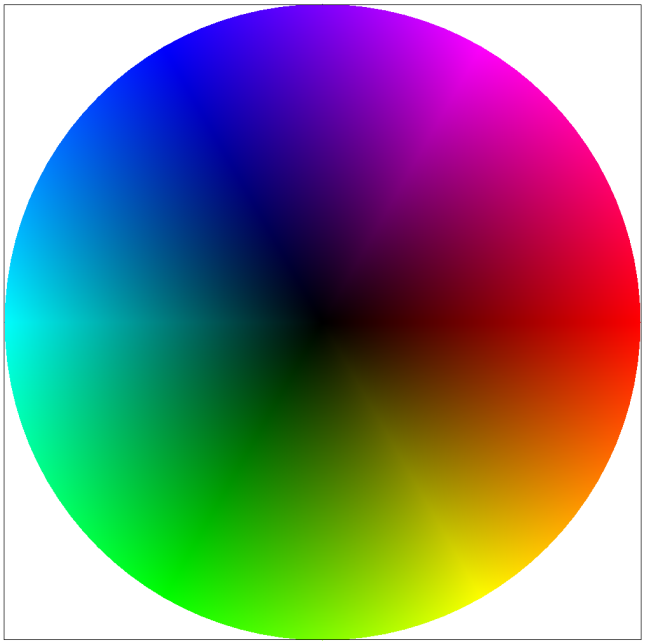
Resolution 1600 x 1200. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.



Framerate: 60 fps

BaryCentric Coordinates can be thought as the the center point of a triangle weighted relative the distance from each Vertex. Take the triangle for example, with vertices colored Red, Green, and Blue. You can see that the colors blend and transition throughout the triangle. There are no borders and the colors are smooth. This is becuase every pixel in the triangle is calculated via it's relative distance to the vertices and the colors corresponding are weighted repsectively. In other words, we can represent eaxh pixel in the triangle as the weighted sum of how far each pixel is relative to each vertex.

Resolution 1600 x 1200. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.



Framerate: 56 fps

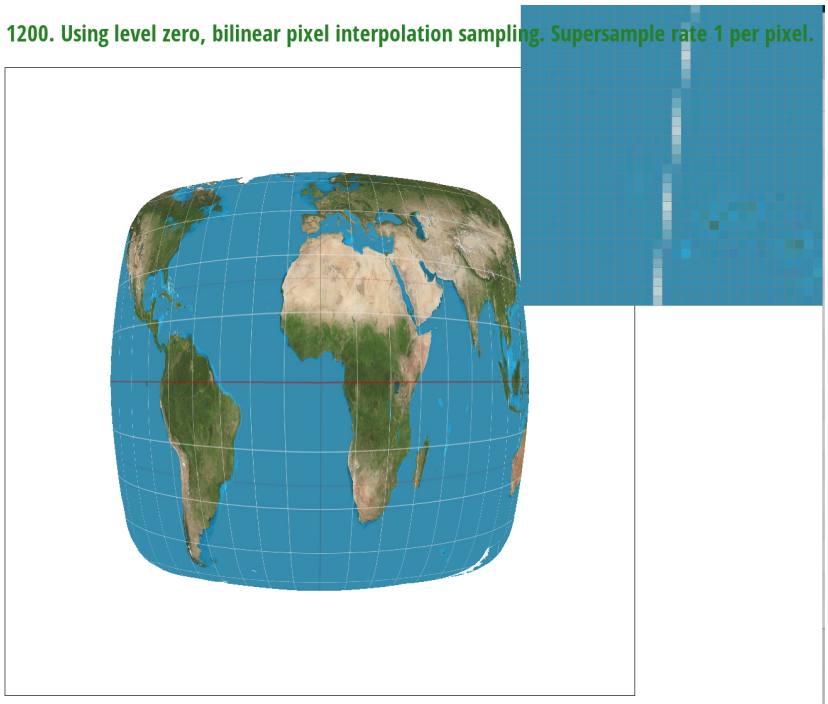
Task 5

Pixel Sampling is the strategy used to convert real-time continuous data into something discrete - in the world of graphics, that's sampling continuous data logged as discrete data that can be rendered by pixels. So we are sampling the world to determine what color/data eaxh pixel can render.

In Task 5, 2 methods were implemented: Sample Nearest and Sample Bilinear. Sample Nearest returned the colors of the nearest texel in the texture mapping whereas Bilinear Samplng used interpolation to compute the texel to index at. Both methods required texture mapping coordinates (u,v) which were computed same as in task 3 with Barycentric coordinates.

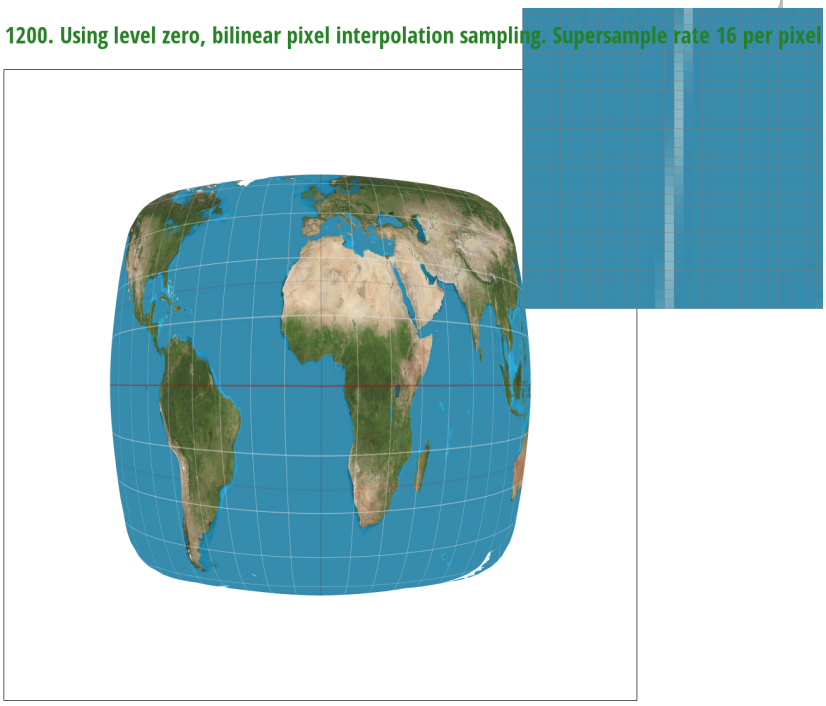
Bilinear Sample Rate 1 vs 16

Resolution 1600 x 1200. Using level zero, bilinear pixel interpolation sampling. Supersample rate 1 per pixel.



Framerate: 58 fps

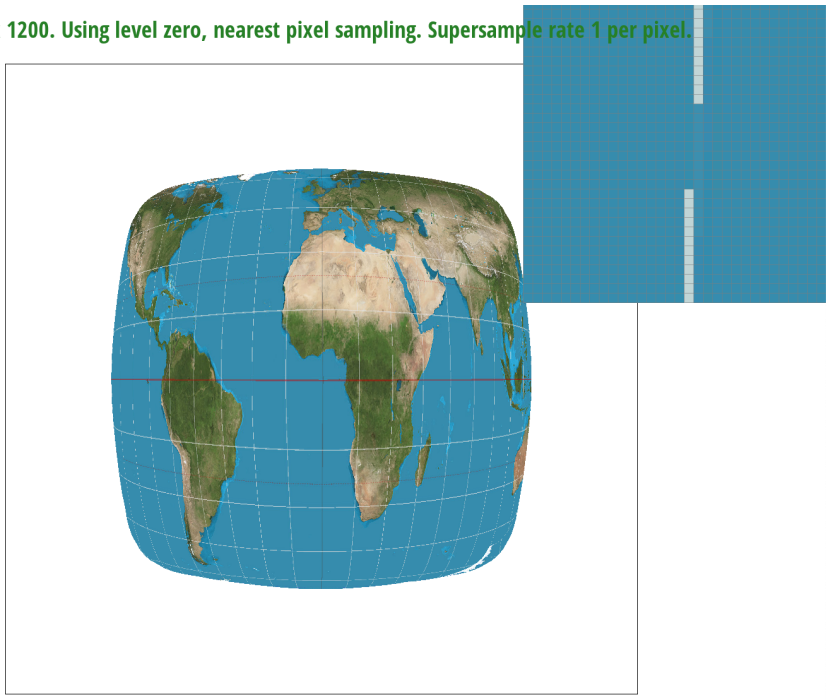
Resolution 1600 x 1200. Using level zero, bilinear pixel interpolation sampling. Supersample rate 16 per pixel



Framerate: 53 fps

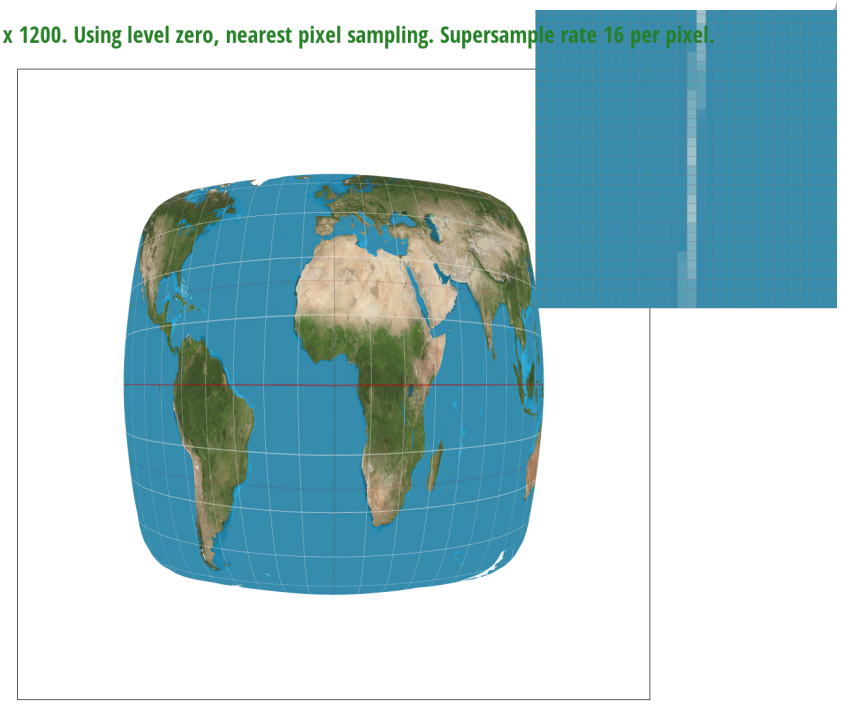
Nearest Pixel Sample Rate 1 vs 16

Resolution 1600 x 1200. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.



Framerate: 51 fps

Resolution 1600 x 1200. Using level zero, nearest pixel sampling. Supersample rate 16 per pixel.



Framerate: 45 fps

Differences Analysis

Overall, it's visually obvious that supersampling improves the image output from both pixel techniques. Overall, Bilinear sampling can be described to have a blurrier out come while nearest Pixel sampling can produces very pixelated images with jaggies. In my case, an incomplete line because of how pixelated the image turned out. In this case, the differences are very noticeable without the pixel inspector. But I think in instances of smaller images, situations where the pixels are more obvious, you can see a visual differnece. From a computation standpoint, Bilinear Interpolation requires more math per pixel so it could be computationally heavier to use Bilinear sampling if the image were really big. And a larger image means more pixels so using nearest sampling wouldn't be the worst because it would be more difficult to see the jaggies since most ppl won't be looking at the image up close or able to see the detail.

Task 6

Level Sampling and My Implementation

Level Sampling is generating an image from sampling to render at a level. The higher the level, the more fuxxy/less detailed the image is. The application of this is to emphaize depth of field and mimic how the human eye percieves the world - specifically farther objects are less detailed while closer objects are more detailed. Having certain amounts of detail puts logic into our image so that it can "make sense."

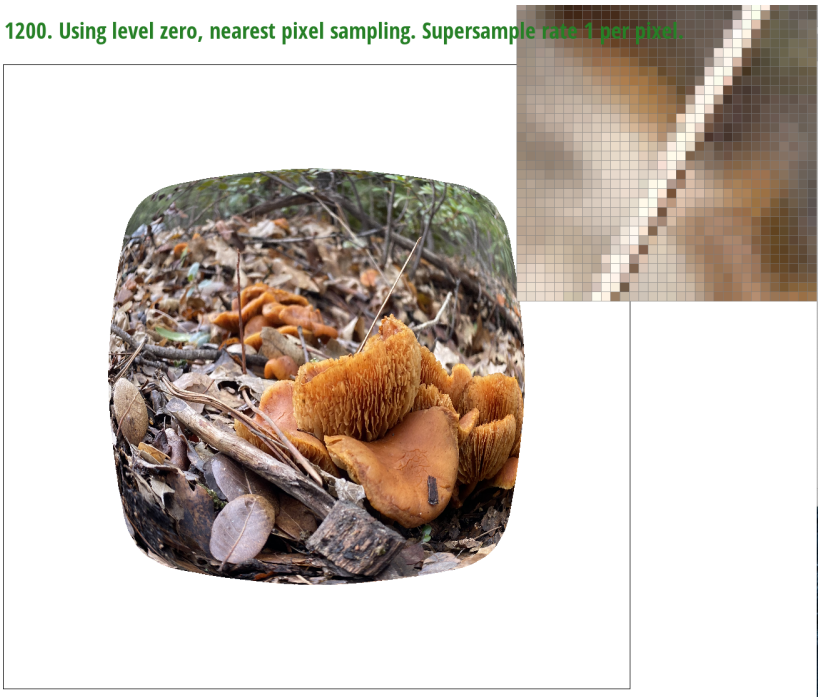
I implemented texture mapping through barycentric coordinates where u,v (texture mapping coordinates) are determined according to the barycentric coordinate weights from the sample buffer vertices. So I calculated alpha, beta, and gamme and calculated a weighted avg of (u0, v0), (u1, v1), and (u2, v2) to find (u,v). Given some psm and lsm, the sample helper function would accordingly sample by nearest pixel or bilinearly. Within sample_nearest and sample_bilinearly, the calculated level was used to index into the mipmap to find the correct mipmap texture - if the level were too high, I clamped it so that it would index mipmap[size - 1]. Additionally, I calculated barycentric coordinates for (x + 1), y and x, (y + 1) becuase those were used to calculated the levels of each triangle.

TradeOffs

Overall, Each sampling method had their strengths and weaknesses. Nearest neighbor sampling would be the least computational and require less memory usage because it finds the closest texel to utilize. However, Level sampling and supersamplng produve more accurate images though it'll take longer and require more memory. Supersampling requires the most most memory and will be the most computationally heavy since it is averaging out each color to the same degree whereas Level sampling only cares about specific pixels since some will be less detailed. The less detailed pixels would require less computation and memory since including the excess information in the image could produce an image less desirable image. So level sampling saves more space than super sampling but still requires more than nearest neighbor sampling. It ranks the same in computation speeds since memory corresponds with the amount of data being computed.

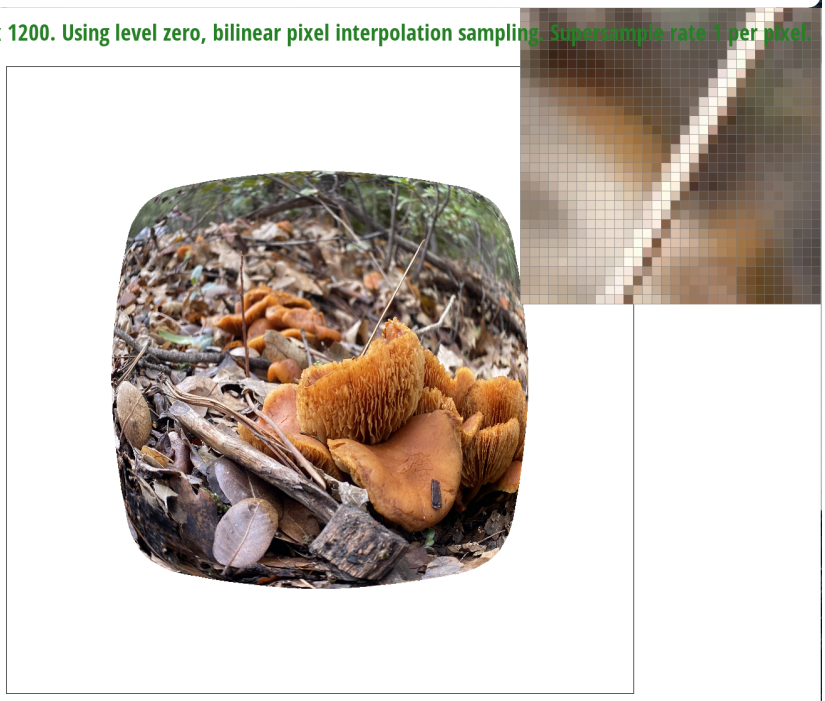
Level Zero Mip Map with Nearest Neighbor vs Bilinear Sampling

Resolution 1600 x 1200. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.



Framerate: 58 fps

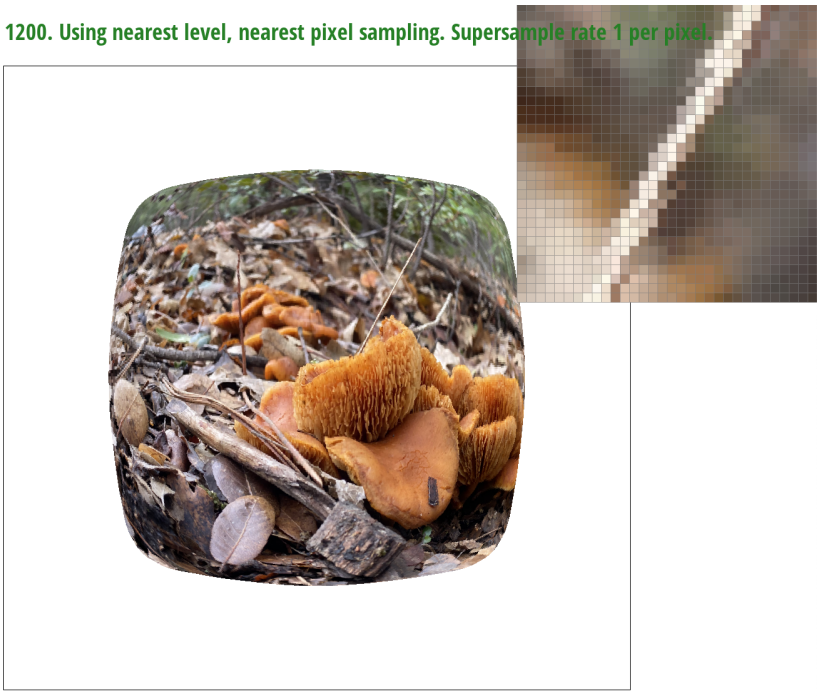
Resolution 1600 x 1200. Using level zero, bilinear pixel interpolation sampling. Supersample rate 1 per pixel.



Framerate: 53 fps

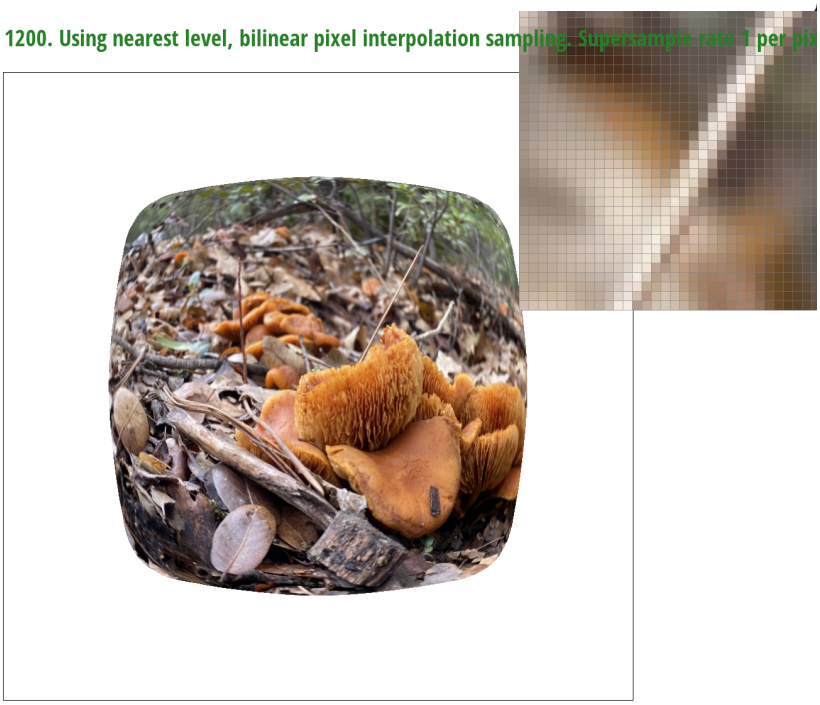
Leveled Mip Map with Nearest Neighbor vs Bilinear Sampling

Resolution 1600 x 1200. Using nearest level, nearest pixel sampling. Supersample rate 1 per pixel.



Framerate: 56 fps

Resolution 1600 x 1200. Using nearest level, bilinear pixel interpolation sampling. Supersample rate 1 per pix



Framerate: 48 fps