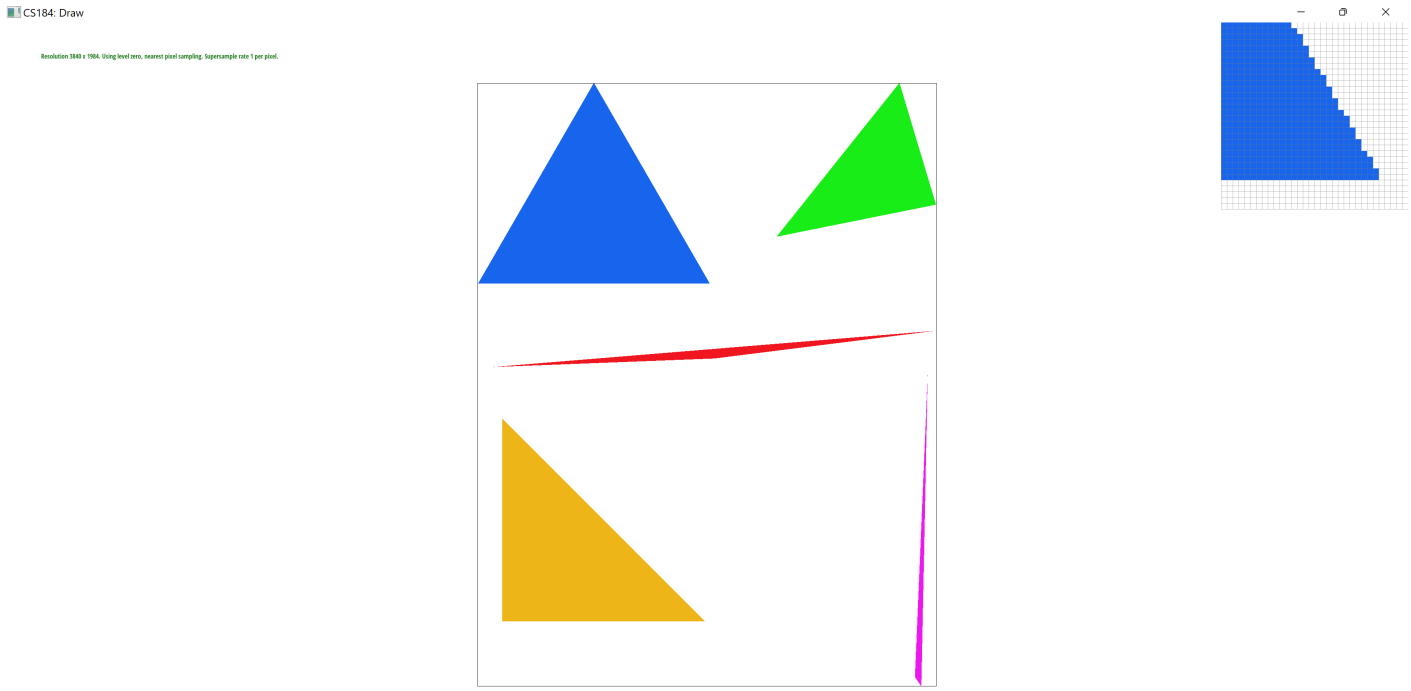
Project 1: Rasterizer

Raiymbek Akshulakov
Yersultan Sapar

Overview

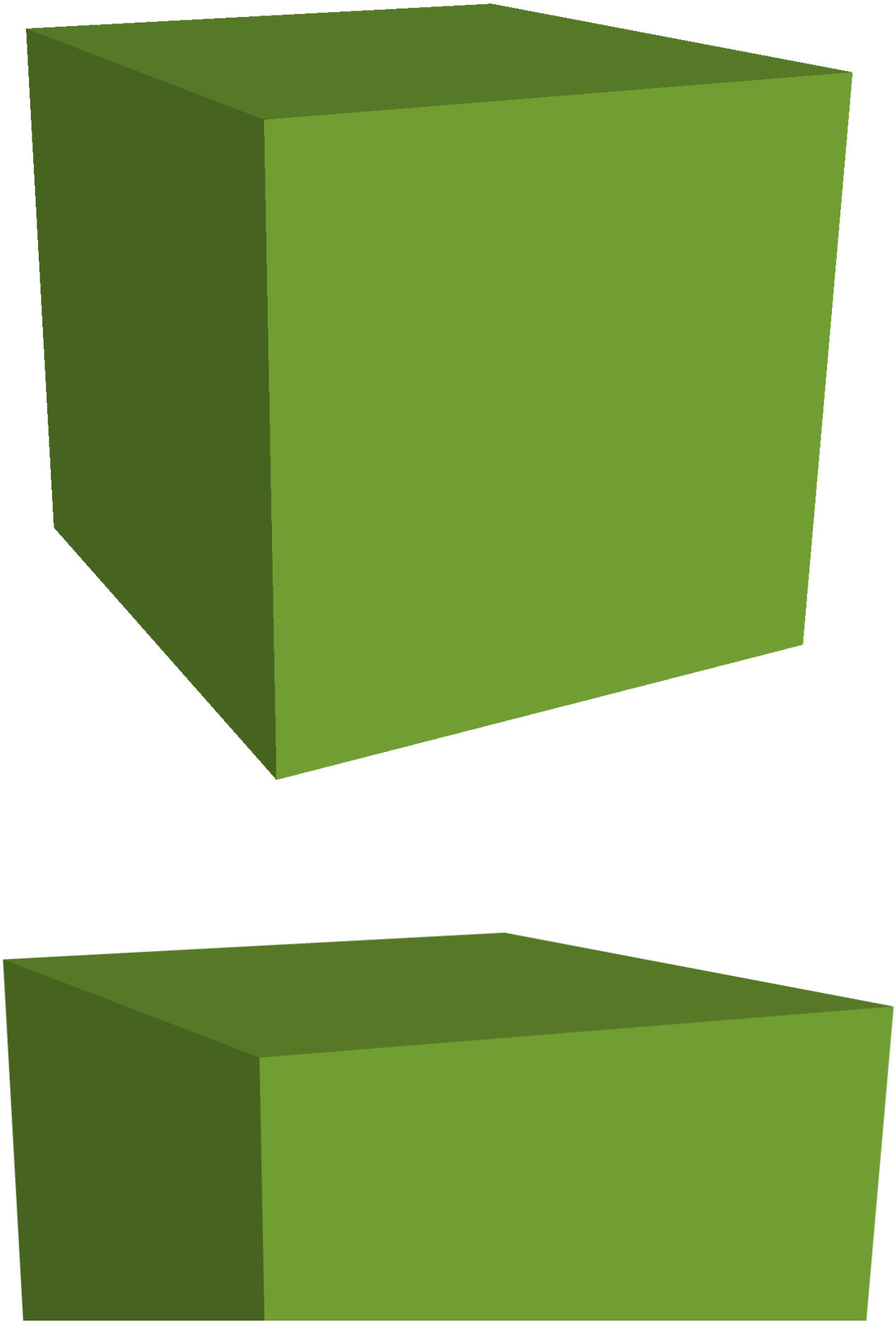
In this project, we have created the Rasterizer engine that parses svg files and uses primitives (like triangles) to render the image. We have also implemented additional optimizations like supersampling, mipmap level sampling to mitigate aliasing effects.

Task 1



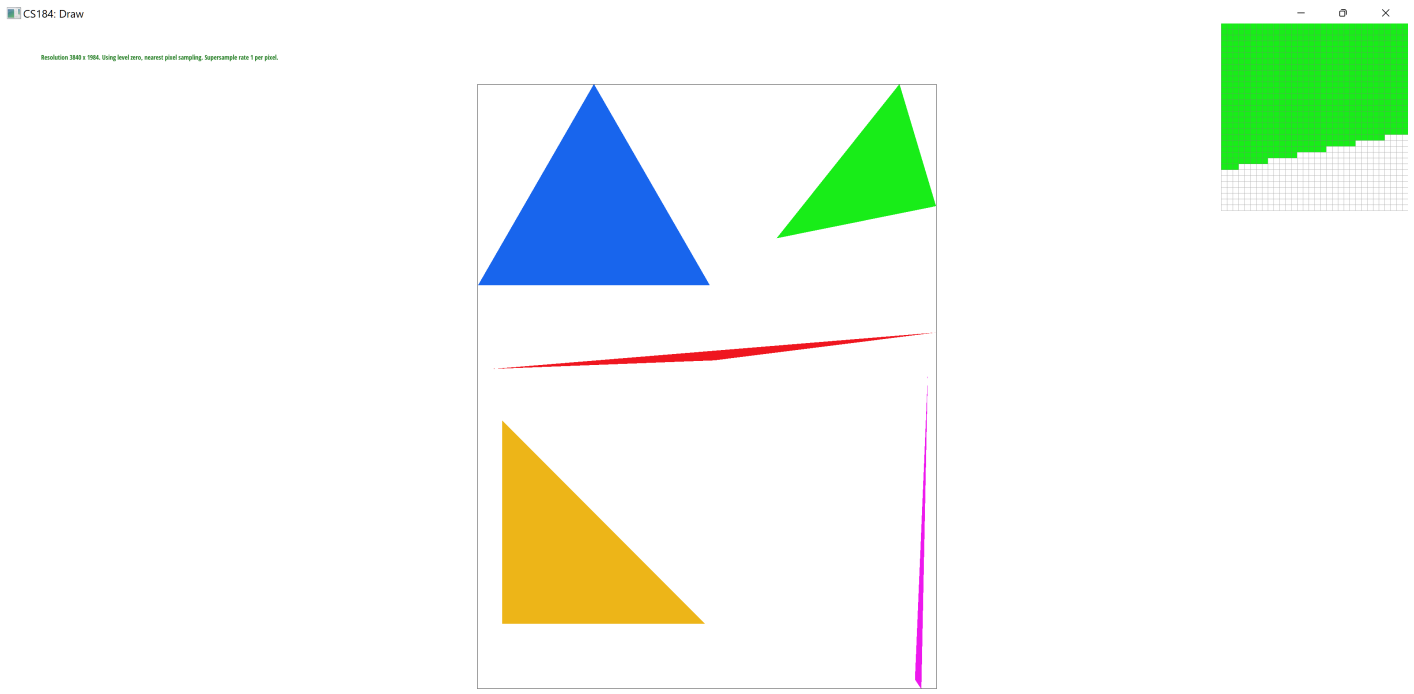
To rasterize the triangle we first need tringle itself - three float vertices. Here is the problem which pixels should be colored as part of the tringle and which are should not. Pixels are only on integer coordinates, we can just give float point o tit to color. To finally get the edges of the triangle and understand which pixels are part of the triangle (in other words rasterize), we need to go over all pixels in the bounding box and genetically check if the center of a pixel is inside of the triangle. If the pixel center is inside then it is thought of as part of the tringle.

Task 2

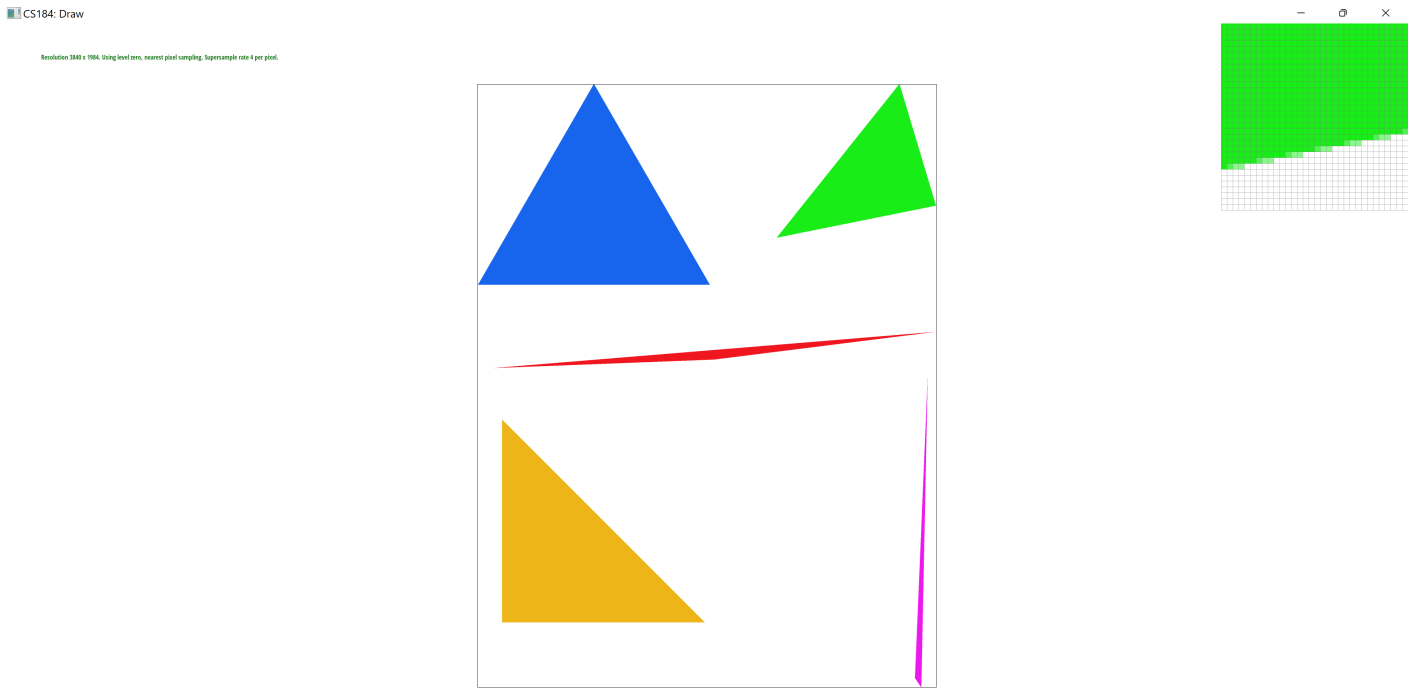


Sampling Rate of 1 vs. Sampling Rate of 9

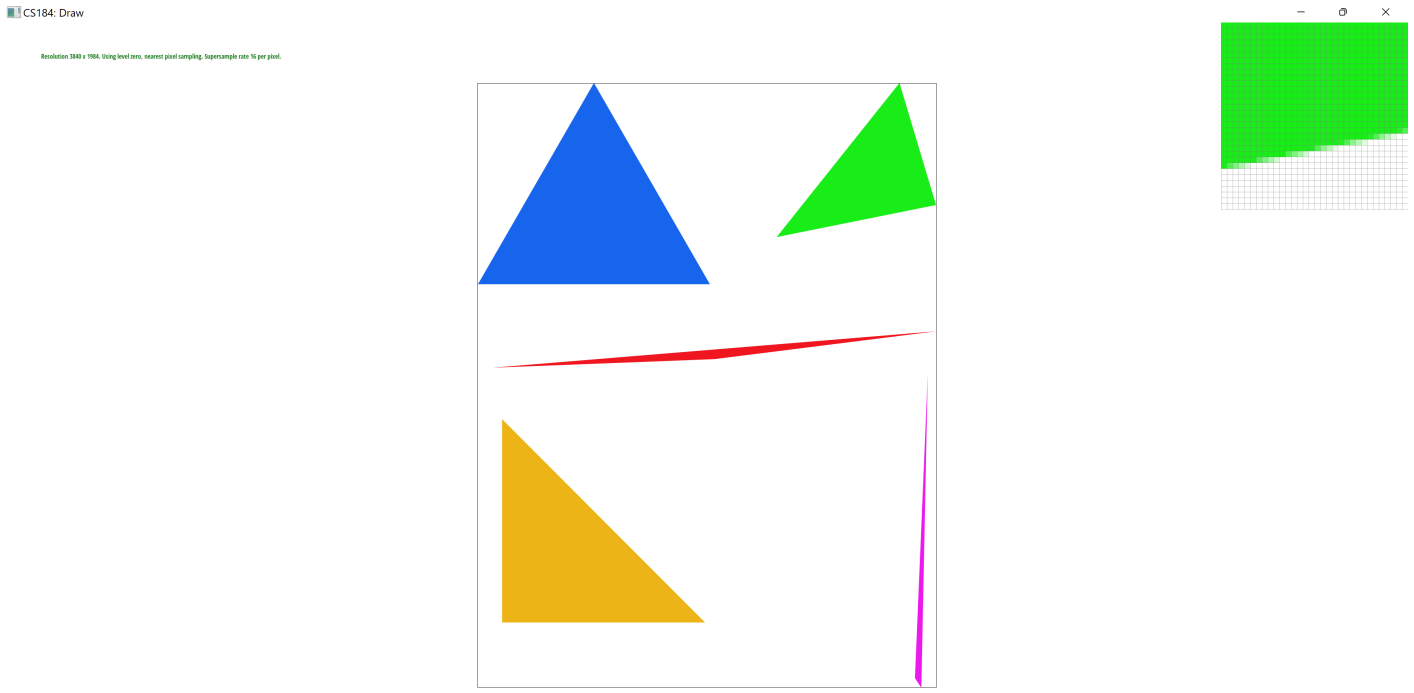
If you look at this image even from far away it is noticeable how the edges do not really appear straight. It is because even though pixel on the edges is technically in the triangle, but the structure of the pixels does not allow for the actually straight not horizontal line. One way to get rid of it and make the edges look straight is to filter each by point based on how much of this pixel is inside of the triangle. One way to simulate this filtering is to imagine several subpixels inside each pixel. We are going to assign each subpixel a color based on in which tringle its center is. In the end, for each pixel we are just going to get an average of those colors over subpixels. Here are results for 1/4/16 subpixels for one pixel:



Sampling Rate 1



Sampling Rate 4

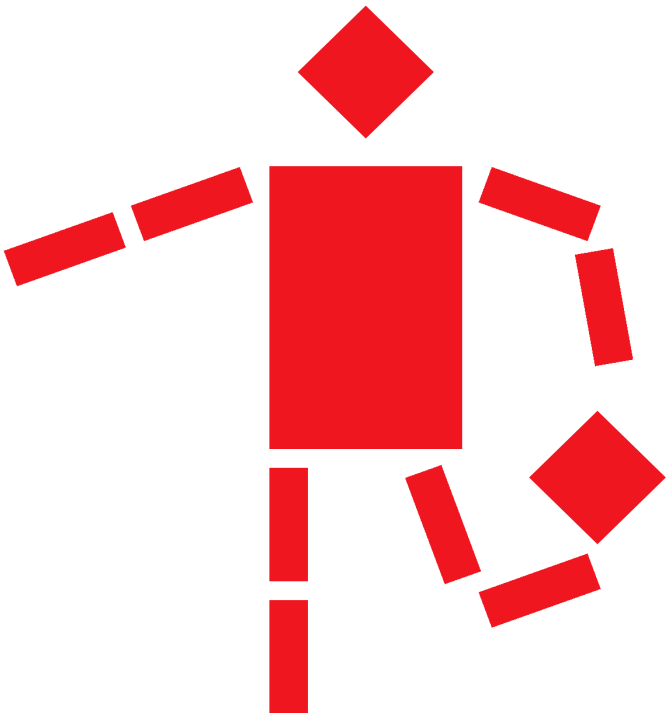


Sampling Rate 16

As you can see in images above as we increase the sampling rate the triangle border becomes more and more smooth as pixels near the border are closer to white color and lines become smoother from far

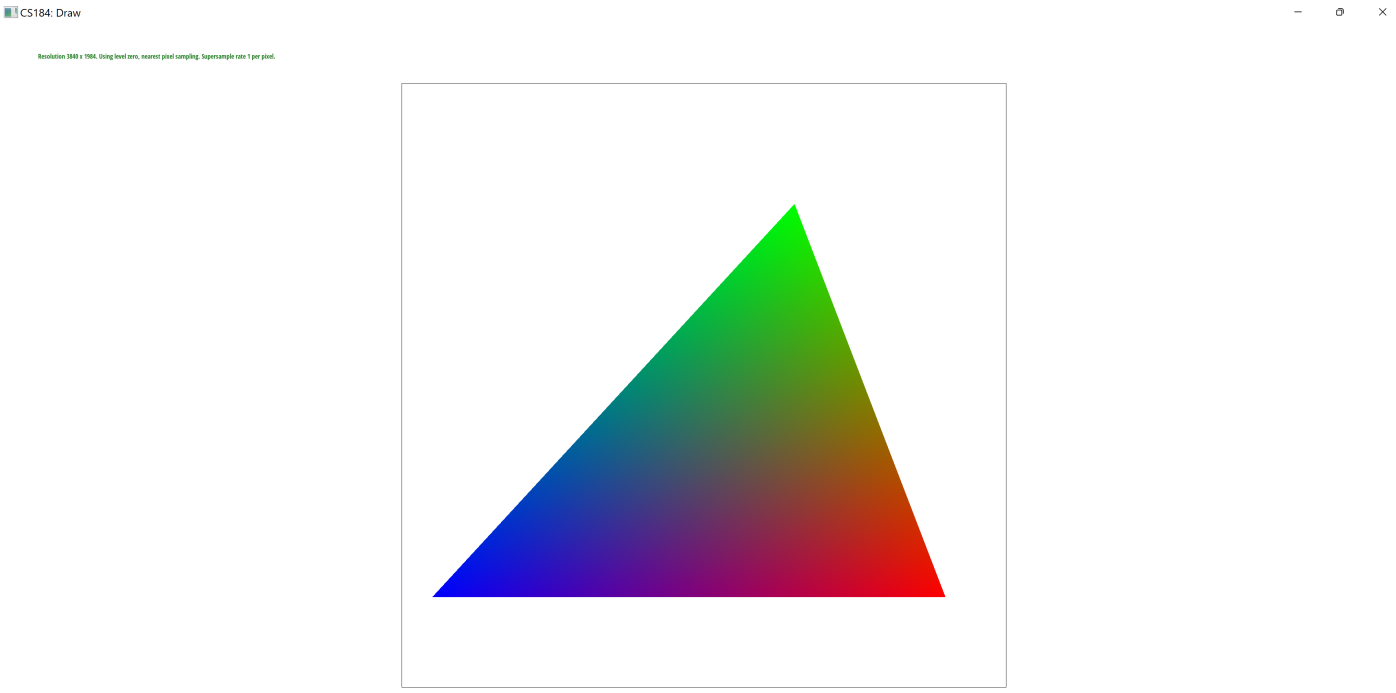
Task 3

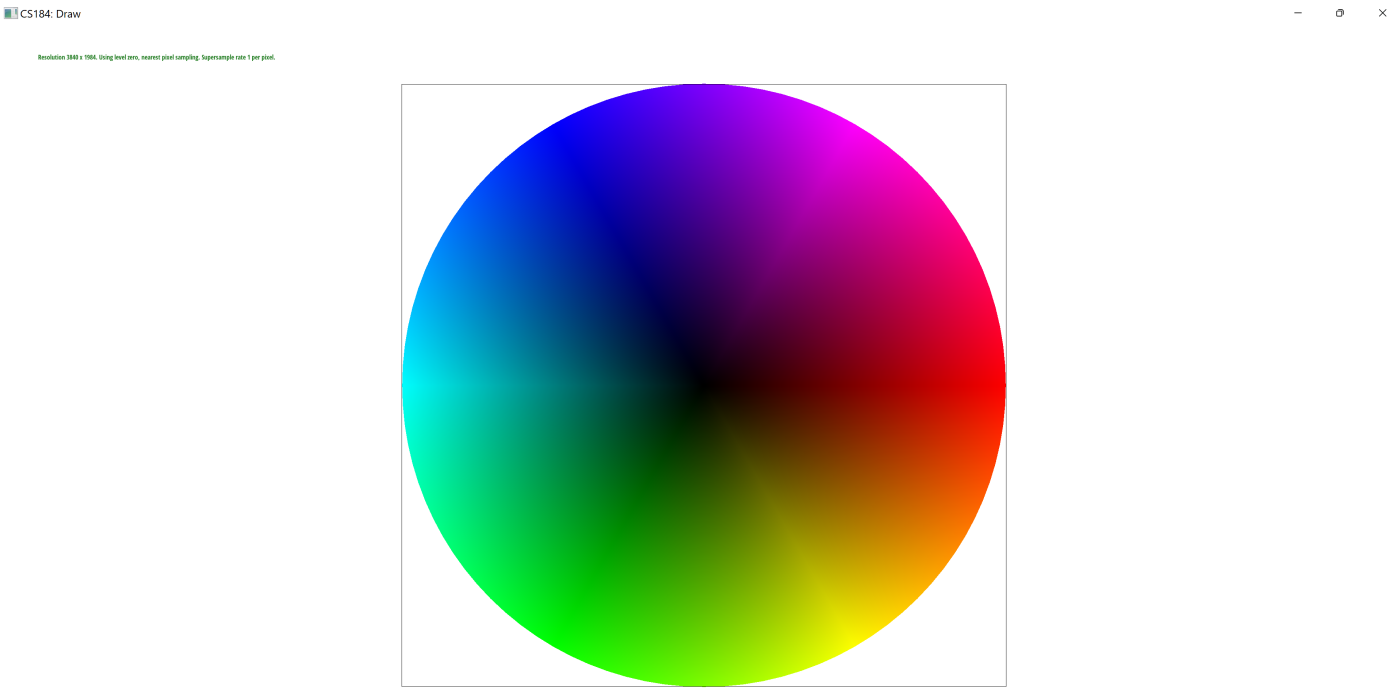
For this my_robot I tried to make him do something similar to what a soccer player does here so I added a ball and added some rotation on the leg to look like he also kicked it with his backheel. Also just added some rotation to the hands to look a bit more natural



Task 4

Barycentric coordinates are triples of numbers (t1,t2,t3) that define a point in a 2d plane based on its relation to the triangle vertices. It is like we put masses at the vertices and those coordinates define the distribution of the masses in each point in the 2d plane. Another metaphor would be if we put different liquid colors at each of the vertices and this liquid got distributed from those points. So each point can be defined by the percentage of each color in there. This way we can visualize it by mixing color in each point in a triangle based on the barycentric percentages.

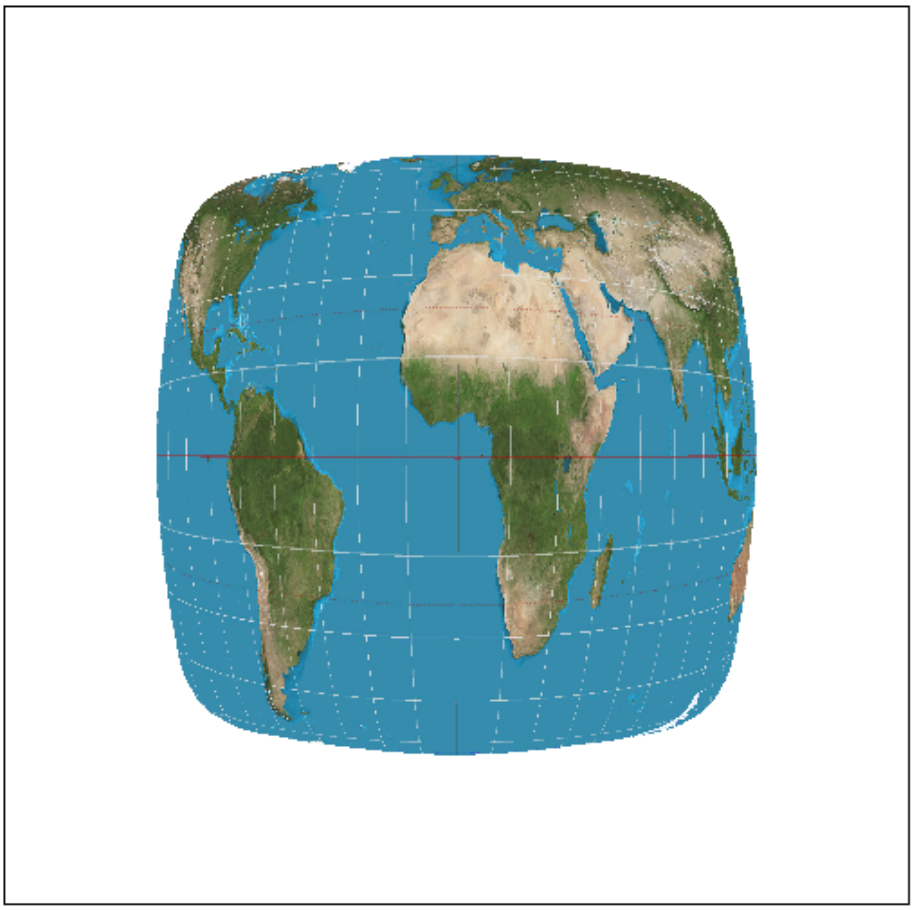




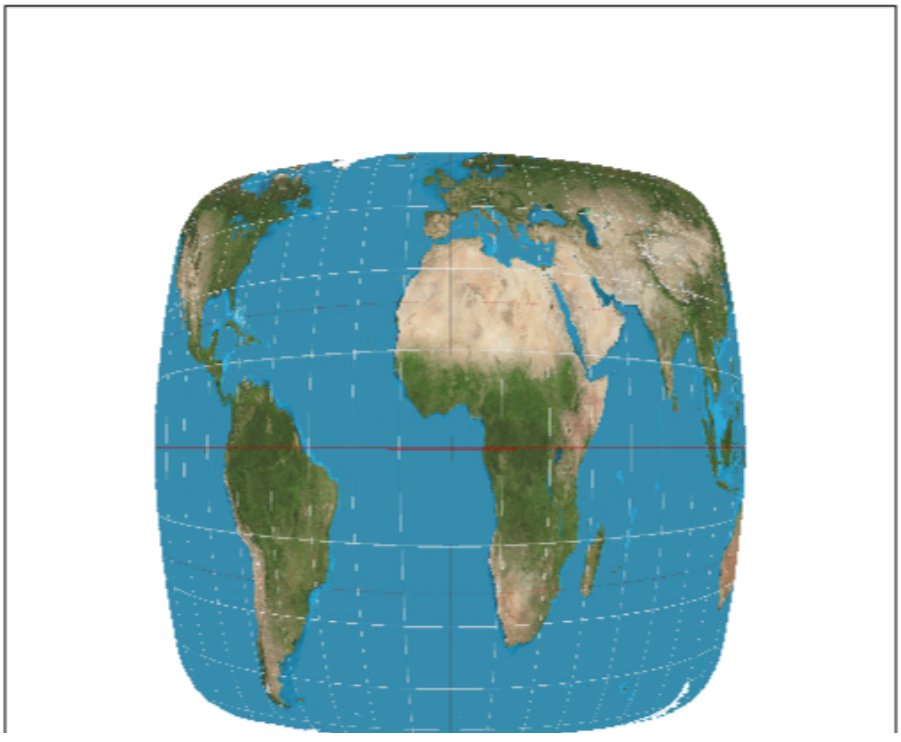
Task 5

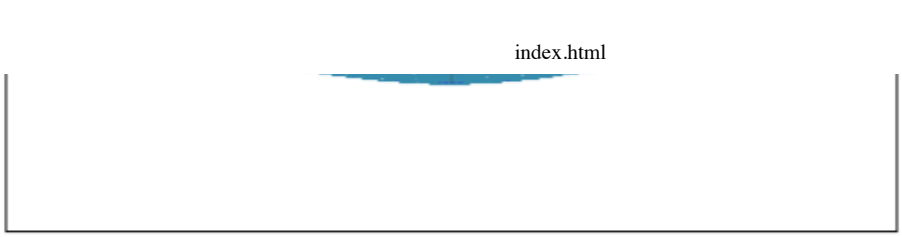
In this problem, we need to sample pixel color in our scene triangle given the corresponding triangle in the texture. We build a correspondence point in the texture space by taking the point with the same barycentric coordinates. However, we still need to know the color of the point since the texture space is discrete but our point is continuous. There are two ways to get the color of the continuous point that we are going to explore - nearest neighbors and bilinear. In nearest neighbors, we are finding the nearest discrete point and simply taking its color. In the bilinear case, we are finding four nearest discrete locations, and similar to barycentric coordinates we are trying to define the point in the rectangle as the sum of weighted rectangle vertices. Once we define it the color we need is just the weighted sum of the vertice colors.

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



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





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

Resolution 800 x 600. Using level zero, bilinear pixel interpolation sampling. Supersample rate 16 per pixel.

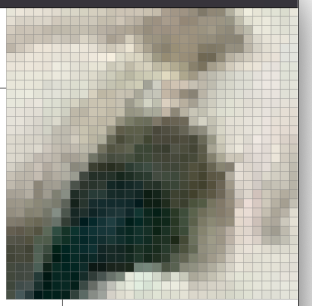
Task 6

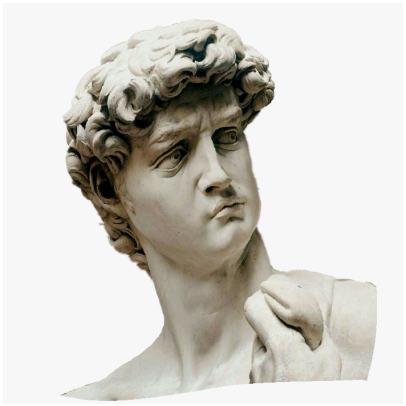
- **Explain level sampling in your own words and describe how you implemented it for texture mapping.**
  - Instead of sampling directly from the full texture, level sampling approach builds up a hierarchy of levels of the same texture, where each level stores the same texture, but in lower resolution. Sampling from the appropriate level with downfiltered textures drastically improves the performance, comparing to the naive sampling across all texels.
  - The implementation had three options for the level sampling. The "Level Zero" is equivalent to having no mipmap, so we

just sample directly from the full resolution texture using the appropriate method (nearest or bilinear).
    - With "Nearest Level", we calculated the distance difference to the next (x + 1, y) and (x, y + 1) pixels on texture coordinates, and used the log2 value of the farthest one to calculate the level.
    - "Linear Leveling" is a method to smooth out the "jumps" of the previous method by using a continuous value for the level. We sample on two neighboring levels and interpolate over that colors to achieve smoother transitions across the texels.
- **You can now adjust your sampling technique by selecting pixel sampling, level sampling, or the number of samples per pixel. Describe the tradeoffs between speed, memory usage, and antialiasing power between the three various techniques.**
    - The supersampling by nature presents a huge trade-off in performance, while mitigating the aliasing effect. Has no effect on memory, since we do not really store anything.
    - The obvious tradeoff of the level sampling is memory usage, since we have to store the same texture in different resolutions across the levels. Constrasting to the speed trade-off of the supersampling, we can say that it is more feasible to use mipmap technique in terms of antialiasing power.
    - Pixel sampling is the level sampling on the level 0, so it does not involve any extra memory usage. However, without the help of supersampling, it probably will not help much with aliasing artifacts.
- **Using a png file you find yourself, show us four versions of the image, using the combinations of L_ZERO and P_NEAREST, L_ZERO and P_LINEAR, L_NEAREST and P_NEAREST, as well as L_NEAREST and P_LINEAR**

CS184: Draw

Resolution 2234 x 1686. Using level zero, nearest pixel sampling. Supersample rate 1 per pixel.

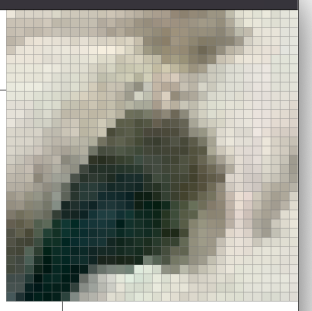


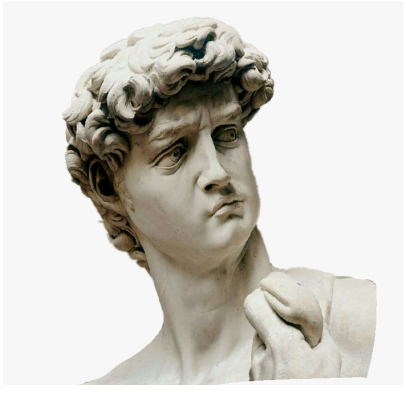


Framerate: 52 fps

CS184: Draw

Resolution 2234 x 1686. Using level zero, bilinear pixel interpolation sampling. Supersample rate 1 per pixel.

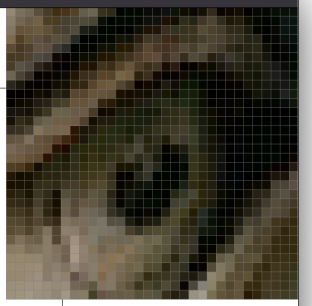


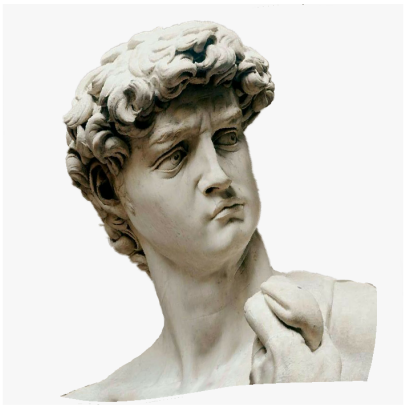


Framerate: 49 fps

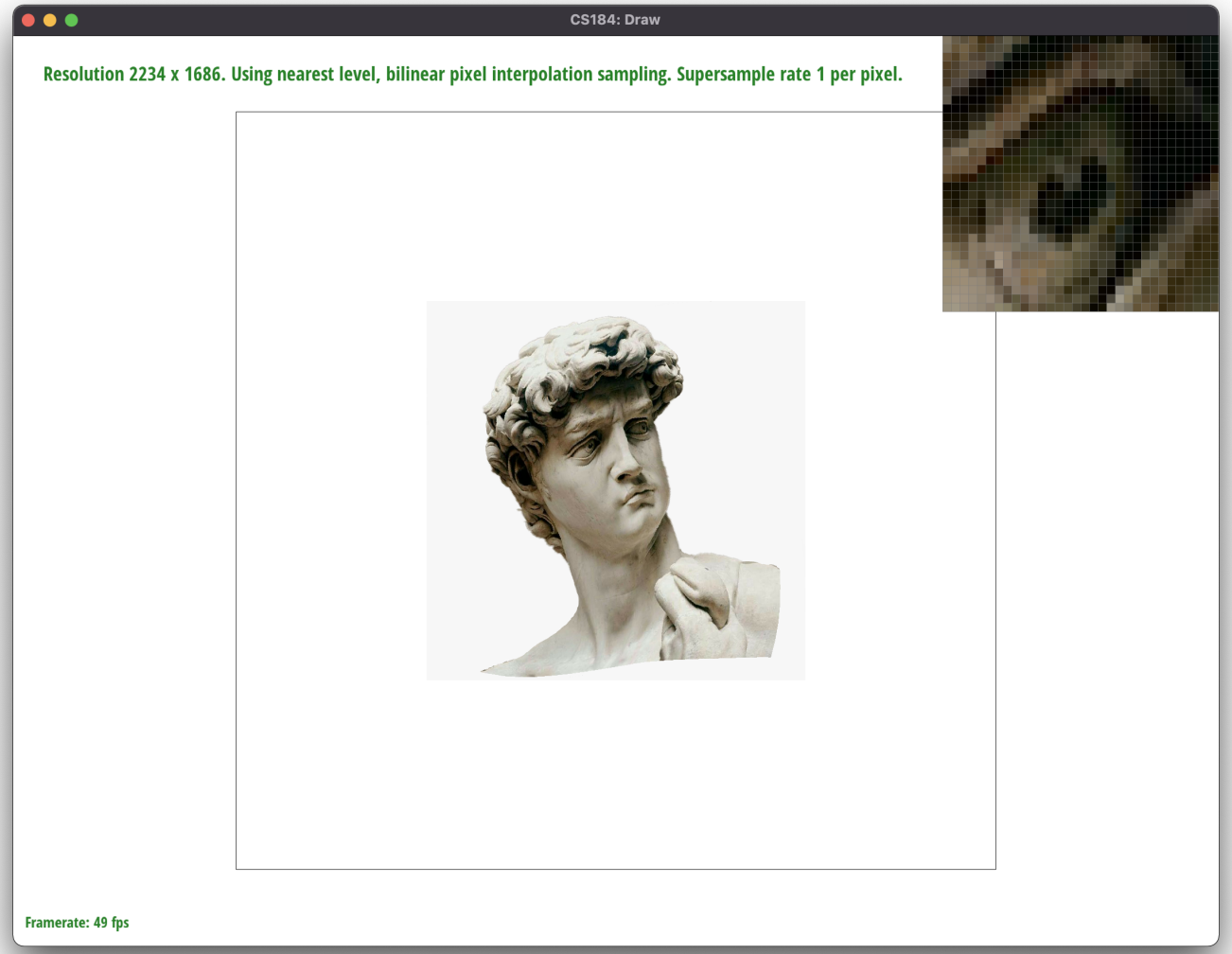
CS184: Draw

Resolution 2234 x 1686. Using nearest level, nearest pixel sampling. Supersample rate 1 per pixel.





Framerate: 52 fps



The website link is https://cal-cs184-student.github.io/sp22-project-webpages-yersultan-17/