CS 184: Computer Graphics and Imaging, Spring 2019

Project 1: Rasterizer

Shenao Zhang, CS184

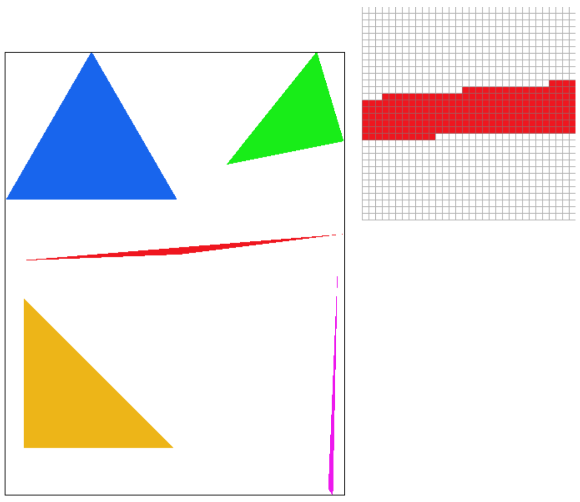
SID:3034487184

In this project(project 1), what I implemented include two parts: rasterization and sampling.
In the first part, I implemented a rasterizer that takes a .svg file as input and rasterizes the corresponding image in screen space. First step is to rasterize single-color triangles which uses test function to test whether a point is inside the triangle. Then we get rasterized triangles with jaggies. Then we can use supersample to antialias triangles, which sample multiple points per pixel and we can control the sample rate. Next, I implemented the matrix for translate, scale and rotate.
When part 1 is done, we need to consider converting texture space to screen space.
In part 2, I first calculate the corresponding barycentric coordinates to futher obtain smoothly varying values across surface. Then, we have two pixel sampling choices: nearest, bilinear interpolation and three mipmap level resampling choices: always level 0, nearest level and linear interpolation. I then implemented these methods to get different images.
Every part when I get the image, regardless of right or wrong, I am excited and feel the powerful magic. Eventually, I get what I expect.

Section I: Rasterization

Part 1: Rasterizing single-color triangles

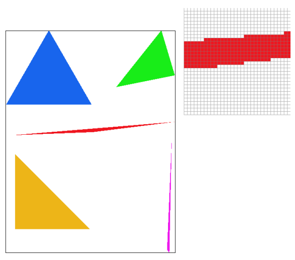
For triangle rasterizing, I test whether the middle of a pixel is in the triangle. First I set the color of a single sub-pixel sample and then the fill-pixel function can set every subpixel be the same color. Second, I use the vector multiply (the lines of the triangle multiply the vectors of sample point and vertices to find the points inside the triangle). Finally, fill in the color using fill-pixel function if the point is inside the triangle.
Because I first count the minimum and the maximum of the triangle coordinates(both x and y), then I get the bounding box of the triangle. I then test each sample by vector multiplication, getting three results. But when the two of them, one is positive, one is negative, stop computing the third multiplication. So the algorithm is no worse than one that checks each sample within the bounding box of the triangle.



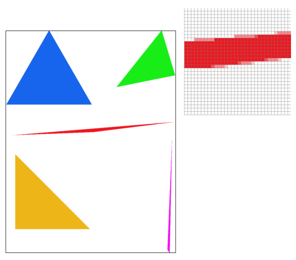
Screenshot for basic/test4.svg with default veiwing parameters.

Part 2: Antialiasing triangles

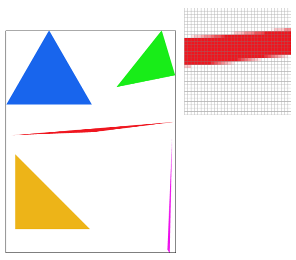
Because sampling only one point per pixel can cause aliasing, we need to sample multiple points in one pixel, called supersampling.
For supersampling, it mainly contains two steps:
1.find the sub-pixel samples inside the triangle.
2.get the average color of the sub-pixel samples(the final output color of that pixel).
When sample rate is larger than 1, supersampling find multiple sub-pixel samples in one pixel which is inside the triangle. When doing this, the jaggies caused by pixels which is near the line of the triangle can be removed by using the average sub-pixel samples. The larger sample rate is, the more accurate point-test is, the average color looks more smooth. I add two iterations for testing sub-pixel samples inside triangles rather than just the center of one pixel. We can control the sample rate to antialias triangles.



Sample rate=1.

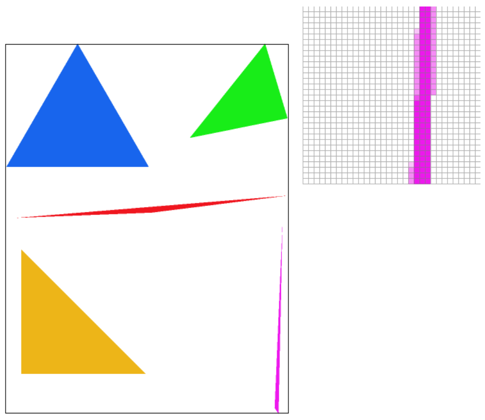


Sample rate=4.

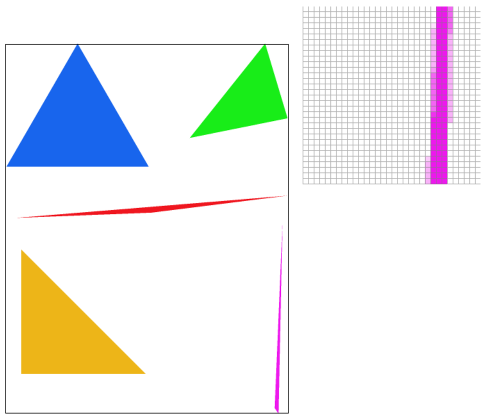


Sample rate=16.

Observing the following three images, we find the larger sample rate is, the smoother edges are. This is because when sample rate is large, we can test more sub-pixel points. And the color of the pixel can better represent the relationship between the pixel and edges of triangles rather than one pixel is(or isn't)in the triangle when sample rate is 1. So as we see, the image seems to be more smooth.
Extra credit: Except the supersampling, we still have a lot of sample methods. Here, I implement the jittered sampling. Jittered sampling is a stochastic process in which values are sampled uniformly over a rectilinear subspace. The exact position of the respective sample in each subrectangle is thereby varied randomly. And the below results show it has better performance.



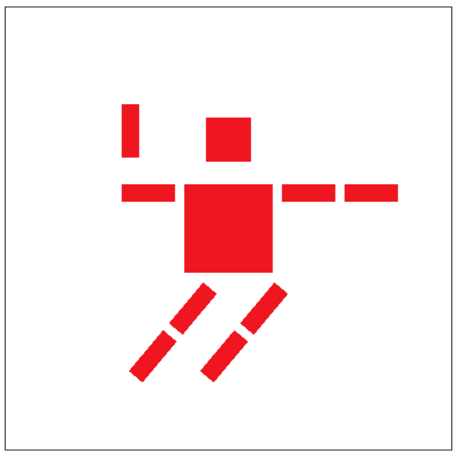
Super sampling(sample rate=4).



Jittered sampling(sample rate=4).

Part 3: Transforms

I create a robot who is jumping and trying to slam dunk.
I first complete the functions in transform.cpp, which include translate,scale and rotate. Then I use these functions to create my own robot. I creat him by rotate both his left and right legs, and rotate his left arm, then translate it. It looks like he is jumping and trying to slam dunk.
The image is as below.

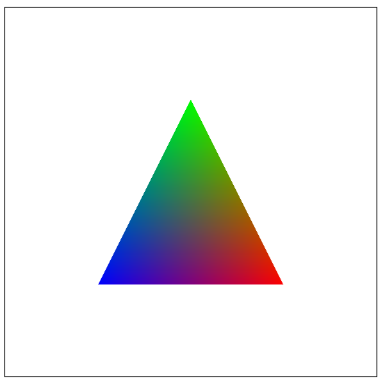


My robot who is trying to slam dunk

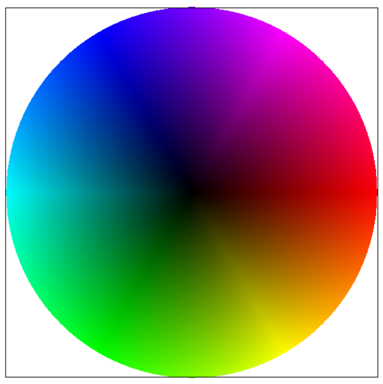
Section II: Sampling

Part 4: Barycentric coordinates

In a triangle, barycentric coordinates represent the distance between a point inside the triangle and its vertices. Then we can use the barycentric coordinates and the color of vertices to calculate the color of its color. For example, we have a triangle with one red, one green, and one blue vertex. Then around the blue vertex, the corresponding barycentric coordinate is around 1. We can observe the color around the blue vertex is also blue(1*blue.color). And in the center of this triangle, the three coordinates is equal, so the color is black(1/3*blue.color+1/3*green.color+1/3*red.color).And that is how barycentric coordinates work.
Below is the color gradients triangle and color gradients circle.



A triangle with one red, one green, and one blue vertex.

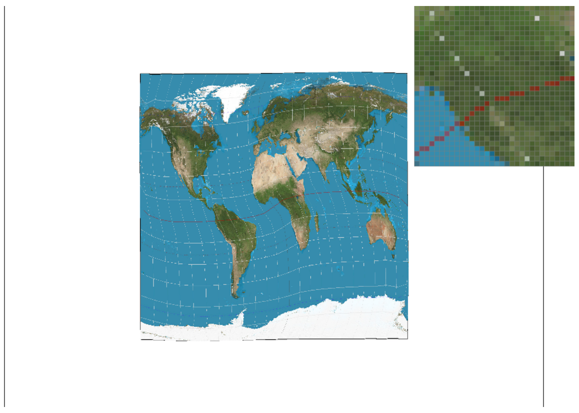


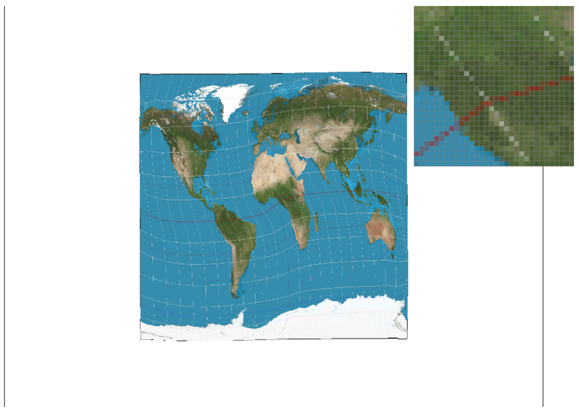
Screenshot of svg/basic/test7.svg with default viewing parameters and sample rate 1.

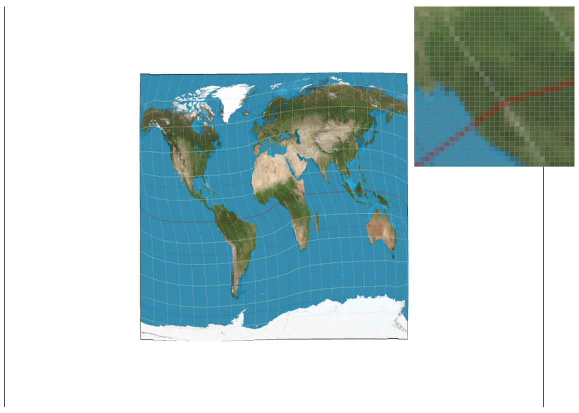
Part 5: "Pixel sampling" for texture mapping

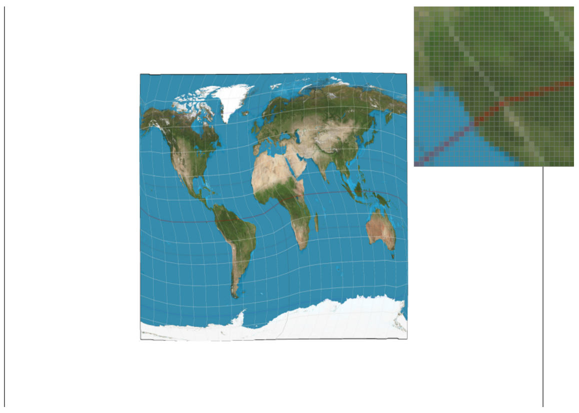
In the texture space, we have nearset sampling and bilinear. The main idea of nearset method is first calculate the corresponding index of texels(Y*w+X, where X,Y is nearest integer of uv.x*w,uv.y*h) and then return the color of this index. It is computational

efficient and easy to implement. But this method can raise aliasing.
So we have another antialiasing method, bilinear method. The color of a point is calculated by four surrounding points. As shown below, we can first use u00,u10 and bilinear function(u00+s*(u10-u00)) to calculate u0, then similarly, we get u1. Using bilinear function, we can get the color of the red point. It is a good method for continuous image but sometimes it results in blur in the image and it needs more computation.


Screenshot for texmap/test2.svg with nearest sampling, sample rate = 1.


Screenshot for texmap/test2.svg with bilinear sampling, sample rate = 1.

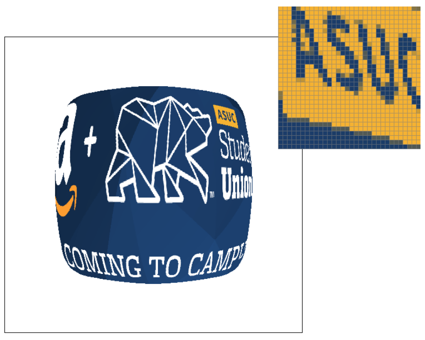

Screenshot for texmap/test2.svg with nearest sampling, sample rate = 16.

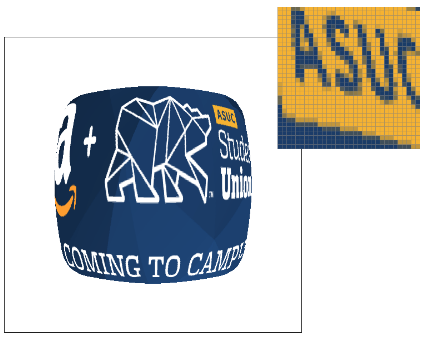

Screenshot for texmap/test2.svg with bilinear sampling, sample rate = 16.

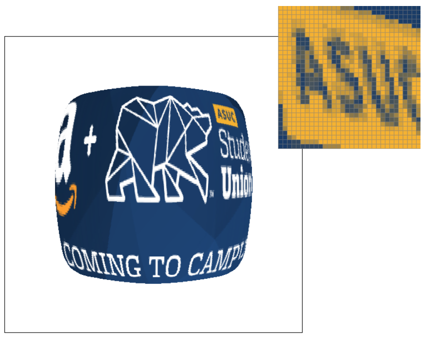
When sample rate is low, which means the image is aliasing, using bilinear can get efficient texture antialiasing. When sample rate is 16, we already use supersample for antialiasing. So now the nearset sampling and bilinear sampling are nearly the same. Both bilinear sampling and supersampleing are for antialiasing and can provide a more smooth color at the edge, although supersampling is costly.

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

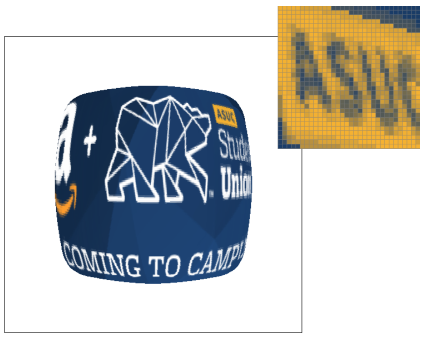
When a pixel cover an area much larger or smaller than a single texel, aliasing or bluring are often visible when textures are actually rendered. So we can store prefiltered textures in the form of a mipmap to approximate the correct result and reduce aliasing. We use mipmap in to store the texture map at different level, where level 0 is the original texture scale. Every level up creates a downsampled scale.For example, when a big object is far away, our sample rate should be high, which results in aliasing. So we can downsample the texture space, i.e.use the high level mipmap.
To calculate the level, the get_level function calculates the mipmap level using uv coordinates. uv coordinates are then scaled by width and height of the texture.
When zooming in the image, bilinear method provides a more smooth color at the edge. But if the image is low resolution, when zooming in more, the bilinear scheme can cause a more blurry visual effect compared to level 0. This is expected since when the level is high, the mipmap of low resolution image contains fewer texels which results in blurry visual effext.
When using image with high resolution, the bilinear scheme can avoid the big jump of colors near the boundaries because linear interpolation can better measure the distance in the screen space and equals to sampling at a proper rate.
The runtime from no level sampling to level sampling increases whatever the sample rate is(1-16). It is expected because level sampling requires more computation of calculating level and mix the mipmap color between levels. There is also an increase in the runtime from sample rate 1 to 16. Because each pixel is now divided into multiple sub-pixel points, which requires more computation and more memory usages. Level sampling can be used to remove the visual artifacts and get a better quality image. But the tradeoff is it takes more time and memory, which is costly.


l=zero,p=nearest


l=zero,p=linear

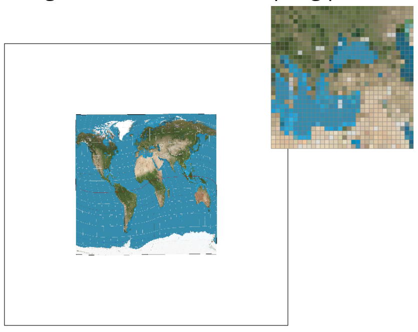


l=nearest,p=nearest

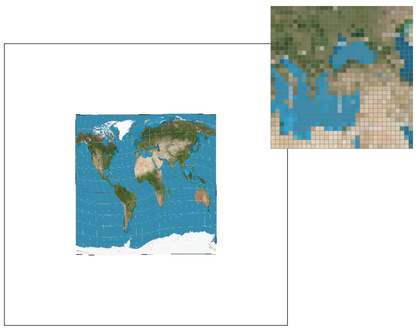


l=nearest,p=linear

We can see that because of the low resolution, level 0 is better than the nearest level resampling. It is analyzed above. As comparison, the below high resolution image, nearest level resampling performs better.

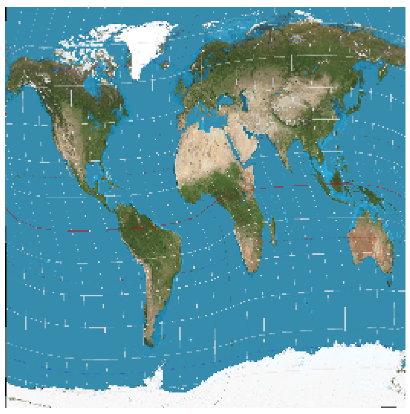


l=zero,p=nearest

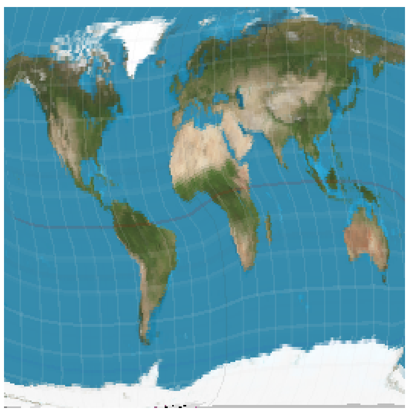


l=zero,p=linear

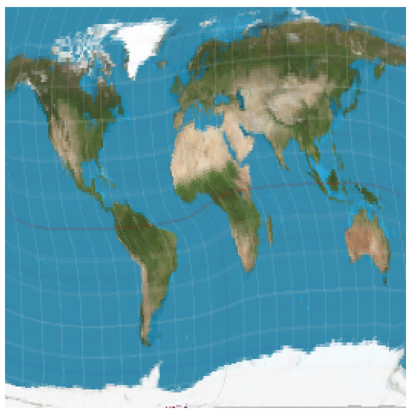
Extra credit: To compare the different level sampling methods, I uses the summed area tables to calculate the sum of values in a rectangular subset of a grid .As the name suggests, the value at any point (x, y) in the summed-area table is the sum of all the pixels above and to the left of (x, y). I generate three images using zero level sampling, nearest level sampling and trilinear level sampling, then calculate these three images' summed area tables, converting the three arrays to images.



zero level sampling

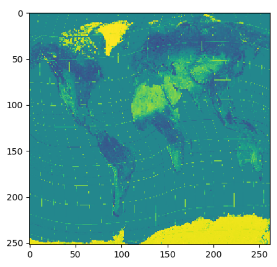


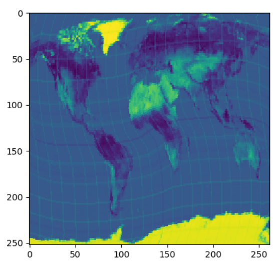
nearest level sampling

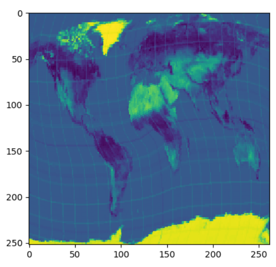


trilinear level sampling

And below is my results of the between there three methods(zero level,nearest level, trilinear level)

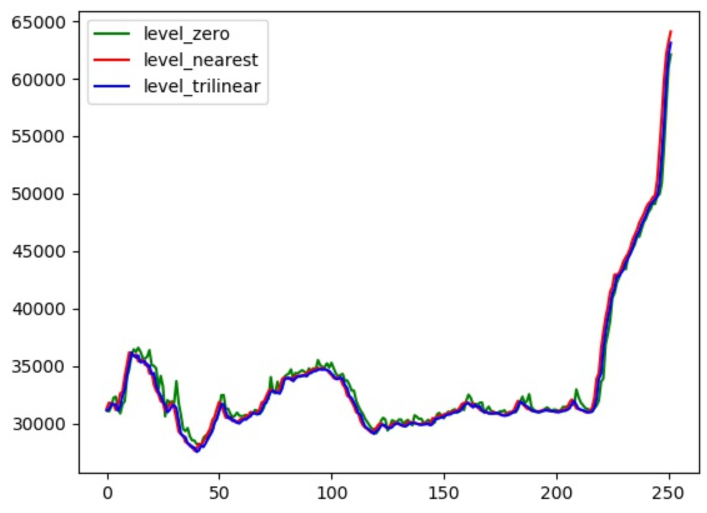






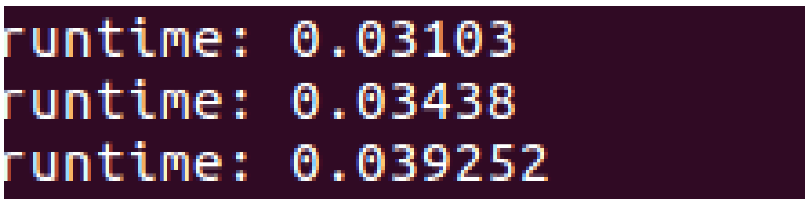
zero level sampling summed area tables nearest level sampling summed area tables trilinear level sampling summed area tables

We can find that there are only slight difference between the nearest level sampling summed area tables and trilinear level sampling summed area tables(2 and 3). To futhere evaluate these three level sampling methods, I use histogram similarity measure which calculates the sum of values each column. And the results are as below. The horizontal axis represents columns of the image, the vertical axis represents sum of pixel values of the image.



Histogram comparison of the three methods.

We can find that there is only slight difference between these three methods.
Then I test the run time of these three methods.



```
runtime: 0.03103
runtime: 0.03438
runtime: 0.039252
```

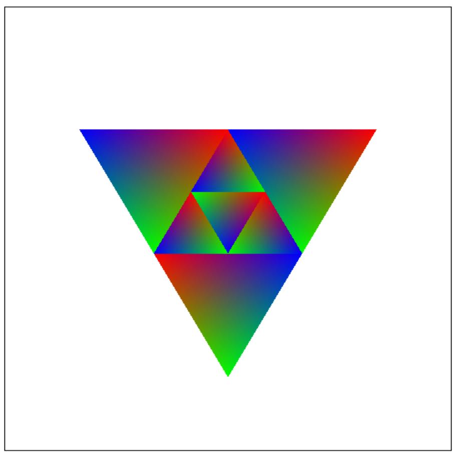
The runtime of the three methods.

We can find that the runtime of these three methods, zero level sampling is the smallest and the trilinear level sampling is the largest because of much more computation. Since the three methods do not have much visual difference for this image(especially nearest and trilinear), we may consider zero level sampling and nearest level sampling because of the smaller runtime.

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!



Hypnosis master!

I first set some simple vertices and their color and interpolate the triangle using barycentric coordinates. Then I get a Hypnosis master! In a Japanese animation "Yu Gi Oh", if you stare at it, you will quickly fall asleep. And with it, the story in the Yu Gi Oh is striking one snag after another.