CS184/284A Spring 2025 Homework 1 Write-Up

Names: Andy Zhang

Link to webpage: https://github.com/cal-cs184-student/hw-webpages-zhangnd16
Link to GitHub repository: https://github.com/cal-cs184-student/sp25-hw1-picasso

Overview

In this project, some common rasterization, texture mapping, and antialiasing methods are implemented and studied. These techniques are frequently used in image and texture processing. One interesting thing I learned in this project is that even simple things like drawing a triangle can be much more complex on the screen due to its discrete displaying range. Computer graphics is a much more deep subject than it might appear, with a lot of intricate implementation details.

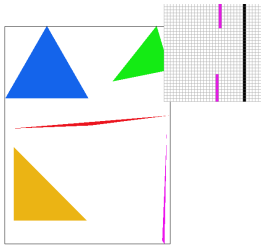
Task 1: Drawing Single-Color Triangles

I reviewed a bit the formula for determining whether a point is inside a triangle. I wrote a helper function to determine whether a point is on the counter-clockwise side of a line (A1, A2). Then, for a point to be inside a triangle, it should be at the counter-clockwise side along with the first-point-to-second-point direction of all three lines. I used a double for loop to achieve this for each pixel.

If the points of the given triangle are arranged in the clockwise direction, this would not work. To address this, I first examined if A3 is on the counter-clockwise direction of line (A1, A2). If not, we can rotate the orientation by switching two of the points, say, A1 and A2.

max value of x/y coordinate of each point of each triangle so that only pixels within the

bounding box by (xmin, ymin), (xmax, ymax) are needed to be sampled. Also need to apply floor/ceil functions to min/max functions so that it does not ignore edge-case pixels.



Task 1: Test 4

Task 2: Antialiasing by Supersampling

The basic idea for this task is to convert the original (width, height) sample buffer into a (width * sqrt(sample_rate), height * sqrt(sample_rate)), so that the whole buffer requires a total size of
$size = width * height * samplerate$

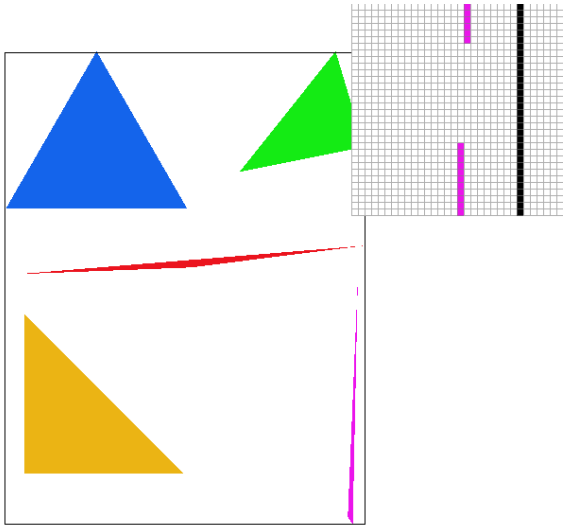
First resize the sample buffer into the (width * sqrt(sample_rate), height * sqrt(sample_rate)) dimension, and sample the triangle onto the buffer as in part 1. Then, to downsample all higher-resolution pixels into the original resolution, all pixels in the bounding box of (x * sqrt(sample_rate), y * sqrt(sample_rate)), ((x + 1) * sqrt(sample_rate) - 1, (y + 1) * sqrt(sample_rate) - 1) contribute to the pixel (x, y) in the original resolution. Each higher-resolution pixel should only contribute 1 / sample_rate, so I summed up all rgb components divided by sample_rate in Color of every pixel to get the the desired color density of downsampled (x, y).

In order to make up the effect of extending resolution operation on filling pixels for lines and points, all supersampled pixels that are in the area of the original pixel should all be filled with the same color, so that the weighted sum of colors of these pixels downsampled into the pixel at the desired location with the given color. Also should adjust other size parameters of functions such as clear_buffers().

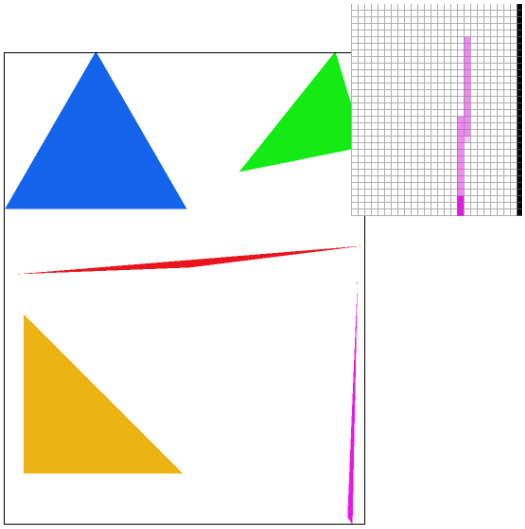
Supersampling is very useful in antialiasing, as instead of using a 0-1 value for the sampled pixel, it approximated how much proportion of that sampled area is covered by the triangle, and use that proportion to represent the area. This process smooths the jaggies around corners and edges of triangles by calculating more information of the sample region and used a wider range to represent the sampling pixel.

The antialiasing effect is noticeable, as with 2*2 sample rate the disconnected portion of the bottom-right triangle

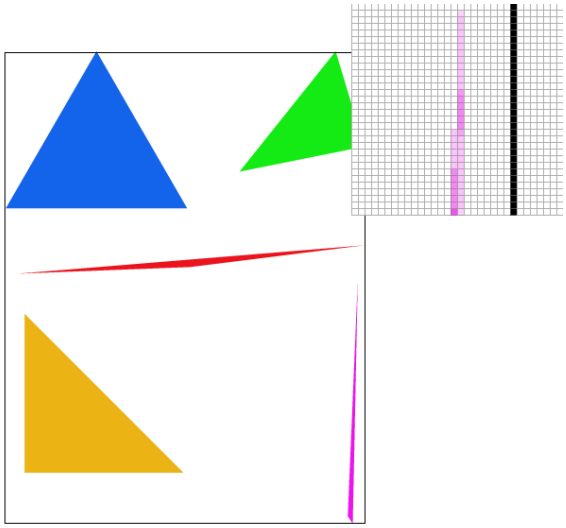
has reduced, and with 4*4 sample rate there is no disconnected pixels.



Task 2: Test 4 with 1*1 sample rate



Task 2: Test 4 with 2*2 sample rate



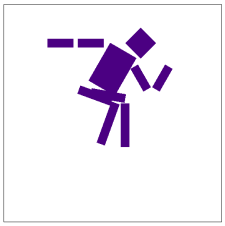
Task 2: Test 4 with 4*4 sample rate

In the first picture, the aliasing is severe because for some middle pixels in the upper part of the triangle, the corresponding region is partly covered by the triangle but the center of the region is not covered, which means the pixel does not show the covered area, and the pixels are disconnected. However, with supersampling, the pixel can show that part of the region is covered, and aliasing is mitigated.

Task 3: Transforms

I tried to make the cubeman to imitate the famous moonwalking by Michael Jackson, as shown below. I first rotate the torso and all limbs into the right angle, and then adjust the translation of each body part by trial and error. After a few rotation, the translation has become less intuitive to me because a cumulative 180 degree rotation ...original coordinates.

Task 3: MJ's mookwalking dance
Source:
https://www.iconradio.com/10242/michael-jacksons-first-moon-walk-an-iconic-moment/

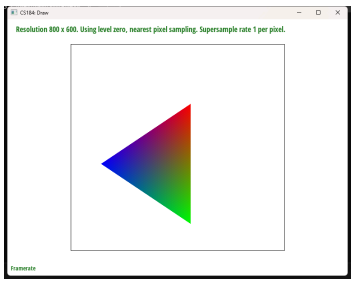


Task 3: Robot, mookwalk

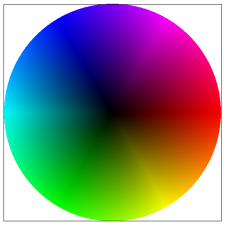
Task 4: Barycentric coordinates

Barycentric coordinate is a method to map every point P within a triangle ABC into three weights α, β, and γ, such that $\alpha A + \beta B + \gamma C = P$, and that $\alpha + \beta + \gamma = 1$. Barycentric coordinate can be used to decompose a point P into a linear weighted sum of all three vertices of the triangle. If the point P is closer to a point, say, A, the weight of A α is higher, and α is exactly 1 when point P is on the vertex A. On the other hand, when α equals to 0, it means that point P is at the fartherest side of vertex A within triangle P, that is, the projection of A onto BC. If it goes even farther, α goes to negative, which means that the point is no longer in the triangle.

Below is a visual demonstration of barycentric coordinates, where each vertex of the triangle is colored with red, green, and blue, and every point inside the triangle is filled with the color of the weighted sum of three α, β, and γ times their corresponding color. For example, the closer to the left vertex, the pixel is more blue since the corresponding weight of that vertex is higher and thus contributes more blue to that pixel. As it goes farther, it has less blue and more red/green in the pixel.



Task 3: Triangle interpolation with color



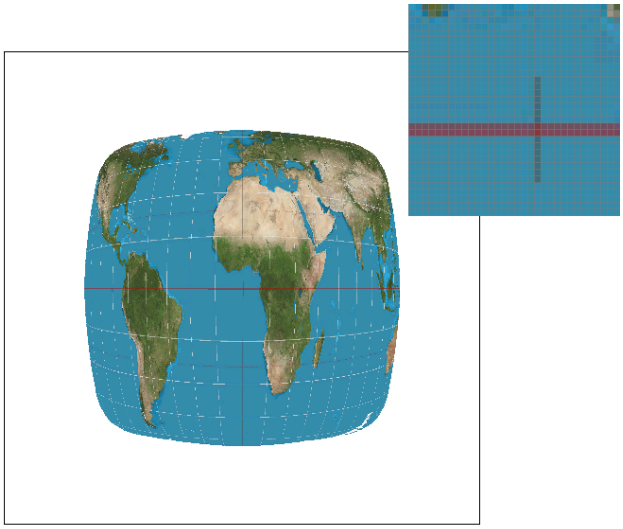
Task 4: Test 7

Task 5: "Pixel sampling" for texture mapping

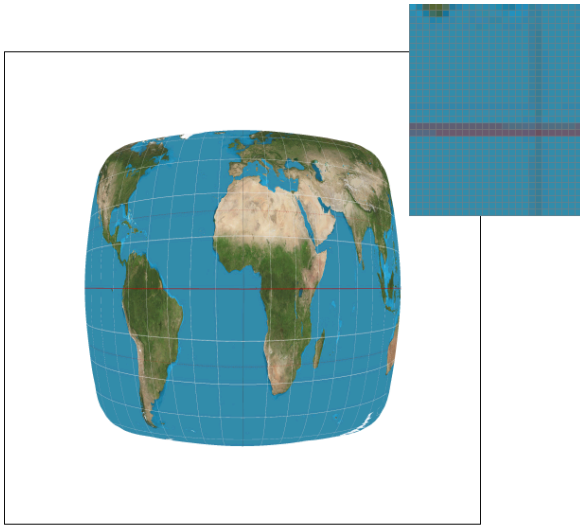
For task 5, I actually tried several bilinear sampling methods with subtle implementation differences that I think might be reasonable, and I chose the one with the best quality.

Given a triangle in the screen space, for each pixel in the triangle, we would want to calculate the barycentric coordinates of that pixel, and calculate the corresponding texel within the texture triangle with the same barycentric coordinates. Most of the time, this texel would not be an integer value and therefore would not be sampled. To address this, the nearest neighbor method takes the nearest sample point in texture and maps that into the pixel, whereas the bilinear sampling method takes the four closest sampling texel points, do a double lerp function on the four texels based on the x/y distance to each of the four texel, and use that weighted sum of texels to map to the pixel.

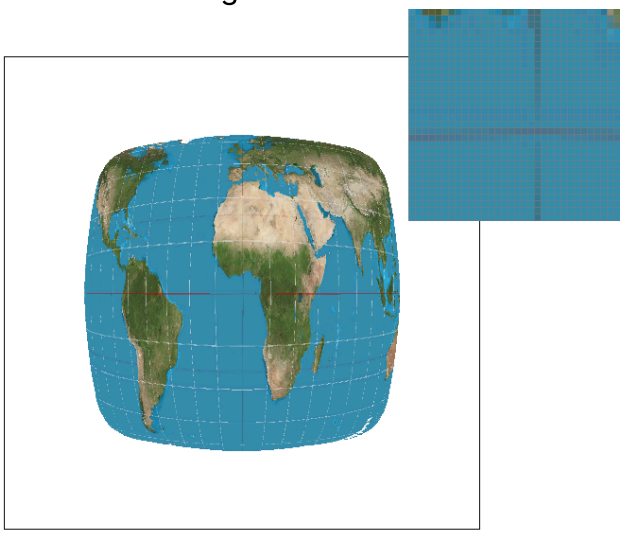
In the following examples, the red lines around the center is vulnerable to aliasing. The nearest neighbor neighbor sampling method draws only a segment of red line around the center. The aliasing is mitigated in bilinear sampling with a full vertical red line. The condition is better in both methods when supersampling is applied.



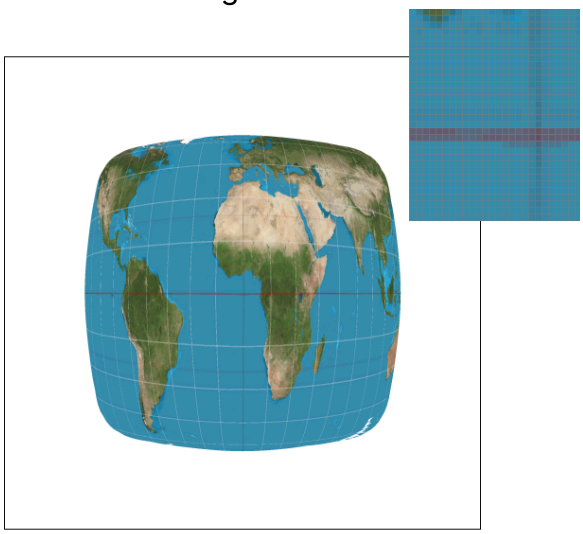
Task 5: Test 1 with 1∗1 sample rate using nearest neighbor



Task 5: Test 1 with 4∗4 sample rate using neares neighbor



Task 5: Test 1 with 1∗1 sample rate using bilinear sampling



Task 5: Test 1 with 4∗4 sample rate using bilinea sampling

Like in the examples above, the two methods would have a great difference when the sampling frequency is high, that is, there is a sharp difference in color within neighboring pixels. Under such condition, bilinear sampling tends to produce a much better quality as it averages the differencing pixels, which reduces aliasing.

Task 6: "Level Sampling" with mipmaps for texture mapping

A triangle at a farther distance would be smaller on the screen space, so that when applying texture, multiple texels would contribute to one pixel, which would cause aliasing. On the other hand, a triangle with a larger screen space than texture space would have one texels contribute to multiple pixels, again causing aliasing. Level sampling is a method to apply antialiasing by applying downsampling [illegible] different extent at different region according to how big is the screen space of that region

compared to the texture space. This can help to antialias texture at different distances and angles, which is hard to achieve with a uniform downsampling.

In this task, level sampling is with mipmap is implemented by calculating the level of the mipmap using the du/dx, dv/dx, du/dy, dv/dy, where x, y are the coordinates in screen space and u, v are the coordinates in texture space. These four derivatives describe the continuous shape of texture space relative to screen space. Then, use the downsampled mipmap at that level to the closest integer to get the desired texture, or get two textures according to the integer levels that bounds the calculated level value and apply a lerp.

Comparing pixel sampling, level sampling, and supersampling-then-downsampling:

Speed: pixel sampling and level sampling are close, depending on the specific method being used, and are comparatively faster than supersampling.

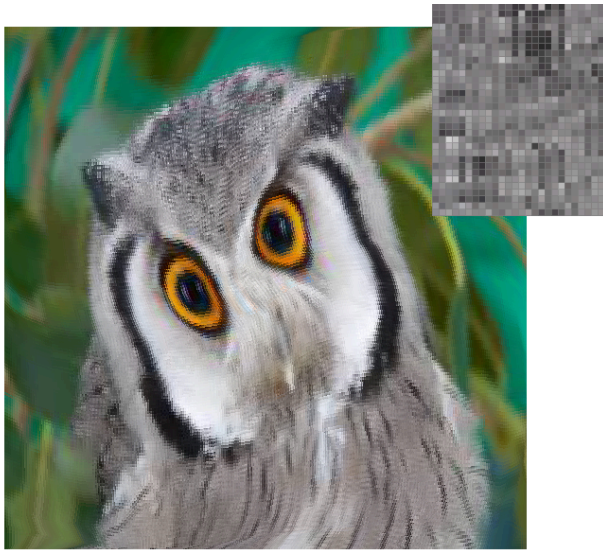
Memory usage: pixel sampling uses little additional memory, while level sampling uses around 2x the original memory, and supersampling/downsampling uses sample-size-times the original memory, if the intermediate supersampled result is stored in memory.

Antialiasing power: the supersampling/downsampling method achieves the best quality among all three, and a comparable result can be achieved by using either/both of pixel sampling and level sampling. Specifically, level sampling helps to antialias with different distances and angles, while pixel sampling is more general.

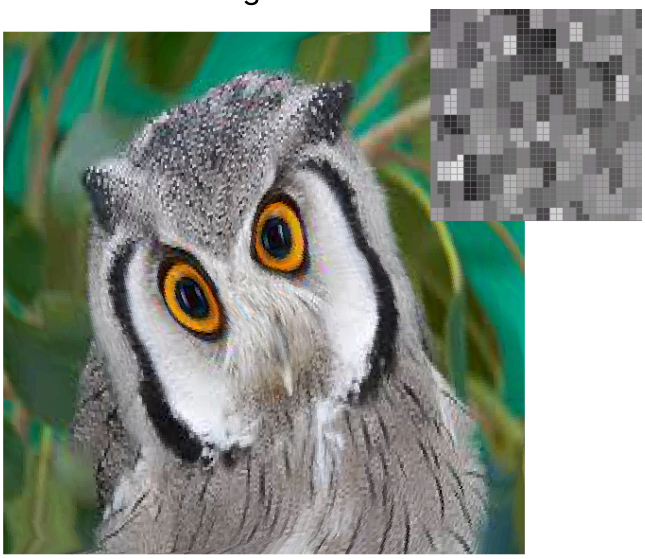
Task 6: A cute owl
Source:
https://animals.sandiegozoo.org/animals/owl

In the following examples, the owl picture is applied with distortive texture mapping. Notice the region around the head of the owl where alternative white and black color occurs. With pixel inspector, it is clear that the sampling frequency is reduced and the antialiasing is noticeable.

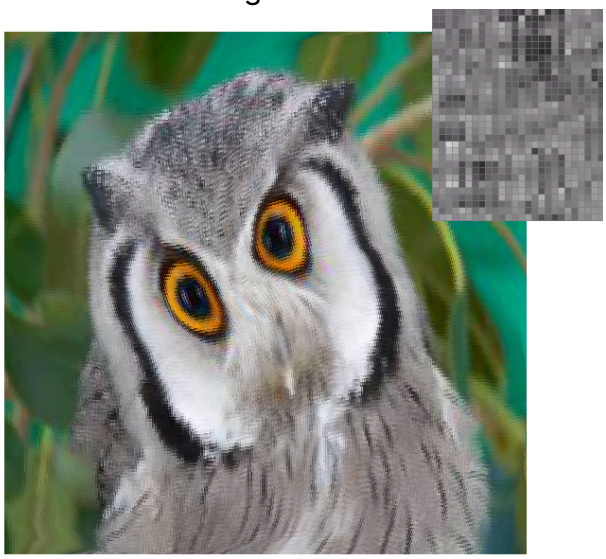
ask 6: Owl with zeroth level mipmap and nearest neighbor



Task 6: Owl with nearest mipmap level and neare neighbor



ask 6: Owl with zeroth level mipmap and bilinear sampling



Task 6: Owl with nearest mipmap level and bilinea sampling

Loading [MathJax]/extensions/MathZoom.js