CS184 Homework 1: Rasterizer

Martin Guo, 3039720257
guozy@berkeley.edu

This webpage is available at:
https://cal-cs184-student.github.io/hw-webpages-sp24-0-0-00-0/hw1/index.html

Overview

In this homework I built an implementation of rasterizing triangles with different sample rates, used transform matrices to realize tranforms of images, and using barycentric coordinates, implemented nearest and bilinear pixel interpolation as well as nearest level sampling and trilinear sampling.

1 Task 1: Drawing Single-Color Triangles

1.1 Algorithm

1. Suppose we are given $\triangle ABC$ with the vertices in order. First calculate the 3 vectors $\vec{AB}, \vec{BC}$ and $\vec{CA}$.

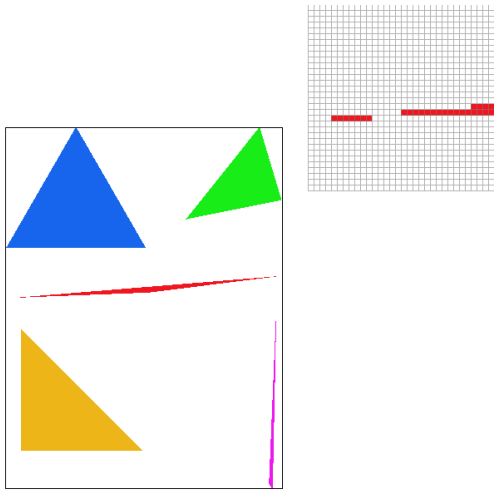
2. Calculate the bounding box of the triangle by recording the maximum/minimum values of vertex coordinates.

3. For each sample P within the bounding box, calculate $\vec{AB'} \cdot \vec{AP}$, $\vec{BC'} \cdot \vec{BP}$ and $\vec{CA'} \cdot \vec{CP}$ where $\vec{AB'}$ is $\vec{AB}$ rotated 90 degrees counterclockwise, similarly for $\vec{BC'}$ and $\vec{CA'}$. These are proportional to the directed distances from P to each edge. If these three values are all non-positive or all non-negative, then P is inside the triangle, and we call `rasterize_point()` to draw the pixel.

The algorithm only checks each sample within the bounding box.

1.2 Result

A result from `basic/test4.svg` with default parameters.



2 Task 2: Antialiasing by Supersampling

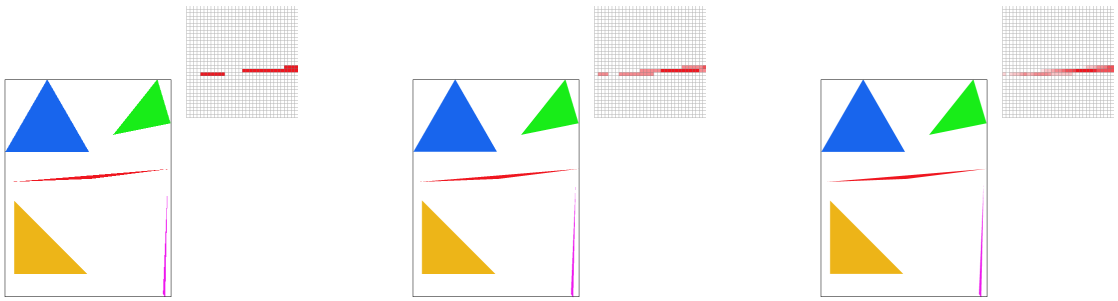
2.1 Algorithm

`sample_buffer` is modified such that it contains `width * height * sample_rate` Color type variables, and the sample at (row-i, column-j) of the pixel at coordinates (x, y) corresponds to `sample_buffer[(y * width + x) * sample_rate + j * r + i]`, where `r=sqrt(sample_rate)`. Sample (i,j) of pixel (x,y) is sampled at coordinates

$$(x + \frac{2i+1}{2r}, y + \frac{2j+1}{2r})$$

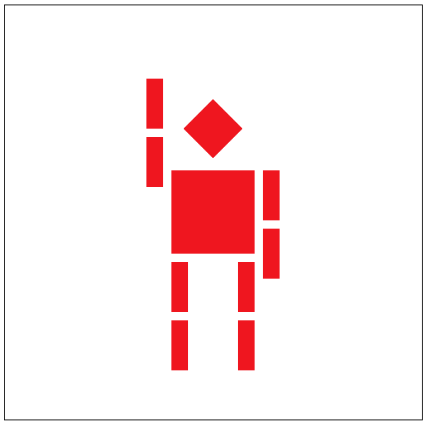
so that all samples have even spacing. When `sample_rate` is 1, the coordinates are precisely $(x + 0.5, y + 0.5)$.
The first 2 steps of the algorithm are the same as before. In step 3, the algorithm does a point-in-triangle test on each sample, and sets `sample_buffer` directly with the color.
Supersampling and averaging reduces the variance of the resulting pixels, thus reducing the frequency of the result and achieving the effects of antialiasing.

2.2 Result



The 3 results from basic/test4.svg are under sample rates 1, 4, 16 respectively. As we can see, the result with sample rate 1 has only red and white pixels; as sample rate increase, there are more pixels with intermediate colors. This reduces the frequency of the image, especially on triangle edges and corners.

3 Task 3: Transforms

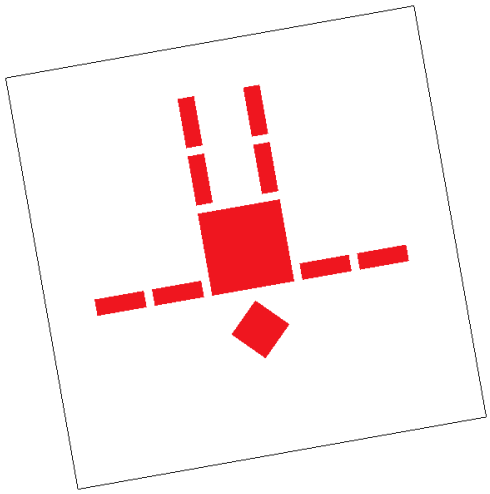


This picture is taken after translating the cubeman's left and right arms to appropriate positions, rotating each of them by +90° and translating them back.

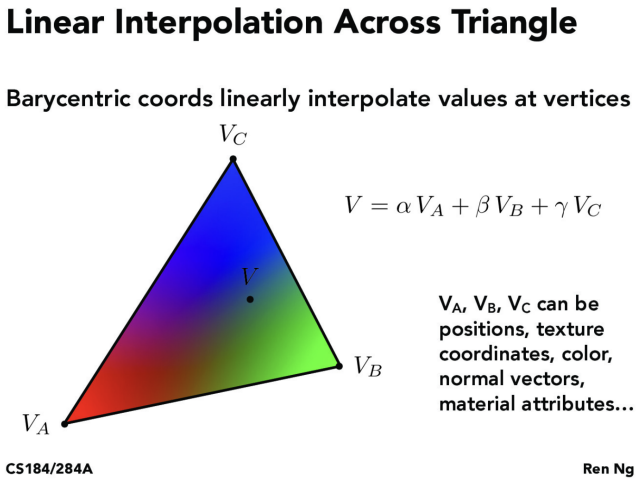
Extra Credit: Implemented a new GUI feature so that when the left(right) arrow key is pressed, the renderer rotates the whole image by 5 degrees counterclockwise(clockwise). The rotation is implemented by right multiplying the `svg_to_ndc[current_svg]` matrix by

```
translate(width*.5, height*.5) * rotate(deg) * translate(width*-.5, height*-.5)
```

where `width` and `height` are member variables of the current `svg`, and `deg` is -5 in the case of left arrow key and 5 if right arrow key.
An image after implementing the transform:

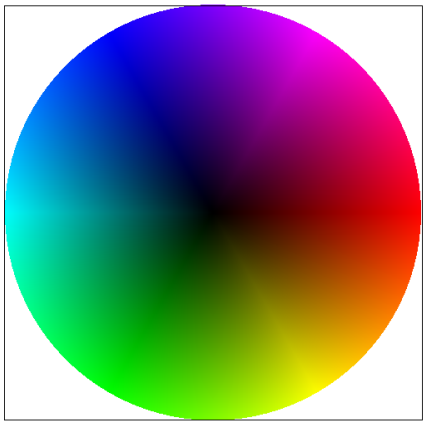


4 Task 4: Barycentric coordinates



Here is a picture from the slides which demonstrates the linear blending effect of using barycentric coordinates. For vertex $V = \alpha V_A + \beta V_B + \gamma V_C$, the coordinate α tells how far away V is from the edge BC compared to the distance from the vertex A to BC. The closer V is to BC, the farther it is from A and the less share of weight it gets from A's color. Another interpretation could be the ratio of the area of $\triangle VV_BV_C$ to the area of $\triangle V_AV_BV_C$. This is more representational in showing that the coordinate values sum up to 1.

The following is the result from `basic/test7.svg` with default parameters.



5 Task 5: "Pixel sampling" for texture mapping

5.1 Algorithm

Pixel sampling is sampling from texture space a color to use for a particular pixel.

5.1.1 Nearest pixel sampling

Scale up the uv coordinates by `width` and `height` and round down to the nearest integer to acquire the indices for the texture grid to sample from; this is the same as viewing each pixel of the texture grid as having coordinates $(x + 0.5, y + 0.5)$ and sample from the nearest texture pixel.

5.1.2 Bilinear pixel sampling

Scale up the the uv coordinates in the same way as nearest sampling, then subtract it by (0.5,0.5). Find the two nearest integers to each of u and v, sample from texture map using all 4 combinations of the integers, and calculate a bilinear mixture of the values as the sampled value of (u, v).
Edge cases: After scaling and shifting the uv coordinates, there exist edge cases where the coordinate is in either $[-0.5, 0)$ or $(max - 1, max - 0.5]$, where max is either *height* or *width*. This implementation just takes the corresponding edge value. For example, if uv coordinates after scaling is $(-0.3, 2.3)$, then the sampled value would be `get_texel(0,2)*0.7 + get_texel(0,3)*0.3`.

5.2 Results

nearest, sample_rate=1

bilinear, sample_rate=1

nearest, sample_rate=16

bilinear, sample_rate=16

The above are the results from `texmap/test6.svg`. In the case where sample rate is 1, nearest pixel sampling exhibits a fair amount of aliasing, while bilinear sampling clearly reduces it. When sample rate equals 16, the difference is less pronounced.
Swapping nearest sampling with bilinear sampling seems to have the most effect in high-frequency parts of the image, or where screen sampling rate is lower.

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

6.1 Algorithm

Screen sample rates may differ within the same image. If only sample from one level of texture, aliasing and blurring may occur in different parts of the image at the same time. Level sampling involves different levels of texture exhibiting different frequencies, and performs texture mapping from a most suitable texture level according to screen sampling frequency.

6.1.1 Choosing level

1. First, scale up `p_uv`, `p_dx_uv` and `p_dy_uv` by level 0 mipmap's width and height.
2. This implementation picks the level according to the *geometric mean* of the 2 differential vectors. That is, $D = \log_2 L$ where

$$L = \sqrt[4]{((\frac{du}{dx})^2 + (\frac{dv}{dx})^2)((\frac{du}{dy})^2 + (\frac{dv}{dy})^2)}$$

   L also has an intuitive interpretation as the area of the texture footprint.

6.1.2 Nearest and bilinear level interpolation

Nearest level interpolation computes the nearest integer level to the floating-point level given above, and sample from that level according to the pixel sampling method. Bilinear level interpolation samples from the 2 nearest integer levels, and compute a linear interpolation of the values.
In the rasterizer program, `p_dx_uv` and `p_dy_uv` are to be computed in addition to the *uv* coordinates before sampling.

6.2 Results

The following are the results from `texmap/test6 (2).svg`, which is a copy of `texmap/test6.svg` with another `.png` file. The pixel inspector is placed at a central-to-upper-left part of the image. Sample rate is 1. Level and pixel sampling methods are shown in the captions.

`L_ZERO, P_NEAREST`

`L_ZERO, P_LINEAR`

`L_NEAREST, P_NEAREST` `L_NEAREST, P_LINEAR`

As we can see, the images displayed at level zero are both highly aliased, where bilinear sampling makes the aliasing more regular. Sampling from the nearest level dramatically reduces aliasing, and has the best antialiasing effect when coupled with bilinear sampling.

6.3 Performance comparisons

Antialiasing method	Supersampling	Bilinear pixel sampling	Mip-mapping
Runtime	Highest	Medium	Medium
Memory usage	Highest	Low	High
Antialiasing power	Medium	Medium	High

In terms of antialiasing power, mip-mapping combined with bilinear sampling has the most effect. Applying supersampling in addition to these doesn't improve antialiasing effect very much.

In antialising, mip-mapping does more on balancing aliasing and blurring over the whole picture. So does linear level interpolation. The following results, also from `texmap/test6 (2).svg`, are a good demonstration of the effects from linear level interpolation. Sample rate is 1, using bilinear sampling. The pixel inspector is positioned near the top right of the image.

`L_NEAREST` `L_LINEAR`

The image on the left samples from nearest level, and we could make out a vertical boundary in the middle of the pixel inspector view, where on the left side the image is aliasing and on the right side it is blurring. Linear level interpolation, on the other hand, interprets texture level as a continuous number, making the picture free of boundaries like this. The line stroke is fairly continuous in the image on the right.