CSC4140 Assignment 3 & 4

Computer Graphics

March 19, 2022

Transformation

This assignment is 10% of the total mark.

Strict Due Date: 11:59PM, March 19^{th}, 2022

Student ID: 118010202

Student Name: Zhixuan LIU

This assignment represents my own work in accordance with University regulations.

Signature: Zhixuan Liu

1 Draw a single color triangles

In this part, I will implement how to draw a single color triangle and uses the svg test files to see whether the triangles are correctly drawn.

Assume the triangle we get has three end points $(x_0, y_0, x_1, y_1, x_2, y_2.)$ And we want to rasterize all the sample points in this triangle. So the first thing we do is to find a rectangular outside this triangle for testing and sampling simplicity.

```
// FIND THE REC BOUNDARY OF THE TRIANGLE
int x_min = min(min(x0, x1), x2), y_min = min(min(y0, y1), y2);
int x_max = max(max(x0, x1), x2), y_max = max(max(y0, y1), y2);
```

After find the rectangle boundary of the triangle we want to rasterize, next we will sample all the points and check whether this sampled point is in the triangle or not by using the line_test.

```
// GO THROUGH ALL THE POINT AND SAMPLE IT AT THE ORIGON OF THE PIXEL WICH IS (0.5, 0.5)
// CHECK WHETHER A POINT IS IN THE TRIANGLE BY USING LINE CHECK
for(int yi = y_min; yi <= y_max; yi++){
   for(int xi = x_min; xi <= x_max; xi++){
     float x_sample = xi + 0.5, y_sample = yi + 0.5;
     if(point_in_triangle(x_sample, y_sample, x0, y0, x1, y1, x2, y2)){
       rasterize_point(x_sample, y_sample, color);
     }
   }
}
```

Now we look deeper into how the how to check whether a point is in a triangle or not. Here we consider two cases: the provided points are ordered in counter clock-wise and clock-wise.

For the counter clock-wise, the production should be all greater than zero. and we include the boundary, which becomes greater and equal to zero.

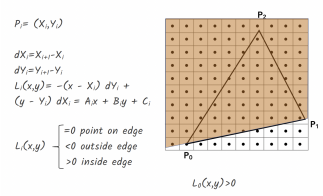


Figure 1: Line test first.

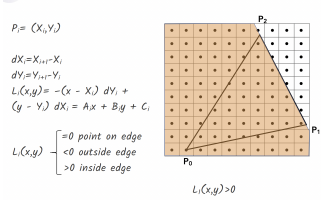


Figure 2: Line test second.

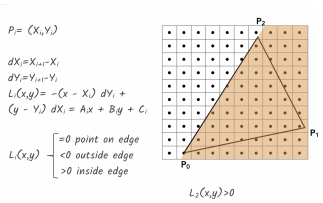


Figure 3: Line test third.

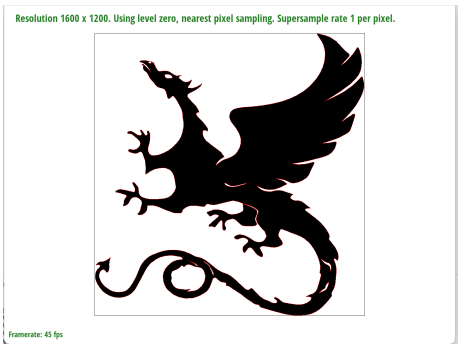
Similarly, we consider the clock-wise triangle, which the line test should be all smaller than zero. In summary, the codes can be written like this which is compatible for all the triangle rending ways and also includes the boundaries of the triangle.

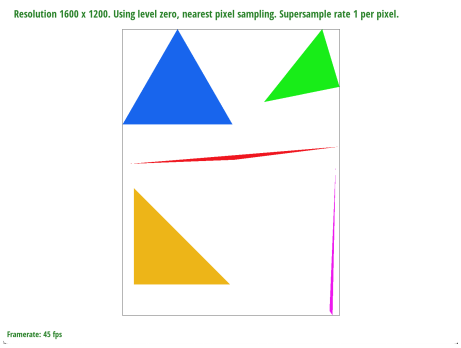
```
// CHECK POINT IN THE TRIANGELE OR NOT

```

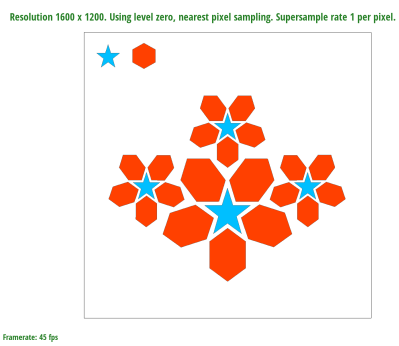
```
3   int line_test(float x, float y, float x0, float y0, float x1, float y1){
4     return (-(x-x0)*(y1-y0) + (y-y0)*(x1-x0));
5   }
6
7  bool RasterizerImp::point_in_triangle(float x, float y,
8  float x0, float y0, float x1, float y1, float x2, float y2){
9     // COUNTER CLOCK-WISE
10    bool counter_clock_wise = (line_test(x, y, x0, y0, x1, y1) >= 0) \
11    && (line_test(x, y, x1, y1, x2, y2) >= 0) && \
12    (line_test(x, y, x2, y2, x0, y0) >= 0);
13    // CLOCK-WISE
14    bool clock_wise = (line_test(x, y, x0, y0, x1, y1) <= 0) \
15    && (line_test(x, y, x1, y1, x2, y2) <= 0) && \
16    (line_test(x, y, x2, y2, x0, y0) <= 0);
17
18    return clock_wise || counter_clock_wise;
19  }
```

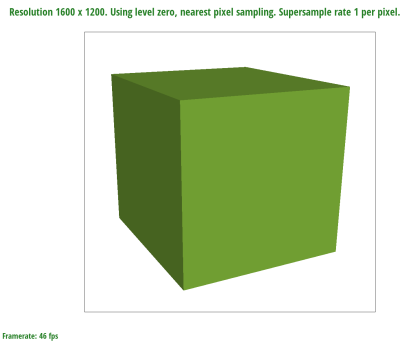
After this, we can successfully render the triangles.

1.1 Selected Results









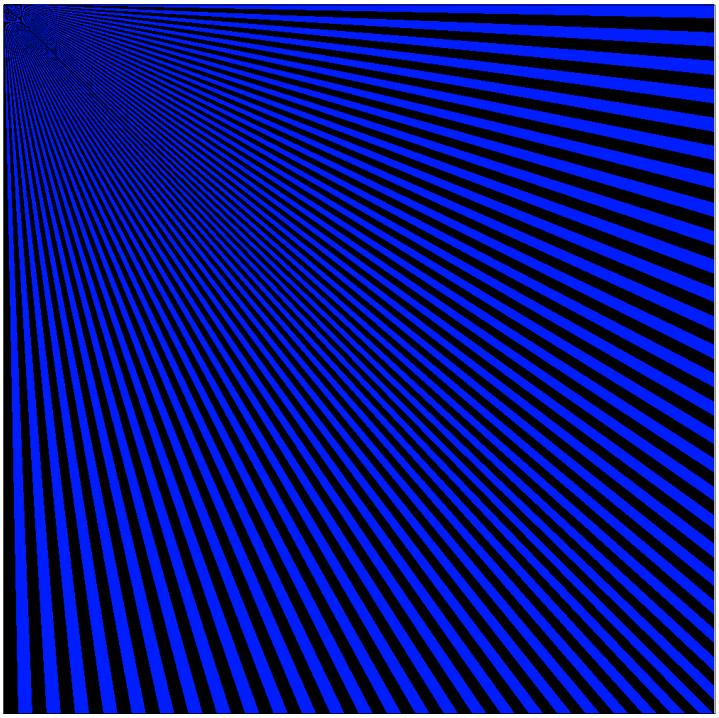


Figure 4: Hardcore 1

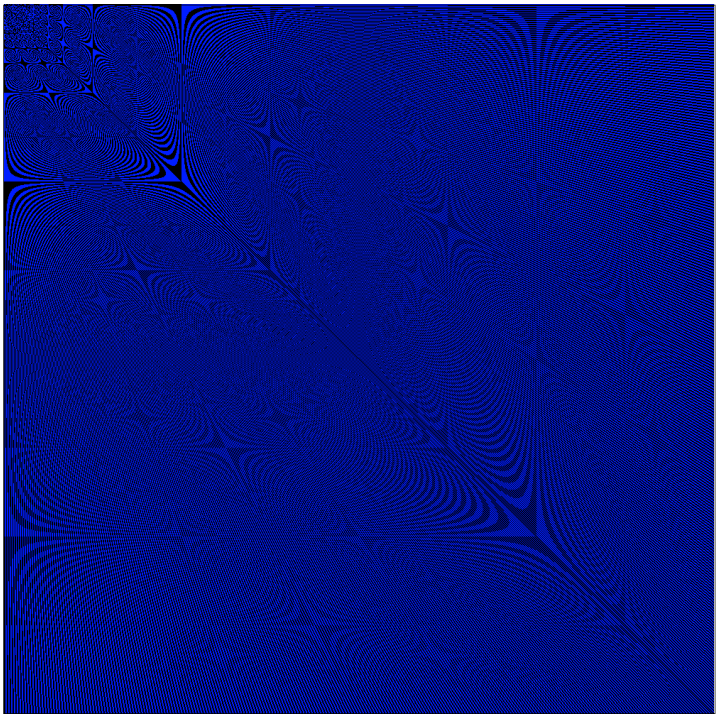


Figure 5: Hardcore 2

1.2 Troubles that I met in this tasks

In this task, I firstly cannot render all the triangles successfully because of I failed to implement the "point in triangle" test. And also if I only implement the counter clock-wise or the clock-wise version of the test, we will only get some of the results like this:

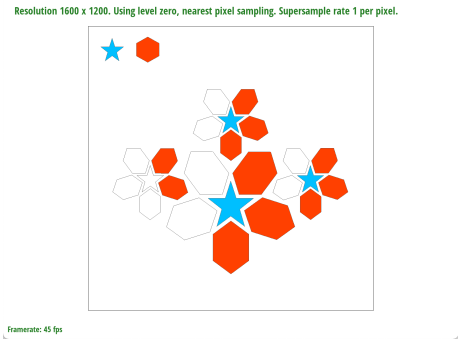


Figure 6: Only counter clock-wise triangle.

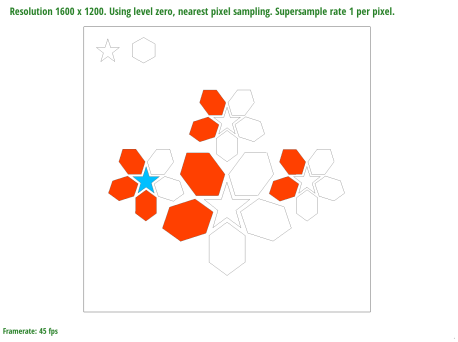


Figure 7: Only clock-wise triangle.

2 Antialiasing

In this section, I will use super sampling in order to implement anti-aliasing. A basic idea of anti-aliasing is that, for a single pixel, we use much more sample point and then calculate the average sampling color.

For example, if we set the sampling rate to be $sample\_rate$, and in one pixel we want to calculate the color for, we will have the number of $sample\_rate$ points to do the virtual sample. The edge of each sample will be $\sqrt{sample\_rate}$. The following image does the further implication,

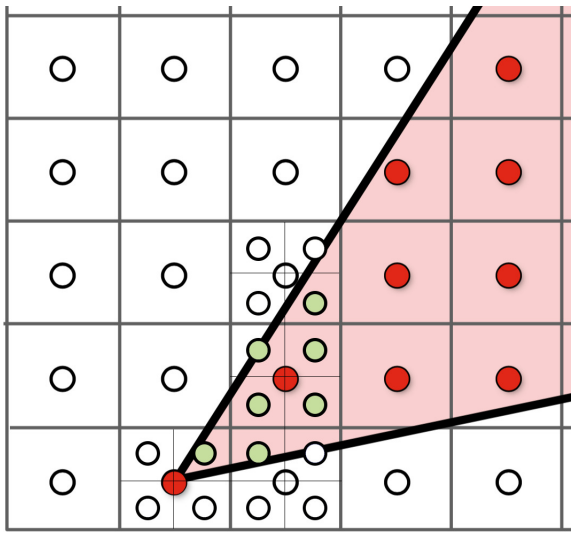


Figure 8: Super sampling illustration.

To implement this, the size of the $sample\_buffer$ which is used to store the information about sample points and colors will be changed automatically and dynamically since the size of the buffer will always being changing due to the rising sampling rate.

```
sample_buffer.resize(width * height * sample_rate, Color::White);
```

And for the actual sample process, we will first go through all the sample points just as what we do in the task 1 before. Then we will just go into the sub sampling points in one pixel to do the sampling and store the color information in the resized buffer.

```
for(int yi = y_min; yi <= y_max; yi++){
for(int xi = x_min; xi <= x_max; xi++){
    // KNOW FOCUS ON ONE PIXEL
    int count = 0;
    for (int yii = 0; yii < edge_rate; yii++){
      for (int xii = 0; xii < edge_rate; xii++){
        float x_sample = xi + (((float)xii + 0.5) / float(sample_rate));
        float y_sample = yi + (((float)yii + 0.5) / float(sample_rate));
        if(point_in_triangle(x_sample, y_sample, x0, y0, x1, y1, x2, y2)){
        sample_buffer[(width*yi + xi)*sample_rate + count] = color;
```

Then in the $RasterizerImp :: resolve\_to\_framebuffer()$ functions, we want to get the color in the framebuffer on the screen. In this case, we will add up all the colors in the buffer for one pixel and calculate the average color by dividing the $sample\_rate$.

```
Color col;
for (int xii = 0; xii < sqrt(sample_rate); xii++){
  for (int yii = 0; yii < sqrt(sample_rate); yii++){
    col += (sample_buffer[(y * width + x) * sample_rate + count])*(float(1)/sample_rate);
  }
```

2.1 Results

The results of super sampling are like this:

2.2 Problems that I met

In this section, I met one problem about the buffer. Since I want to reduce the size of the buffer, therefore, I want to solve the color information in the only one $Color$ variable. However, I met this problem:

3 Transforming

I complete three transforming matrices like this:

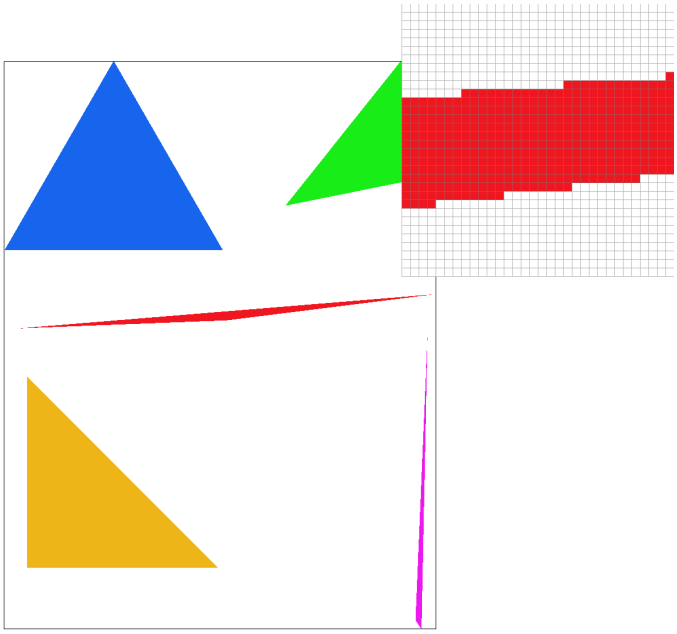


Figure 9: Super sampling rate is 1.

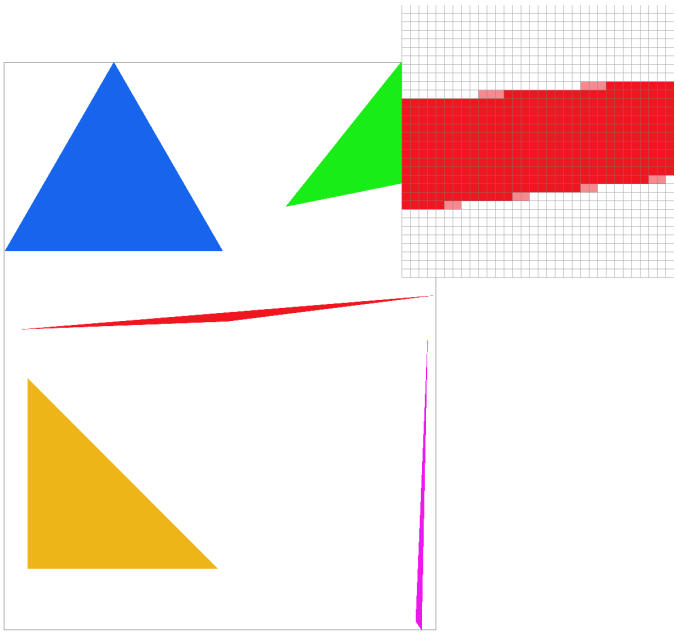


Figure 10: Super sampling rate is 4.

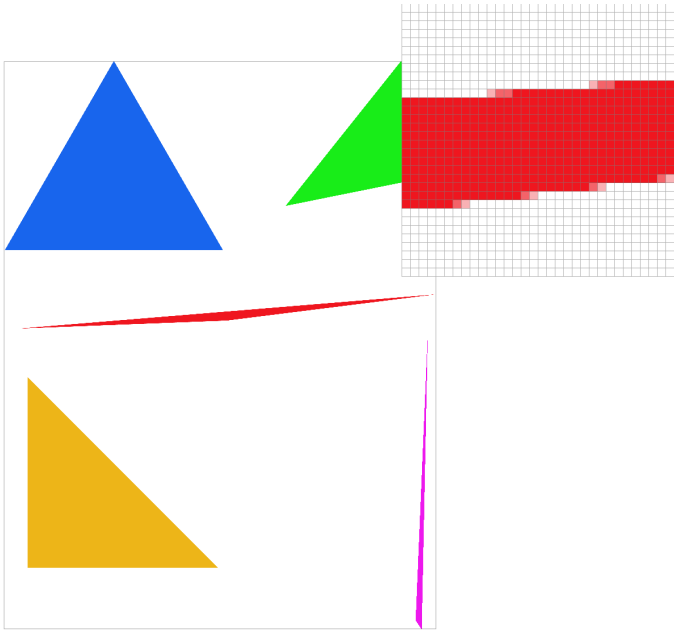


Figure 11: Super sampling rate is 9.

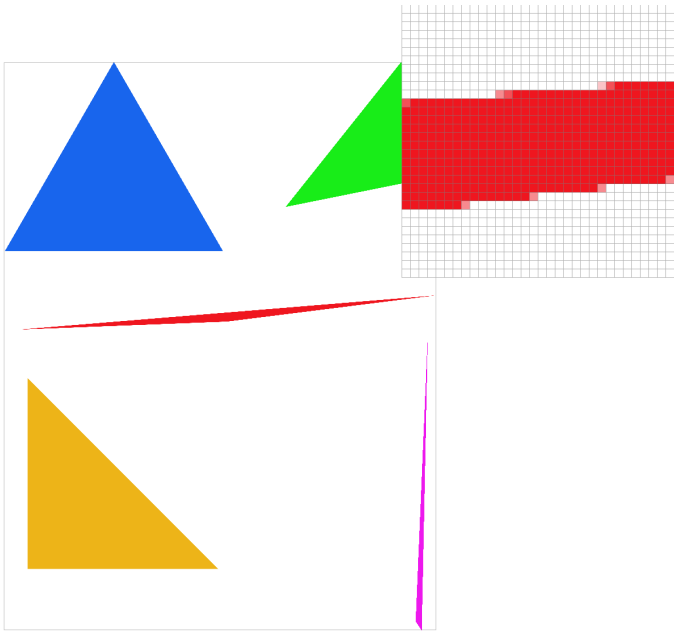


Figure 12: Super sampling rate is 16.

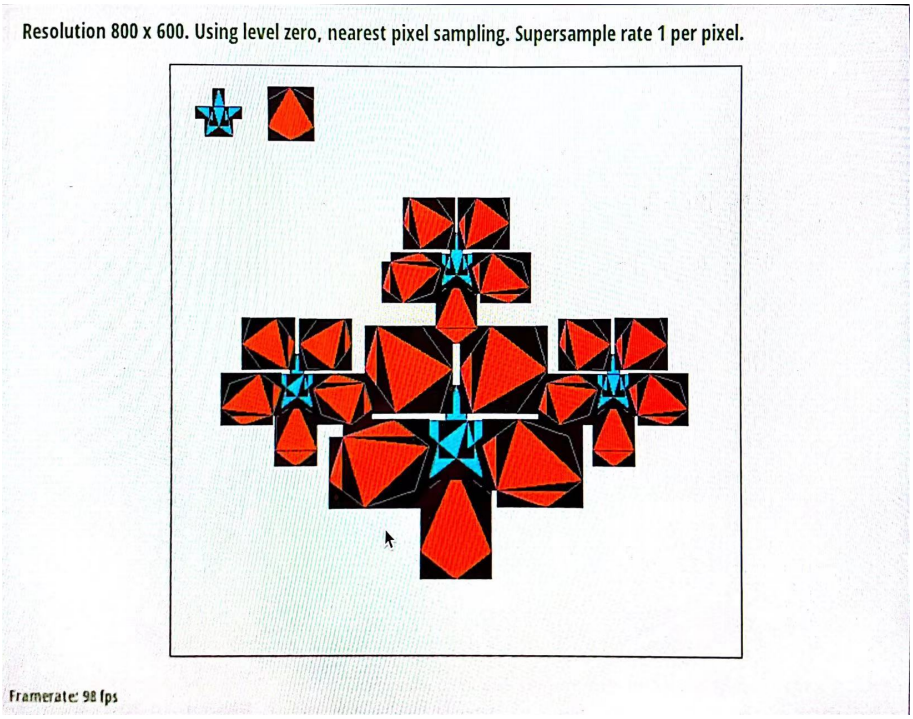


Figure 13: Problems in task 2 related to sampling buffer.

```
Matrix3x3 translate(float dx, float dy) {
  // Part 3: Fill this in.
  return Matrix3x3(1, 0, dx,
          0, 1, dy,
          0, 0, 1);
}

Matrix3x3 scale(float sx, float sy) {
  // Part 3: Fill this in.
  return Matrix3x3(sx, 0, 0,
          0, sy, 0,
          0, 0, 1);
}

// The input argument is in degrees counterclockwise
Matrix3x3 rotate(float deg) {
  // Part 3: Fill this in.
  return Matrix3x3(cos(deg*M_PI/180), -sin(deg*M_PI/180), 0,
          sin(deg*M_PI/180), cos(deg*M_PI/180), 0,
          0, 0, 1);
}
```

3.1 Results

The original robot looks like this:

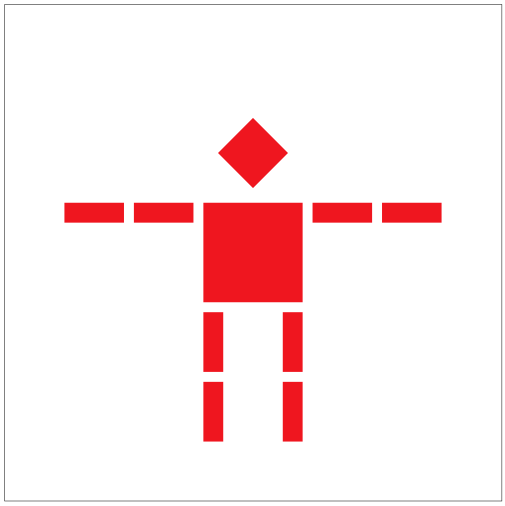


Figure 14: Original robot.

My robot looks like this:

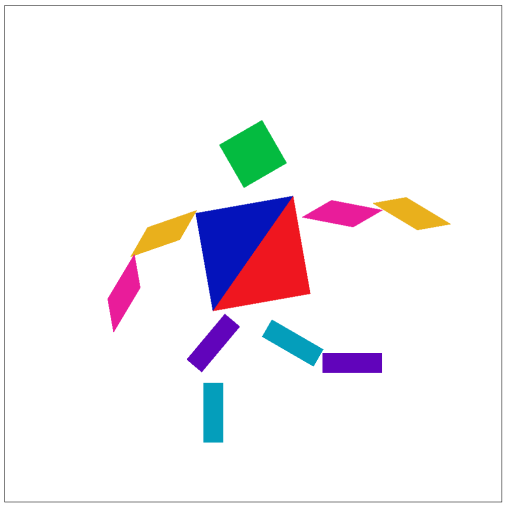
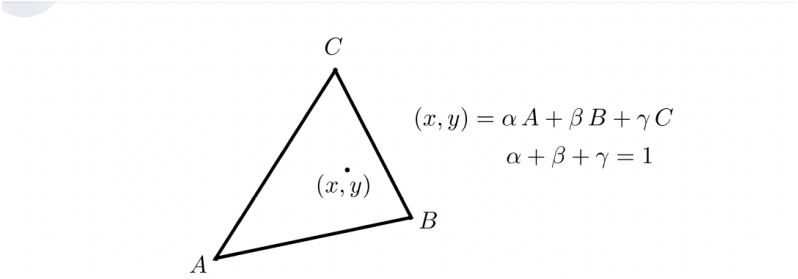


Figure 15: My robot.

4 Barycentric coordinates

In this part, we calculate the Barycentric coordinates for a triangle. I follow the way to calculate the coordinates in the figure:



$$\alpha = \frac{-(x - x_B)(y_C - y_B) + (y - y_B)(x_C - x_B)}{-(x_A - x_B)(y_C - y_B) + (y_A - y_B)(x_C - x_B)}$$

$$\beta = \frac{-(x - x_C)(y_A - y_C) + (y - y_C)(x_A - x_C)}{-(x_B - x_C)(y_A - y_C) + (y_B - y_C)(x_A - x_C)}$$

$$\gamma = 1 - \alpha - \beta$$

Figure 16: Barycentric coordinates formula.

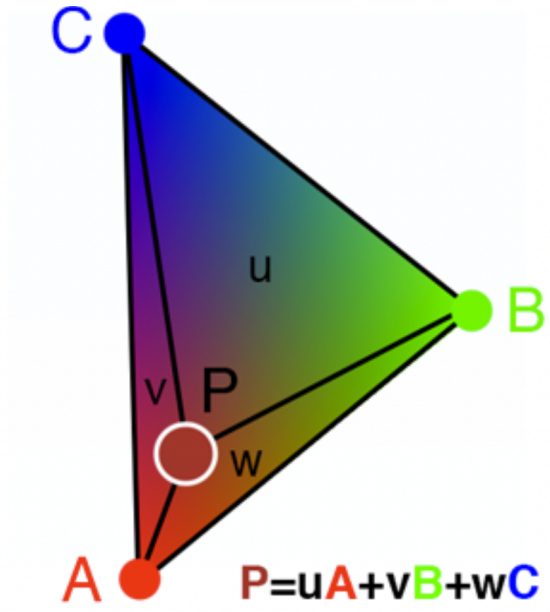


Figure 17: Barycentric coordinates image shown.

And the following codes just follows the formula I discussed about:

```
float x_sample = xi + (((float)xii + 0.5) / float(sample_rate));
float y_sample = yi + (((float)yii + 0.5) / float(sample_rate));
if(point_in_triangle(x_sample, y_sample, x0, y0, x1, y1, x2, y2)){
  float para0, para1, para2;
  para0 = ((x1 - x_sample) * (y2 - y1) + (y_sample - y1) * (x2 - x1))/ ((x1 - x0) * (y2 - y1) +
        (y0 - y1) * (x2 - x1));
  para1 = ((x2 - x_sample) * (y0 - y2) + (y_sample - y2) * (x0 - x2))/ ((x2 - x1) * (y0 - y2) +
        (y1 - y2) * (x0 - x2));
  para2 = 1 - para1 - para0;
  sample_buffer[(width*yi + xi)*sample_rate + count] = para0* c0 + para1 * c1 + para2 * c2;
}
```

4.1 Results

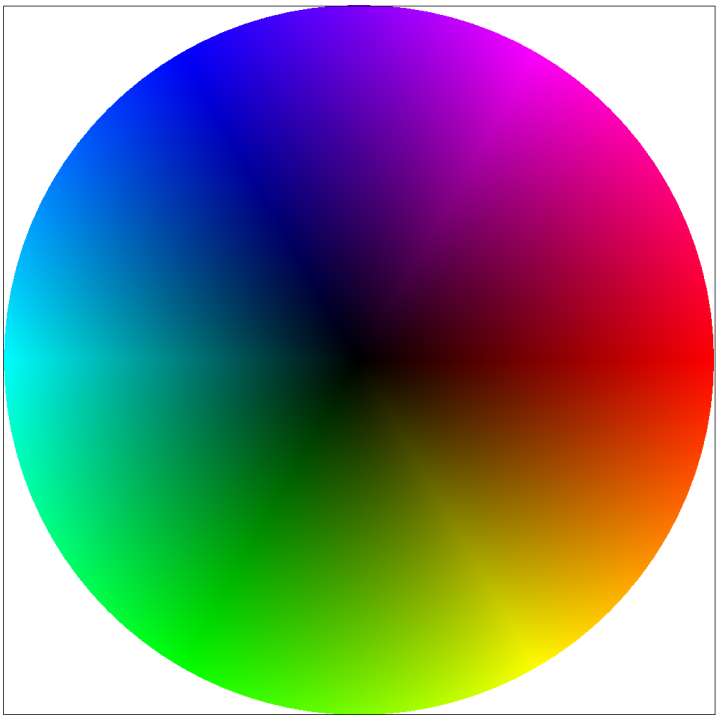


Figure 18: Barycentric coordinates results.

5 Texture Mapping

In this section, I will implement two ways of texture Mapping, in order to map an texel onto a pixel. We will consider to situation: the magnification and minification.

5.1 Magnification

In the magnification, it is good that each pixel will sample at only one texel, however, each multiple pixels may have the same texel values. Therefore, there are two ways for choosing the correct texel.

5.2 Nearest sampling

The first one is nearest sampling, which is find the neast texel neighbor of the sampling point.

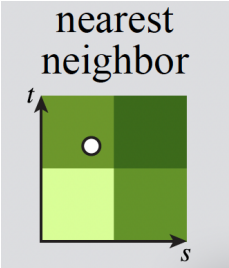


Figure 19: Nearest sampling algorithm.

Fist we get the sample point coordinate on the texel, and then round it to the nearest texel, and then return this color.

The code implmentation can be written like this:

```
1  Color Texture::sample_nearest(Vector2D uv, int level) {
2    // TODO: Task 5: Fill this in.
3    Color col;
4    auto& mip = mipmap[level];
5    // std::cout << level << std::endl;
6    float sample_u = uv[0]*mip.width;
7    float sample_v = uv[1]*mip.height;
8    int mip_sample_u = round(sample_u);
9    int mip_sample_v = round(sample_v);
10
11   col = mip.get_texel(mip_sample_u,mip_sample_v);
12
13   // return magenta for invalid level
14   return col;
15 }
```

5.3 Linear sampling

The first one is nearest sampling, which is finding the four nearest neighbors of the sampling point, and then use the linear function to get a smooth transition. This can be done in three steps.

First is to calculate the transition in the upper level, the second is to calculate it for the lower level, and the third is for the transition in the upper and lower level.

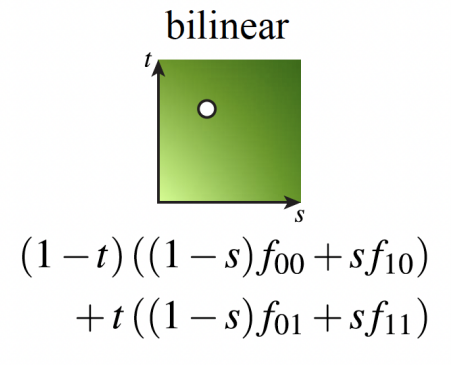


Figure 20: Linear sampling algorithm.

The code implmentation can be written like this:

```
1   Color Texture::sample_bilinear(Vector2D uv, int level) {
2     // TODO: Task 5: Fill this in.
3     Color col;
4     auto& mip = mipmap[level];
5     float u = uv[0]*mip.width;
6     float v = uv[1]*mip.height;
7     int floor_u = floor(u);
8     int ceil_u = ceil(u);
9     int floor_v = floor(v);
10    int ceil_v = ceil(v);
11    float alpha = u - floor_u;
12    float beta = v - floor_v;
13    // (floor_u, ceil_v) (ceil_u, ceil_v)
14    //      1 - beta
15    //        beta
16    // (floor_u, floor_u) alpha; 1-alpha (ceil_u, floor_v)
17    Color ul = mip.get_texel(floor_u,ceil_v);
18    Color ur = mip.get_texel(ceil_u,ceil_v);
19    Color dl = mip.get_texel(floor_u,floor_v);
20    Color dr = mip.get_texel(floor_u,ceil_v);
21
22    Color up_row_middle = (1-alpha) * ul + alpha * ur;
23    Color down_row_middle = (1-alpha) * dl + alpha * dr;
24    col = (1-beta) * down_row_middle + beta * up_row_middle;
25    return col;
26  }
```

5.4 Get the correct texture for texture mapping

Since we have two sampling algorithm, we now need to correctly map (x,y) to the correct (u,v).

Since we only have the information of the points in the vertex of the triangle, if we sampled (x,y) point is in the middle, we need to do the interpolation.

The first thing is to use (x0, y0), (x1, y1), and (x2, y2) to calculate the Barycentric coordinates for (x,y) instead of using uv coordinate because the projection is not affine. Then use the calculated Barycentric coordinates to get the uv coordinate. Then do the sampling mentioned above. The codes are here:

```
1 para0 = ((x1 - x_sample) * (y2 - y1) + (y_sample - y1) * (x2 - x1))/ ((x1 - x0) * (y2 - y1) + (
      y0 - y1) * (x2 - x1));
2 para1 = ((x2 - x_sample) * (y0 - y2) + (y_sample - y2) * (x0 - x2))/ ((x2 - x1) * (y0 - y2) + (
      y1 - y2) * (x0 - x2));
3 para2 = 1 - para1 - para0;
4
5 Vector2D uv = para0 * Vector2D(u0, v0) + para1 * Vector2D(u1, v1) + para2 * Vector2D(u2, v2);
6 params.p_uv = uv;
7 Color col;
8 col = tex.sample(params);
9 sample_buffer[(width*yi + xi)*sample_rate + count] = col;
```

5.5 Results

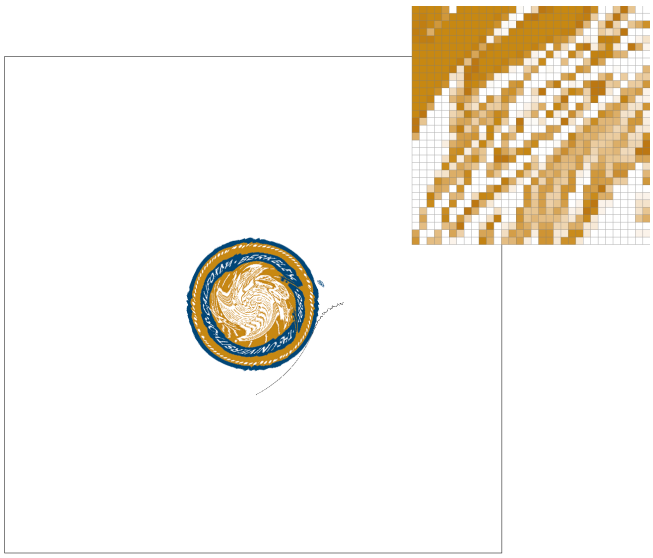


Figure 21: Nearest sampling algorithm showcase.

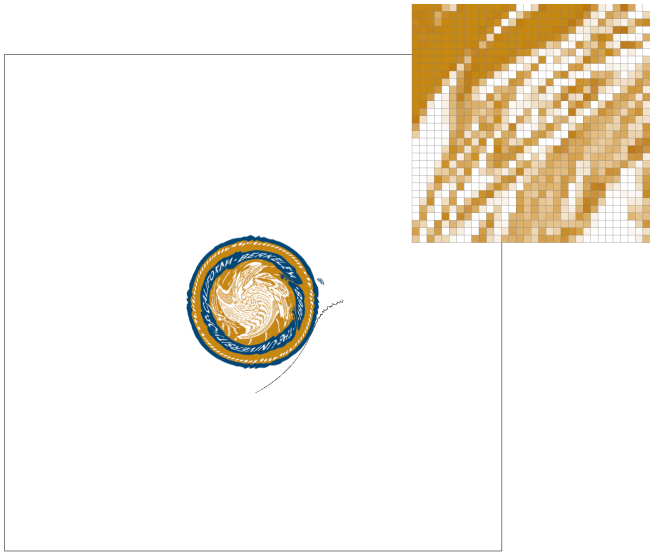


Figure 22: linear sampling algorithm showcase.

6 Mipmapping

In addition to the above mentioned two sampling algorithm, we include a so-called mipmap. It can be calculated using the algorithms in the pictures 23 and the following codes:

First we will get the level D of the mip map.

```
1  float Texture::get_level(const SampleParams& sp) {
2    // TODO: Task 6: Fill this in.
3    Vector2D target = sp.p_dx_uv.norm2() >= sp.p_dy_uv.norm2() ? sp.p_dx_uv : sp.p_dy_uv;
4    float target_u = target[0] * this->width;
5    float target_v = target[1] * this->height;
6    // float target_u = target[0];
7    // float target_v = target[1];
8    float L = sqrt(target_u*target_u + target_v * target_v);
9    float d = log2(L);
```

We will also have function to choose which algorithm to use:

```
1  Color Texture::sample(const SampleParams& sp) {
2    // TODO: Task 6: Fill this in.
3    float d = get_level(sp);
4    int nearest_d = round(d);
5    int floor_d = floor(d);
6    int ceil_d = ceil(d);
7    int alpha = d - floor_d;
8
9    Color col;
10   if (sp.lsm == L_ZERO){
11     if (sp.psm == P_NEAREST){
```

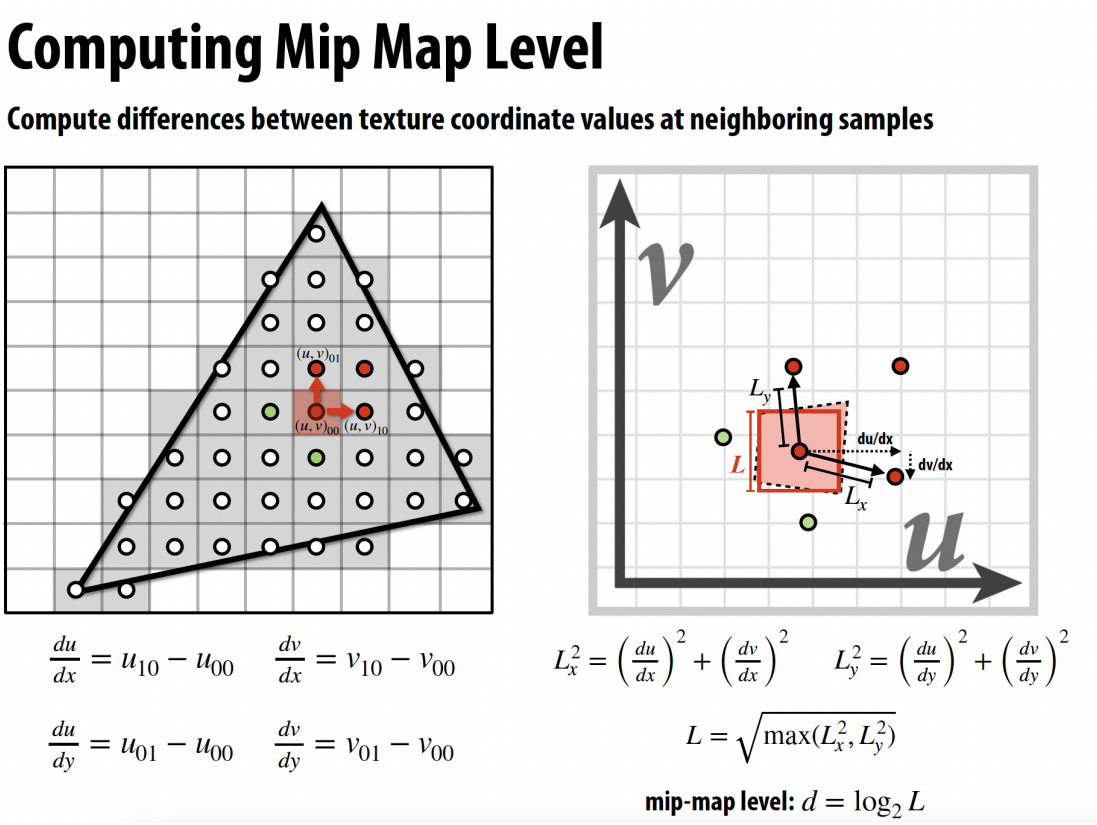


Figure 23: Mipmap calculation.

```
12        col = sample_nearest(sp.p_uv, 0);
13      }
14      else if(sp.psm == P_LINEAR){
15        col = sample_bilinear(sp.p_uv, 0);
16      }
17    }
18    else if (sp.lsm == L_NEAREST){
19      if (sp.psm == P_NEAREST){
20        col = sample_nearest(sp.p_uv, nearest_d);
21      }
22      else if(sp.psm == P_LINEAR){
23        col = sample_bilinear(sp.p_uv, nearest_d);
24      }
25    }
26    else if (sp.lsm == L_LINEAR){
27      Color col_up, col_down;
28      if (sp.psm == P_NEAREST){
29
30        col_up = sample_nearest(sp.p_uv, ceil_d);
31        col_down = sample_nearest(sp.p_uv, floor_d);
32        col = (1 - alpha) * col_down + alpha * col_up;
33      }
```

```
34        else if(sp.psm == P_LINEAR){
35          col_up = sample_bilinear(sp.p_uv, ceil_d);
36          col_down = sample_bilinear(sp.p_uv, floor_d);
37          col = (1 - alpha) * col_down + alpha * col_up;
38        }
39      }
40  // RETURN MAGENTA FOR INVALID LEVEL
41      return col;
42    }
```

6.1 Results

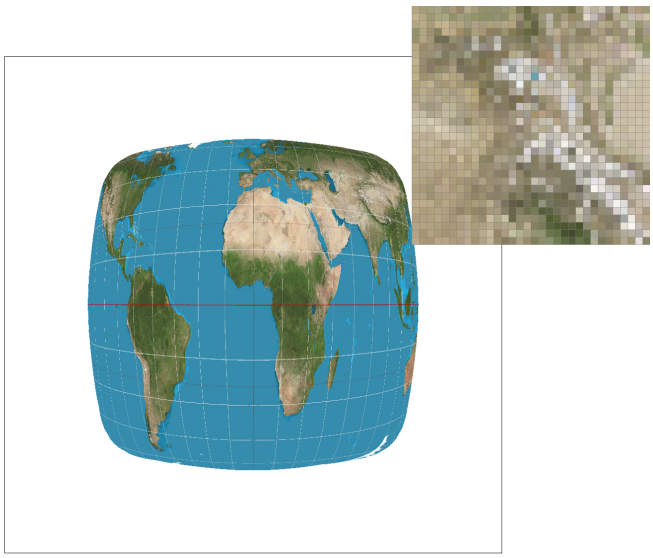


Figure 24: Mipmap::Zero level sampling algorithm showcase.

6.2 Additional Results

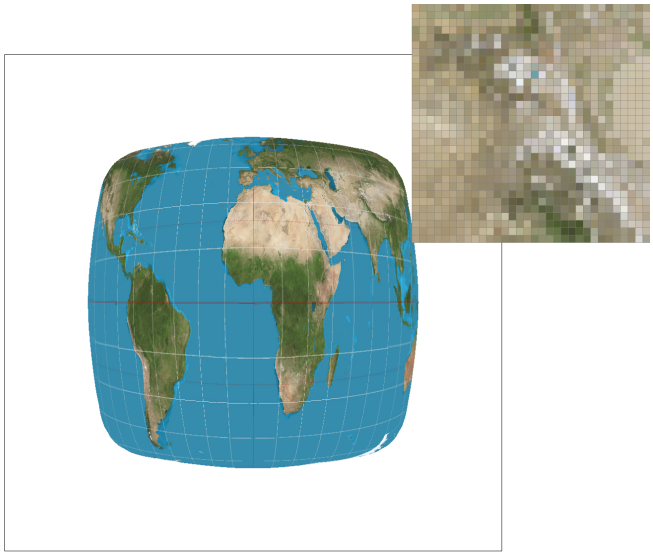


Figure 25: Mipmap::Nearest level sampling algorithm showcase.

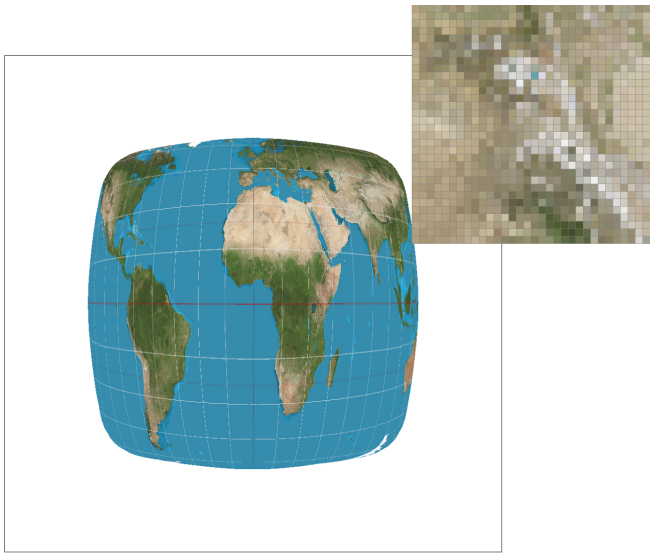


Figure 26: Mipmap::Bilinear sampling algorithm showcase.

Figure 27: Nearest level sampling algorithm showcase.

Figure 28: Bilinear sampling algorithm showcase.