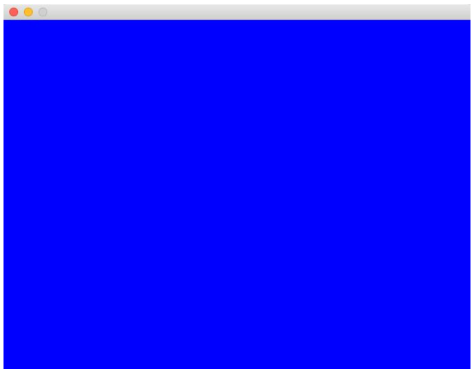
Rasmus Fredrikson

DH2323 LAB 1

2. Introduction to 2D Computer Graphics

2.1 Color the Screen



To change the color of the screen I changed the code
`vec3 color(0,0,1)`

0,0,0 gives black. 0,0,1 gives blue. RGB values. We can look up RGB codes to get other colors.

2.2 Linear Interpolation

The interpolation formula is:

$$y = y_0 + (x - x_0) * \frac{y_1 - y_0}{x_1 - x_0}$$

So the code for the interpolation is:

```
for (int i = 0; i < result.size(); i++) {
    result[i] = a + (b-a)/(result.size()-1)*i;}
```

When changing to vectors, we have x, y and z instead of only one value. We add them to the interpolation.
The code is:

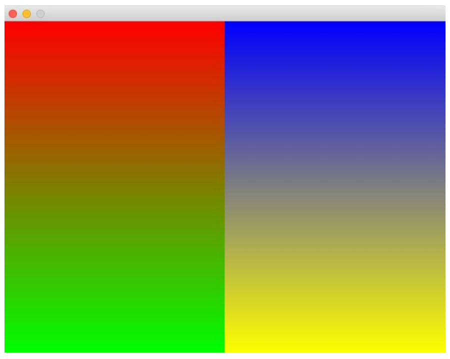
```
for (int i = 0; i < result.size(); i++) {
    result[i].x = a.x + (b.x-a.x)/(result.size()-1)*i;
    result[i].y = a.y + (b.y-a.y)/(result.size()-1)*i;
    result[i].z = a.z + (b.z-a.z)/(result.size()-1)*i;}
```

Problems: finding the right way to write the interpolation formula. It calculated in the wrong direction at first.

The following code is used for the special case where vector is 1:

```
if (result.size()==1){
        cout << "vector has to be larger than 1";
        return;}
```

2.3 Bilinear Interpolation of Colors

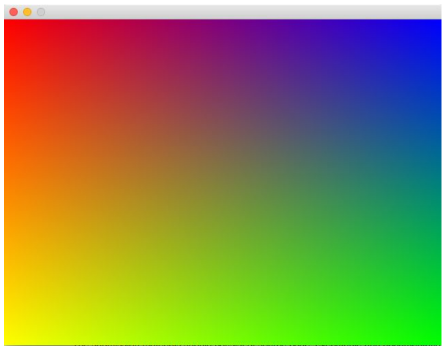


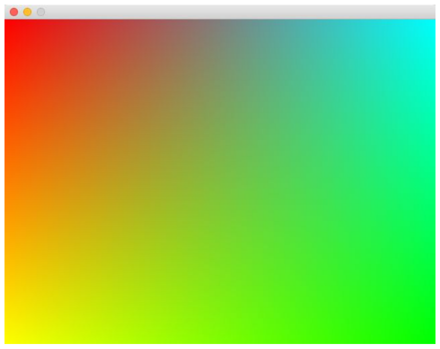
First I only interpolated and drew topLeft to bottomLeft and topRight to bottomRight.
This gave me the following picture:

A problem I had here was that I first tried using x instead of y in the code.

I then tried interpolating for each row to get the left and right to mix. At first, I had the interpolation inside the second for-loop but that made everything much slower. Moving the interpolation outside gave the desired result, to the right.

Changing places of the green and yellow gave the left picture below, and when I played around with the colors a little the first result I had was the picture below to the right.





3 Starfield

To make random initial positions between -1 and 1 for x and y, we have the following code:

```
        float r = -1 + float(rand()) / float(RAND_MAX)+float(rand()) / float(RAND_MAX);
        stars[i].x = r;
        r = -1 + float(rand()) / float(RAND_MAX)+float(rand()) / float(RAND_MAX);
        stars[i].y = r;
```

To make random initial positions between 0 and 1 for z, we have the code:

```
        stars[i].z = float(rand()) / float(RAND_MAX);
```

3.1 Projection - Pinhole Camera

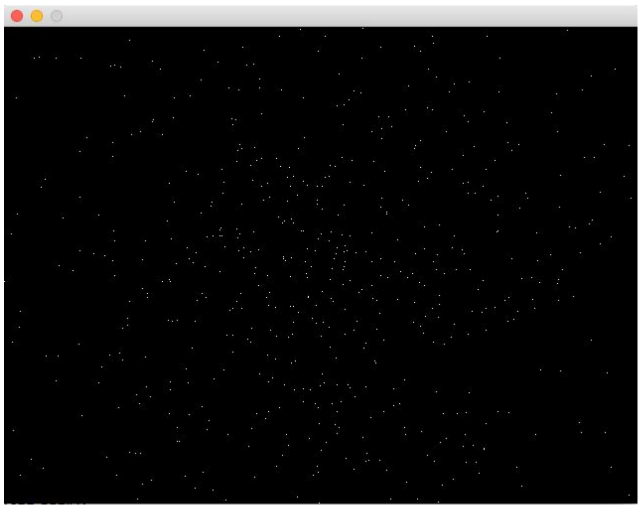
Vertical field 90
Horizontal field W/2

First we have black screen, with the code given in the lab instructions (`SDL_FillRect( screen, 0, 0 );`).

The code in the for-loop to draw the white stars is:

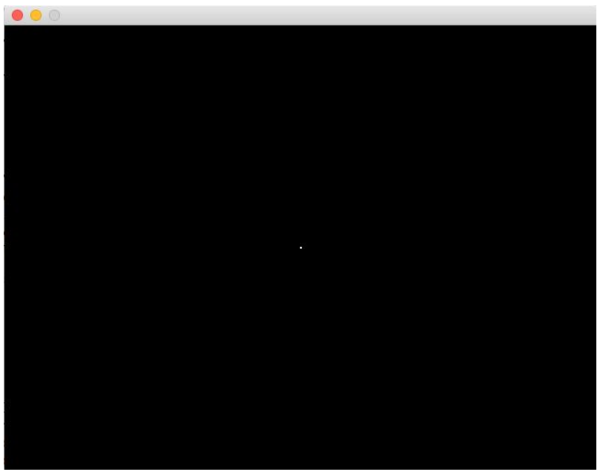
```
for( size_t s=0; s<stars.size(); ++s ) {
        float u = (SCREEN_HEIGHT/2)*(stars[s].x/stars[s].z) + (SCREEN_WIDTH/2);
        float v = (SCREEN_HEIGHT/2)*(stars[s].y/stars[s].z) + (SCREEN_HEIGHT/2);
        vec3 color = 0.2f * vec3(1,1,1) / (stars[s].z*stars[s].z);
        PutPixelSDL(screen, u, v, color);
        }
```

u and v are calculated according to equation 3.
The following is drawn:



3.2 Motion

Only z needs to be updated, because u and v are updated automatically, since they both are determined by calculations in which z is a value.



First I made the stars move away and disappear by just adding 0.1 to x, y and z.
I then made them respawn when they were outside of the scope from equation 2. However, when adding dt and the velocity to make the stars move, I had a few problems and the stars acted very weird, for example by just having one white pixel in the middle, as seen in the picture to the right. I believe a problem was the values being very small and therefore the stars being too far away.

In the end, I understood that I only needed to update z, and I changed the code making the stars respawn if they were outside of the scope. The code in my Update() function is:

```
void Update() {
        int t2 = SDL_GetTicks();
        float dt = float(t2-t); //Time since last update
        t = t2;
        float v = dt*0.0005;

        for( int s=0; s<stars.size(); ++s ) {
        stars[s].z += v*(-1 + float(rand()) / float(RAND_MAX));

        if( stars[s].z <= 0 )
                stars[s].z = 1;
        else if( stars[s].z > 1 )
                stars[s].z = 0;
        }
}
```

The velocity we chose was 0.0005, after having tested different values.

I also added the code `(-1 + float(rand()) / float(RAND_MAX))` when updating z, because I thought it would be cool for the stars to not all have the same velocity. We discovered that this gave a cooler 3D effect.

In full screen, my final product was the following: