DH2323 Computer Graphics and Interaction

Lab 1: Set-up and Introduction to 2D and 3D Graphics

Ramona Häuselmann

ramonaha@kth.se

April 3, 2021

1 Setup

I'm using my own computer with Ubuntu 18.04, Visual Studio Code and Cmake to complete the labs.

2 Introduction to 2D Computer Graphics

2.1 Color the screen

I experimented with the color model by setting the color to

1. red: `vec3 color(1, 0, 0)`
2. green: `vec3 color(0, 1, 0)`
3. blue: `vec3 color(0, 0, 1)`
4. yellow: `vec3 color(1, 1, 0)`
5. magenta: `vec3 color(1, 0, 1)`
6. black: `vec3 color(0, 0, 0)`
7. white: `vec3 color(1, 1, 1)`

2.2 Linear Interpolation

I wrote the function `void Interpolate( float a, float b, vector<float>& result );` using the formula for linear interpolation: To interpolate between two points x_0 and x_1 we use

$$f(x) = \frac{x_1 - x}{x_1 - x_0} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1) \tag{1}$$

The range of `x` is given by the size of the result vector: $x \in [0, result.size() - 1]$.
To catch the special cases if the result vector has size 0 or 1 I return immediately if the size is 0 and I return the average of `a` and `b` in case the size is 1. Otherwise I use equation (1).
I tested my implementation with the values and output given in the assignment instructions. To do that I wrote the function `void TestFloatInterpolate()`.
To implement `void Interpolate(glm::vec3 a, glm::vec3 b, std::vector<glm::vec3>& result)` I used the same approach and applied it to each dimension of the `vec3`. I tested my implementation with the values and output given in the assignment instructions. To do that I wrote the function `void TestVec3Interpolate()`.

2.3 Bilinear Interpolation of Colors

```
glm::vec3 topLeft(1,0,0); // red
glm::vec3 topRight(0,0,1); // blue
glm::vec3 bottomLeft(1,1,0); // yellow
glm::vec3 bottomRight(0,1,0); // green
```

```
glm::vec3 topLeft(1,0,1); // magenta
glm::vec3 topRight(1,1,1); // white
glm::vec3 bottomLeft(0,1,1); // cyan
glm::vec3 bottomRight(0,0,0); // black
```

3 Starfield

3.1 Static Starfield

To create a static starfield in `main` I first initialize all star positions randomly in the given range. Then I project the x and y positions with the equations of the pinhole camera.

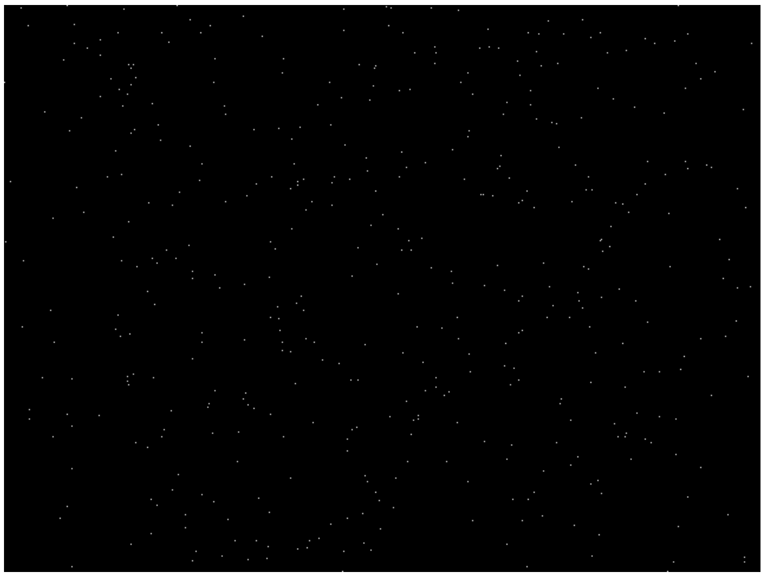

Figure 1: static starfield

The viewing angle can be calculated with equation (2):

$$tan(\frac{\alpha}{2}) = \frac{L}{2f} \tag{2}$$

where `f` is the focal length and `L` is the length of the image. So if we use $f = H/2$ and want to compute the horizontal field of view we get $tan(\frac{\alpha}{2}) = \frac{W}{2f}$ where W is the screen width. We use H=480 and W=640. If we calculate we get $\alpha = 106.26°$.

3.2 Motion

To create the motion effect in the `Upddat()` function I update each star's distance value with
`stars[s].z -= VELOCITY * dt / 1000.0f; //use dt in seconds, velocity is in m/s.`
`dt` is in milliseconds, therefore I divide by 1000.0f so that I can choose the velocity in `m/s`. For the velocity I chose a value of 5. With that I get a reasonable moving starfield effect.