Lab 1 report

Victor Hansjons Vegeborn
victorhv@kth.se

DH2323 Computer Graphics and Interaction
KTH Royal Institute of Technology

The labs were set up in XCODE 7.2.1 on my personal mac book pro by following the guide provided by Chengqiu Zhang on the course web.

1 Lab part 1: interpolation

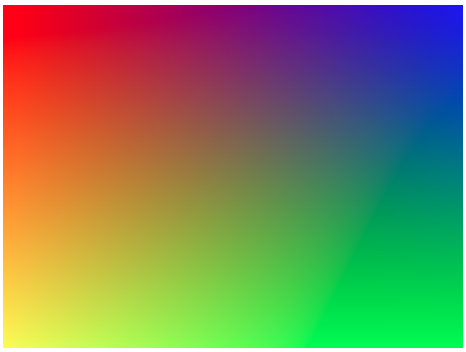

Fig. 1: The resulting still image rendered using interpolation.

The first part of this lab covered interpolation and how it can be utilized for rendering 2d images like the one above. As a color is defined using a vector $\mathbf{c} = (r, g, b)$ where the rgb-values are set between 0 and 1, we can interpolate between two values to get a color gradient.

The approach in this lab was to define the colors of the four corners in the image i.e red, blue, yellow and green, then interpolate color vectors, first from the red to the yellow corner, and then from the blue to the green. This data of color vectors represent the color gradients for the most left and most right pixel columns. By iterating through this data row by row, the pixels for each row could be interpolated and finally be drawn onto the screen.

No problems was encountered in this part of the lab. Writing the interpolation function, and handling the special case when the result vector (confusingly,

the name for the data structure that held all color vectors in this lab is also called vector within c++) size was one, was handled by returning the mean average of the two color vectors. If $\mathbf{a}, \mathbf{b}$ are color vectors and n is the number of points in the final data structure, the interpolation function is defined mathematically by:

$$Interpolate(\mathbf{a}, \mathbf{b}, n) = \begin{cases} \mathbf{a} + \left(\frac{\mathbf{b}-\mathbf{a}}{n-1}\right) i & \text{if } n > 1, 0 \leq i \leq n \\ \frac{\mathbf{a}+\mathbf{b}}{2} & \text{if } n = 1 \end{cases}$$

2 Lab part 2: Starfield

2.a The pinhole camera and projection

In this part of the lab, a transformation $T : \mathbb{R}^3 \mapsto \mathbb{R}^2$ takes 3d-points (or vectors) and projects them onto a view plane. The perspective that is given is modeled after a pinhole camera. For a given 3d-point $\mathbf{p}^i = (x^i, y^i, z^i)$, where f is the cameras focal length, W, H is the camera width and height, the transformation to a 2d-coordinate (u^i, v^i) is defined by:

$$(u^i, v^i) = T(x^i, y^i, z^i) = \left(f\frac{x^i}{z^i} + \frac{W}{2}, f\frac{y^i}{z^i} + \frac{H}{2}\right)$$

This works by letting the axis u, v and x, y be oriented in the same direction in which it is possible to calculate the axis offset u^i by multiplying the distance to the view plane with the ratio between x^i and z^i. Furthermore, to keep the screen pixel origin at the top-left corner, the offset of $\frac{W}{2}$ is added to u^i, and $\frac{H}{2}$ to v^i respectively. In figure 2 below, the relationship between 3d-point and the view plane is presented in a more comprehensive way.

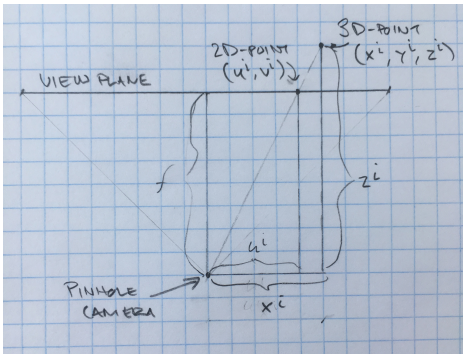

Fig. 2: The transformation T works by utilizing the geometrical relationships between $x^i : z^i$ and $u^i : f$.

2.b Moving the stars

The stars move along the negative z-axis. This gives the effect of moving forwards and passing by the stars. This was achieved by first initializing the stars at random positions within the dimensions specified in the lab, and then update every stars z-position by adding a constant velocity. As the execution time of the program varies due to system load, the velocity is multiplied by the relative time between each rendered frame. This makes the movement of each star appear smooth. By wrapping the z-component of each star within the zero-to-one-space i.e keep the z-component within 0 and 1, an infinite movement trough stars are achieved.

The above is calculated within the update-function. If z_t^i is the z-component of the i:th star, at time t, the time passed time between two frames is dt and the velocity v is in the positive z-direction, the next z-component of the i:th star is calculated by:

$$z_t^i = z_{t-1}^i - dt \cdot v$$

This works by subtracting the velocity (times the passed time dt) in the negative direction from the last frame position. Below is a still image from the working program.

Fig. 3: Still image of the resulting stars moving towards the camera, giving the illusion that the camera perspective is traveling towards the stars.

2.c Encountered problems

I encountered a problem while not initializing the variable that held the current time frame. The compiler used some initial garbage value when calculating the time difference between the first two frames, often with a very small value. When running the first frame, the positions of the stars were set way off and the projection rendered all stars at the center of the screen. This was easily fixed by initializing the current time variable.