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

Computer Graphics with Interaction DH2323

Author: Niklas Blomqvist, nblomqvi@kth.se
Lab partner: Annika Strålfors

May 26, 2014

Contents

1 Setting up the Environment

This lab consisted of three parts, setting up the lab environment, an introduction to 2D graphics and an introduction to 3D graphics. The hardest part was actually setting up the lab environment, more specifically to get the SDL to work properly. The source code in this lab is not consisting only of the final output as the other two labs, but are divided into two parts, the 2D part and the 3D part.

1.1 Simple DirectMedia Layer

Installing the Simple DirectMedia Layer, SDL, on my system running Ubuntu was a bit of a hassle. Originally I tried to install the newest version, the SDL 2.0.3 but after a couple of hours investigating problems with it I decided to settle for the SDL 1.2 version, which surprisingly worked on the second attempt.

2 Introduction to 2D Graphics

The program loops through all pixels in the canvas and with the help of SDL draws them with a specific color. Each vertex has a specific color and the program interpolates between these to create the following picture.

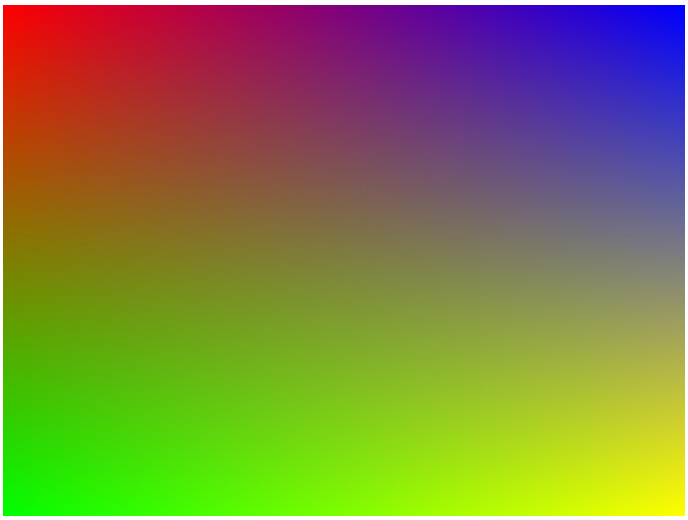

Figure 1: Bilnear Interpolation of Colors

We hit a bump at the beginning of this implementation. We couldn't get the program to compile properly. This was due to the fact that C++ needs functions to be declared before being called, whereas with Java, the language I am most accustomed with, this is not needed. After that, everything was pretty much straightforward. Source code can be seen in Appendix A.

The Interpolate function takes two vectors, interpolates between these vectors and adds the result in an array of a specific length, in this case, the width of the canvas(Appendix A, row 94). The interpolate function has a case for handling resulting vectors of length 1(Appendix A, row 97).

3 Introduction to 3D Graphics

In the last part of the lab we were suppose to create a 3D effect where it would look as though you were moving through space. This was created by updating the stars location on the canvas before rendering them making it look like they were moving towards you. The following picture shows how the screen would look in an instance, but during runtime the stars (the white pixels) would move.

Figure 2: The star field

We created a function that randomizes a float between two numbers(Appendix B, row 84). This is used for placing the stars randomly on the canvas. The stars are given imaginary coordinates in the three dimensional plane. These coordinates are then translated to the coordinates in the two dimensional plane(Appendix B, row 105-106). These coordinates(u and v) represents x,y coordinates in the canvas. Updating the z-coordinate will make it look like the stars are moving towards the camera in the z-plane but in reality, the x and y coordinates(u and v) changes. The rate at which the z-coordinate changes(Appendix B, row 136) determines the speed in which you move through space.

3.1 The Horizontal Field of View

In the assignment a question is posed, asking what the resulting horizontal field of view is if the vertical field of view is 90 degrees. We calculated it to be 106,26 with the formula found on wikipedia:

$$H = 2 * arctan(tan(\frac{V}{2}) * (\frac{w}{h}))$$

where H is the horizontal field of view, V is the vertical field of view, w is the width and h is the height.