3D Visualization of petroleum data

Project 2: Visualization using OpenGL Shader Language

Einar Baumann

November 13, 2013

1 Color mapping with splitter value and shaders

In this program an adjustable splitter value enables the user to "tune" the colors to highlight different parts of the seismic data (screenshots using different splitter values are shown in Figure 2). An illustration of a color scale using a splitter value is sown in Figure 1.

The fragment shader computes the color of each fragment from the splitter s value and an input "results" value v. The color is computed as follows:

$v \leq s$	Red	$\frac{v}{s}$
	Green	$\frac{v}{s}$
	Blue	1
$v > s$	Red	1
	Green	$\frac{1-v}{1-s}$
	Blue	$\frac{1-v}{1-s}$

If we had only used the fixed function pipeline instead of shaders, we would have needed to recreate the texture from new (adjusted) values every time the splitter value changed, and reupload it to the graphics card.

The main advantage of fragment shaders and not the fixed function API to change the colors, is that we only have to upload the texture to the graphics card once - the shader takes care of changing the colors depending on the splitter value instead. The splitter value is the only thing that has to be uploaded to the graphics card when the color is to be changed. This dramatically increases performance for larger textures, because the bus transferring data to the graphics card is far slower than the GPU itself.

Note that the value of v in the shader input variables will always be in the range $[0.0, 1.0]$ because the texture pipeline clamps all values to this range. So our data, with the range $[-128, 127]$, is first stored as unsigned bytes using the `unsigned char` datatype in the texture object. This adds 128 to each value, thus making the new range $[0, 255]$, where $0 = -128$ and $255 = 127$. The texture pipeline then clamps these values to the $[0.0, 1.0]$ range, where $0.0 = -128$ and $1.0 = 127$.

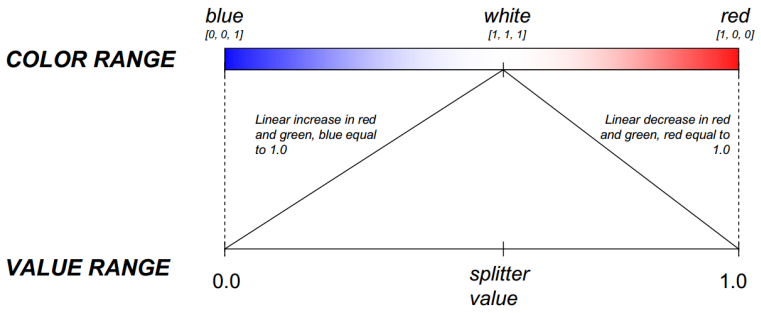
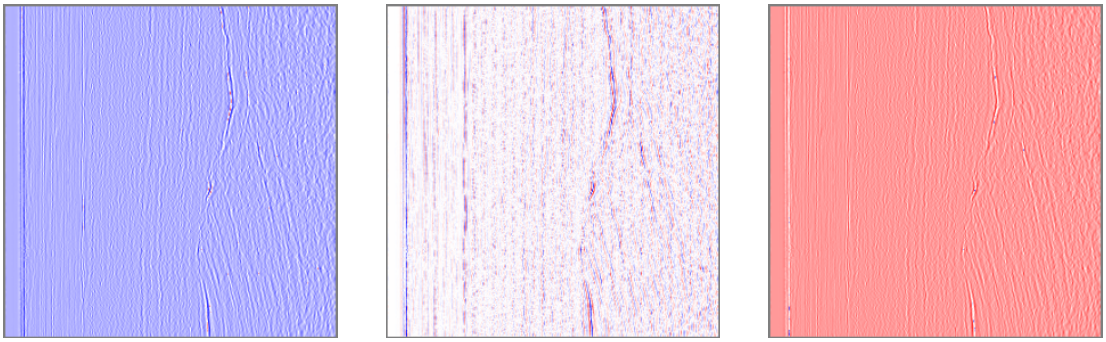


Figure 1: Illustration of a color scale utilizing a splitter value.

2 GL_LUMINANCE

One of the main benefits of using `GL_LUMINANCE` and not `GL_RGB` values as in project 1, is that each point can be defined by a single byte value instead of three. This means that the amount of data that needs to be transferred to the graphics card when using `GL_LUMINANCE` is one third of the amount that has to be transferred when using RGB.

`GL_LUMINANCE` also "guarantees" that the fragment shader recieves correct texture values in the domain $[0.0, 1.0]$ (as discussed in the previous section). The way I understand the converting the texture pipeline does (and this is not a deep understanding by any measure), clamping RGB-sets to a the $[0.0, 1.0]$ domain may be problematic; at the very least it complicates things.



(a) Split = 0.5 + 0.03. **(b)** Split = 0.5 **(c)** Split = 0.5 - 0.03

Figure 2: Here, one face of the cube is shown with different split values. Initially the split value in the program is 0.5; this is shown in 2b. Figures 2a and 2c show the same area, but with the split value incremented/decremented thrice.

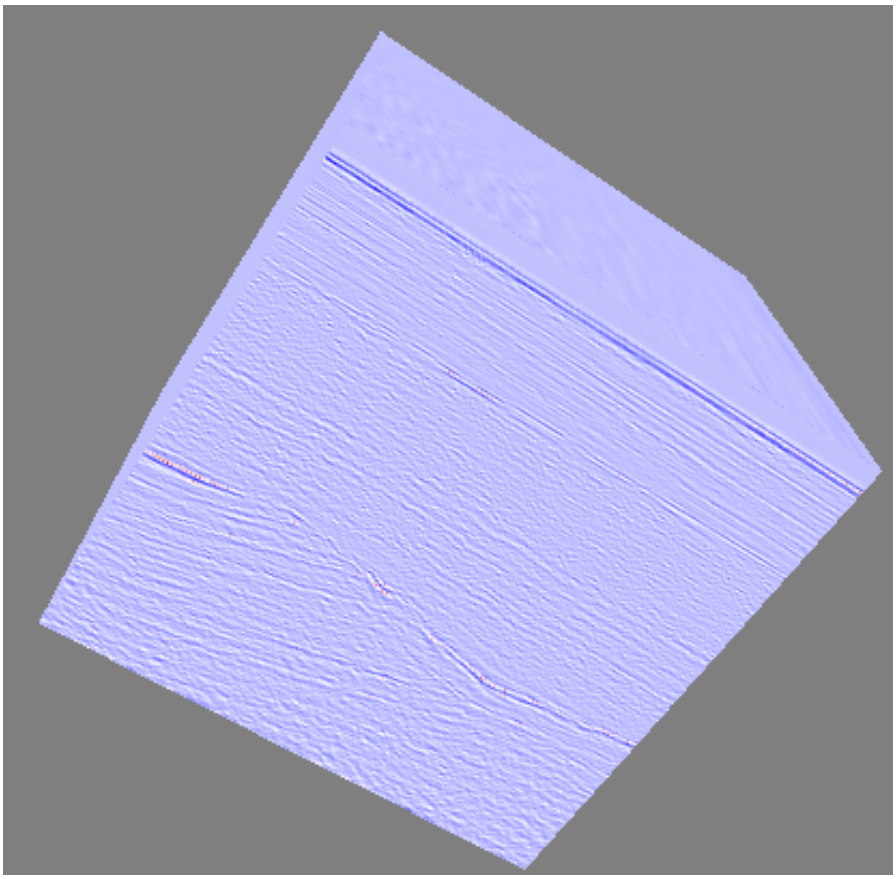


Figure 3: This figure shows the cube while rotating. The split value here is 0.5 + 0.02.