Visualizing Terrain Data through Ray Tracing

Daniel Aaron Salwerowicz, Christopher Kragebøl Hagerup

October 11, 2018

Introduction

Our goal in this project was to use raycasting to visualize terrain data of Bjerkvik in a 3D-space. The data was provided in the form of a text file with points, and our aim was to see if we could use colors to visualize different parameters in the 3D-model. Examples of parameters that we could use were *height* and *point density*.

Methods

We used GMlib to represent and render the terrain in 3D-space, and used a technique called ray casting to render the terrain to the window.

In our application, data is first read from a file. After all of the samples have been stored in the file, we determine the maximum and minimum values in the x-, y- (and z?)-axis. After we have found these values, we use them to normalize all of the data to our space, where we partition the volume into samples, and determine the amount of points in each sample. This amount is used to determine the color and alpha (transparency) of the sample.

Sans and Ramona [1] defines ray casting as a technique for rendering three-dimensional shapes without having to reconstruct the data. For each pixel in the image, a ray is cast into the canvas, and the models in the picture are visualized by having the light interact with the samples from the data. Unlike the similarly named ray-tracing, ray casting does not trace rays recursively, and instead only does it once. The benefit of using ray casting is that the work is light and can easily be panellised, making it ideal for a Graphics-Processing Unit (GPU).

We also make use of a transfer function in our code. A transfer function is a function that determines the color of the different samples in the model based on one or more parameters. In our case, the transfer function highlights the point density. In addition, the transfer function is vital for making sure that the volume is displayed correctly, as the shape will not be mapped properly without it

Results

Analysis

Discussion

Bibliography

[1] Fransisco, S. and Rhadamés, C. (2017). A comparison between gpu-based volume ray casting implementations: Fragment shader, compute shader, opencl, and cuda. *CLEI Electronic Journal, Volume 20, Number 2*.