Visualizing Terrain Data through Ray Tracing

Daniel Aaron Salwerowicz, Christopher Kragebøl Hagerup

October 11, 2018

Acknowledgements

We would like to thank Børre Bang and Aleksander Pedersen for giving us the initial foundations for the code of the project, and for helping us with setting up the project and volumetric visualization.

We would like to thank Kent-Arne Larsen, Olav Kjartan Larseng and Hans Victor Anderson Lindbäck for helping us understand how to apply colours to the transfer function, and how to apply this to the GUI.

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 through feature scaling, project it into our volume, 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. The transformation we use for the data is detailed below:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} * DIM$$

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

Our result is a program where you can see the terrain visualized. The visualization can be rotated and viewed from different angles, and you can easily make out the terrain (Even though it is a bit stretched at the moment). On one of the sides of the GUI (right or left), there are four coloured boxes, which provide the colour points for the transfer function. Clicking on the boxes adds a point with the colour clicked to the transfer function, and you can click a box multiple times to add different gradients of a colour. Our terrain visualization code is bundled with the curve visualization, and can be enabled by pressing 'T' in the finished program.

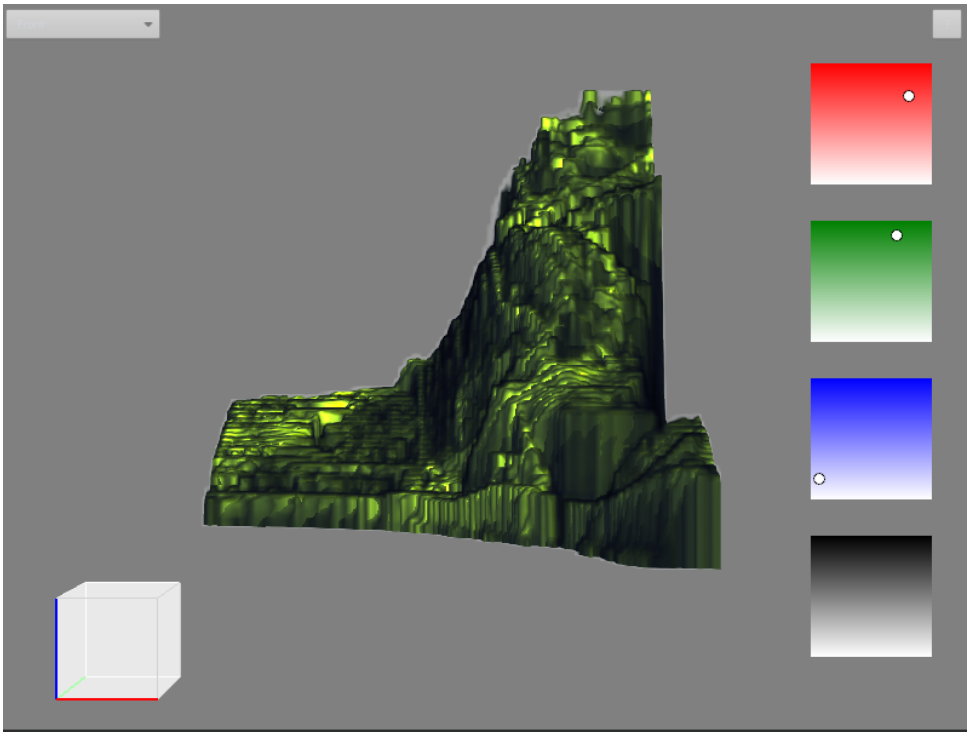

Figure 1: Current ray cast of the points in our program, with brighter terrain indicating higher point density

Analysis

From our interactions with the teachers and observations, we conclude that we have managed to render the terrain properly with ray casting, although it currently is stretch

quite a bit vertically. The colours are also visualized properly, according to the densities we set, but density is the only parameter we can visualize with colours right now.

We originally wanted to change the colours by the use of a drop-down menu, but after discussing with Alekander, we agreed to use the coloured boxes instead, as this proved the most intuitive for the end user.

Discussion

So far, we are happy with the result of our work, which is the visualizer. Setting colours for the transfer function feels intuitive in our opinion (as you only need to click the colored squares to set the colors). It also runs moderately fast, with reading the data from the disk taking the longest amount of time (although it reads the entire file in 5 seconds on our computers, so it is not exactly slow). Meanwhile, transforming the data to fit our model and rendering it takes a fraction of a second, so we are quite content with the performance.

We have several opportunities for further development in our visualization. One such opportunity is to change the ray tracing and transfer function to render colours based on other things, such as the height of the map, or some other volumetric data. Another thing we could add if given a bit more development time is functionality for removing individual points from the transfer function (as you can only remove the points all at once right now).

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.*