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 let user change the transfer function to visualize the volume data in different ways.

Methods

We used GMlib and some start code made by Aleksander Pedersen to represent and render the terrain in 3D-space, and used a technique called ray casting to render the terrain to the window. Ray casting uses rays sent from camera onto every pixel of image space to represent the volume data in object space.

In our application, we read data from file. After reading all samples stored in the file, we determine the maximum and minimum values in the x-, y- and z-axis. Using these values, we 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. WE then use the number of points in each cell to determine the color and transparency values of the cell. Formula used for transforming each coordinate of point is:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} * \text{Cell Count}$$

Cell count refers to number of cells along each axis (100 in our case). 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 space, a ray is cast from camera into the object space. Then it

records where it meets density data of the volume and using various transfer functions renders the volume to image space. Unlike the similarly named ray-tracing, ray casting does not trace rays recursively. The benefit of using ray casting is that the work is simple and can easily be parallelized, making it ideal for a Graphics-Processing Unit (GPU).

Transfer functions can be used for various purposes, in our case we used it to dynamically change colours and transparency of the volume. GUI we have set up for that lets user choose any number of points they would like on the four colour squares and see how they change the volume as well as to reset the points and volume. Code overview for our project can be found in figure 1.

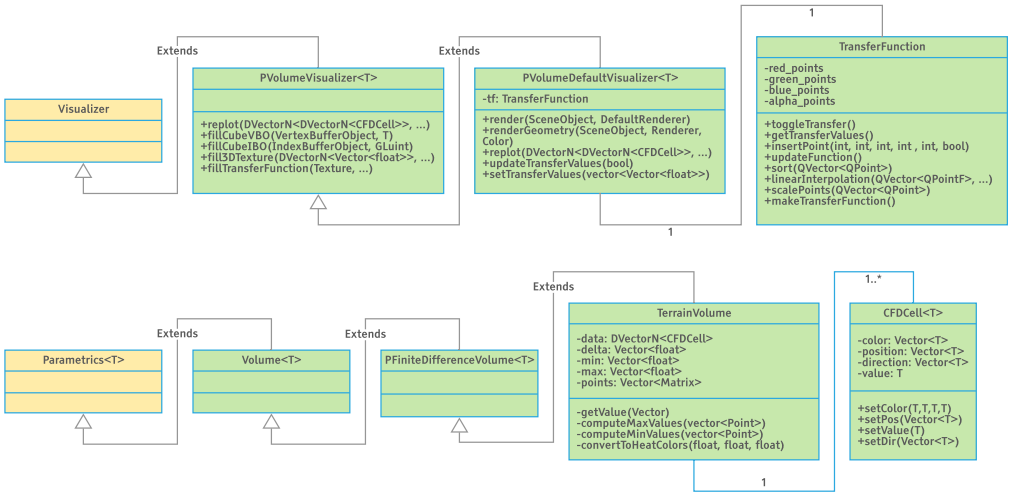

Figure 1: UML class diagram our program.

Results

Our program lets users see the terrain volume and change the transfer function on the go. The visualization can be rotated and viewed from different angles to see the terrain volume better. User can also set a colour or transparency value for the volume. One can also reset the points to default value. Example of the visualized data can be seen in figure 2.

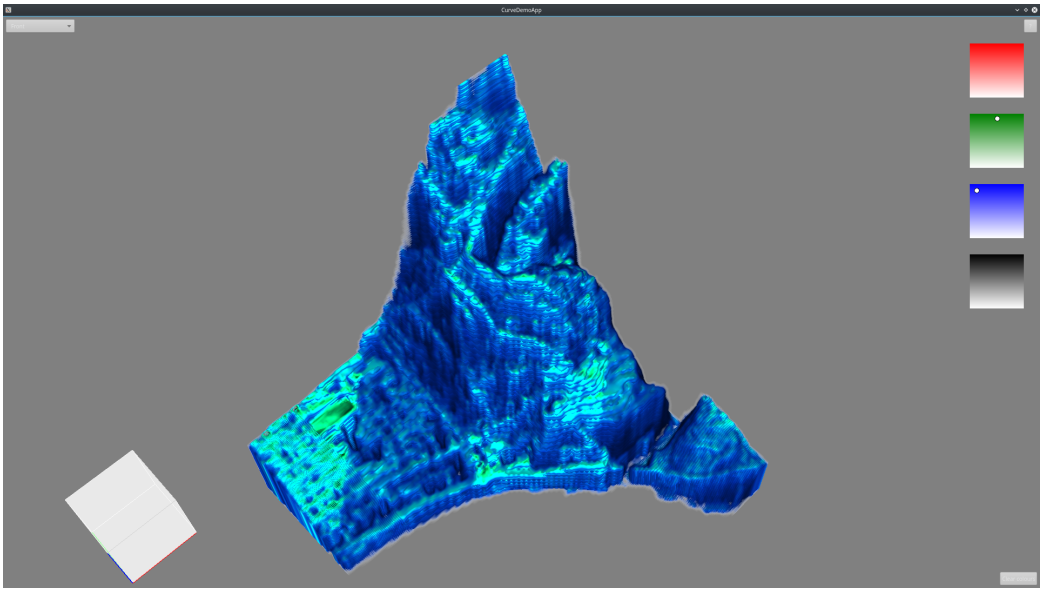

Figure 2: Visualization of volume data, with brighter terrains indicating higher point density.

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

One problem we have with the data provided to us is that it's only a surface data, so we cannot properly render the volume. As we do not know the point density for cells under the surface.

Discussion

Overall we are quite happy with the result of our work. 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 quite fast, considering the high resolution we use. There are in total 1 million cells that our app draws and we have managed to run it with over 15 millions cells, though it takes about a minute or more to calculate and show the data. Normally reading the data, transforming it to fit our model and rendering it takes about three

seconds in total, 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 parameters. 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). 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.

Another improvement in program is to visualize the volume realistically, so that the volume is not stretched in z-axis, as well as getting more accurate data where we know the point density under the surface, so that we can set volume transparency properly.

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. As well as Jostein Bratli for helping us with GUI programming for GMlib.

We have also cooperated with our classmates Kent Arne Larsen, Olav Kjartan Larseng, and Victor Lindback and they helped us with some small problems that we had.

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.*