Real Time Visualization Task 2

Note: You have been given the same framework as before. This is so you don’t have to relearn the whole communication part. For obvious reasons, this means that there are some parts that are obsolete.

Linux implementations are allowed, but you might want to bring your own laptop to any presentation.

Important: Currently not all the data is loaded, as this results in an extremely long calculation time. The data is loaded in MainWindow::openFileAction(). Currently only the first 500 items are loaded. Remove the limitation for full testing. There is a “smalldata” file with 1000 items in it.

You will need to add CUDA to the project.

Your task:

There are 2 parts to the task: alterations on the client side and the CUDA implementation on the server side.

Server Side

On the server side there is an implementation of a KDE (see KDEEstimator2D). This needs to be ported to CUDA. This must done according to the paper included in the zip file of the framework.

Note: The paper assumes that there are same number of positions-of-interest (where the density function is evaluated) than input samples. In our case it is far from true. In addition, the pseudo-code of the optimized CUDA version assumes that the number of input samples are dividable by the block-size (BSIZE in paper), which is also not true for our case. The CUDA implementation must be adapted for this.

Look for processTextMessage(QString message) for communication on the server side (in case you had problems in the previous exercise).

Most of the work will be in streamserver.cpp and KDE2D.cpp

Client side:

1. The user must be able to pan and zoom on the image. There is already a very simple implementation for zooming which is an example only
2. A transfer function (color map) for coloring the data must be implemented in a fragment shader on the client side

The shader will look something like this

```html
<script id="vertexShader" type="x-shader/x-vertex">
    varying vec2 vUv;
    void main() {
    }
</script>

<script id="tfShader" type="x-shader/x-fragment">
    varying vec2 vUv;
    void main() {
    }
</script>
```

**Bonus Task:**

Create an isocontour map based on the KDE. Implementation is up to you. You will need to document how you did this.

**Alternative Implementations (OPenCL, python, Linux)**

Linux is not prohibited, but it may be a good idea to take your laptop along if you need to demonstrate something.

Python: Someone requested implementing the project in python on the server side. Again, this is not prohibited. Same as Linux, it is recommended to bring your computer along. The only support for this will be a sqlite database and a simple SQL query: ```sql = "SELECT DepDelay, ArrDelay FROM ontime WHERE TYPEOF(ArrDelay) IN ('integer', 'real') LIMIT 50000" .``` Alternatively you can load the binary files

**WARNING:** If you use any libraries, make sure to mention which ones and include an installation and how-to-run guide for your implementation.
OpenCL: There will be no support. If I have problems compiling/running it, expect to visit my office. Bring your computer to any demonstrations, and expect to be asked basic questions. Document your code extremely well (i.e. assume the person reading it does not know OpenCL). I have been able to run OpenCL examples from https://developer.nvidia.com/opencl and my graphics card is a geforce 980. I advise testing it out in the vislab so you have a computer to fall back on if necessary.