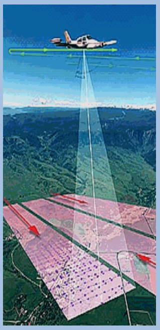
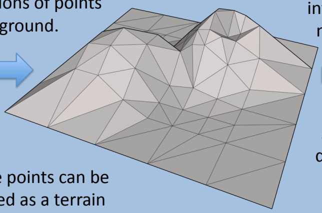
From Point Cloud to 2D and 3D Grids: A Natural Neighbor Interpolation Algorithm using the GPU

Alex Beutel



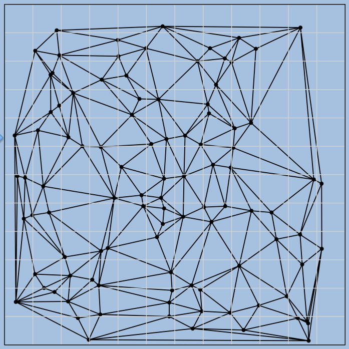
Planes scan the terrain using lasers to record the height of billions of points on the ground.



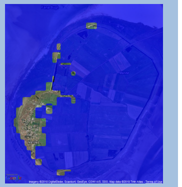
These points can be viewed as a terrain on the computer.

To use the data in geographic information systems we must model the terrain as a grid.

Because there is no structure to the scanned data, we must **interpolate** the height at each grid point. Doing this quickly for millions of grid points based on billions of data points is **our challenge**.



Once the data is modeled as a grid, we can use it to perform flood mapping.



To the right is the flood mapping of Mandø, DK based on two different resolution grids. Only the high-resolution grid shows the dike that prevents the island from flooding daily.

Problem:

With modern LiDAR technology the amount of topographic data, in the form of massive point clouds, has increased dramatically. One of the most fundamental GIS tasks is to construct a grid digital elevation model (DEM) from these point clouds. Our challenge is make an efficient, scalable algorithm that can construct high-resolution large-scale grid DEMs. We aim to do this for both spatial data as well as spatial-temporal data.

Previous work:

For grid DEM construction:

- Linear interpolation (Agarwal et al. 2005) – Simple, relatively fast, but not smooth
- Regularized splines with tension (RST) (Mitasova et al. 1993) – high quality, great with sparse data, but slow

Natural Neighbor Interpolation (NNI) on the GPU:

- Hoff et al. (1999) used the GPU to construct the Voronoi diagram
- Fan et al. (2005) used the GPU to perform NNI on 32 points simultaneously

Our approach and contributions:

Build high-quality, large-scale grid DEMs with a natural neighbor based interpolation scheme using the GPU

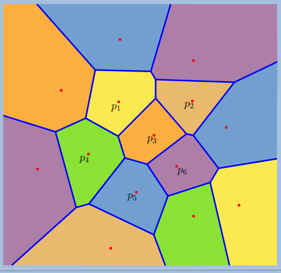
- Handle gaps in data by introducing the idea of *region of influence*
- Exploit the fact that we only interpolate at grid points using clever blocking. Handle 10^6 NNI queries in one pass. Previous maximum of ~32 [Fan et al. SIAM, 2005]
- Use CUDA to improve performance of our implementation
- Perform higher dimensional NNI on the GPU to construct grid DEMs in time based on spatial-temporal data

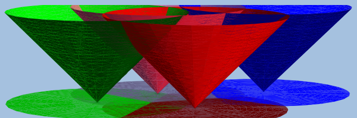
Our Results:

- Our algorithm can construct a high-resolution grid with 150 million cells from 2 billion data points in less than 37 minutes.
- Our algorithm takes approximately 2% of the time of RST and 10% of the time of linear interpolation
- For constructing 3D grids we find memory trade-offs that reduce the total running time by a factor of three.

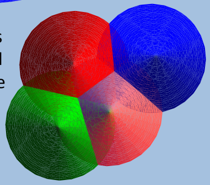
2D NNI for Spatial Data

We base our interpolation off of the Voronoi diagram, a division of space into Voronoi cells. Each Voronoi cell is the region for which a given input point is the closest input point.



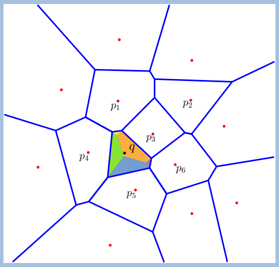


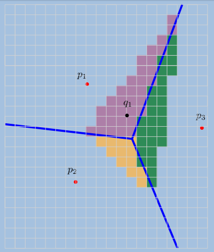
One way to check which point is closer is to draw cones from each input point and see where they intersect. The cone is the distance function and the intersection is the bisector.



If we use the graphics card to draw a cone for each input point and view them all from underneath we see a Voronoi diagram. This is called the lower envelope.

NNI: To interpolate the height at a point, we draw a Voronoi cell for the query and use the area stolen from surrounding Voronoi cells to compute a weighted average of the surrounding points' heights.



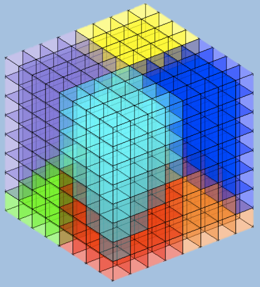


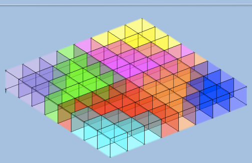
For the area we count pixels drawn on the graphics card. There are 73 pixels total in the Voronoi cell of q_1.

$$h(q_1)=(33/73)h(p_1)+(12/73)h(p_2)+(28/73)h(p_3)$$

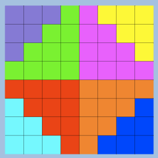
3D NNI for Spatial-Temporal Data

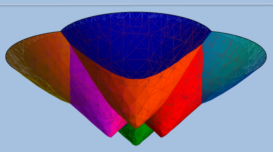
In 3D, a Voronoi diagram is the same general concept – a subdivision of space into Voronoi cells. Where we previously discretized space into pixels, we now discretize 3D space into cubes called voxels.



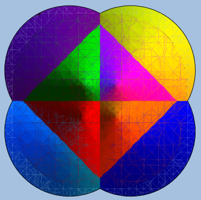


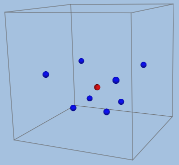
We can reduce the computation of a 3D Voronoi diagram to computing the 2D Voronoi diagram for each slice of voxels in time.



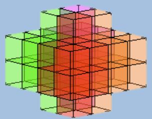


Since points are also offset in time, we not only draw cones but also hyperboloids. Again, looking from below reveals the Voronoi diagram.



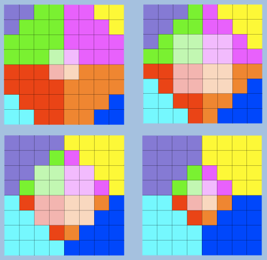


(a)



(b)

NNI in 3D is largely the same as in 2D. Upon adding a query (a), we analyze from which Voronoi cells it stole volume (b). To perform the weighted average we count the stolen pixels in each time slice and at the end find the height of the query, as shown below.



Special thanks to my advisor Pankaj Agarwal and mentor Thomas Mølhave. Additionally, thanks to COWI A/S and the Army Research Office for data.