What we need from the data store / preprocessor:

Needs to store tiles, each tile is same size, and square, but size is not decided. A tile can contain:

- Water (flat - renderer generates)
- Plain ice (a flat surface - again renderer generates)
- Static ice surface - store a height map as a grid of height values
- Edge data - store a sparse list of vertices (ordered). (Water generated at bottom)
- Crack data - like edge data, but later on we'll need to improve to store the movement of vertices over time

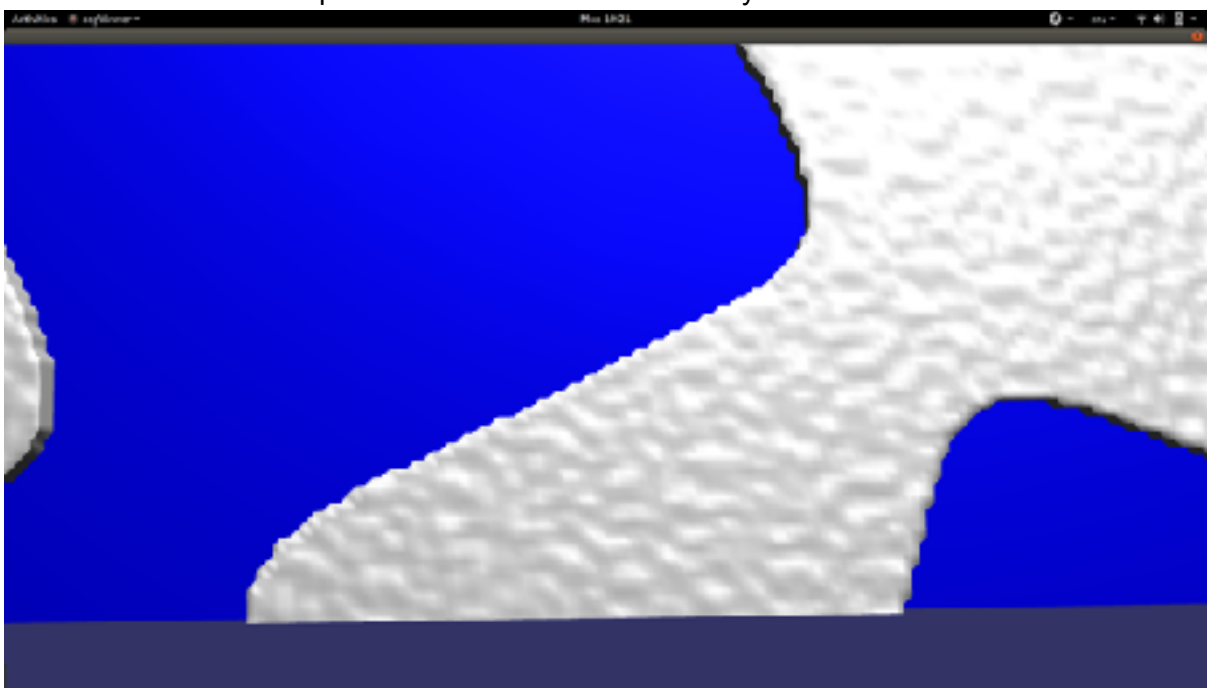
Ordered lists of vertices means that the vertices are ordered by one dimension and then the next, (i.e. if it's xy ordered, (3, 5) < (4, 1) < (4, 3) - ordered by x, and then by y)

The data store will be queried initially on the entire grid of tiles, and should return some data indicating how big the grid is, and what type of tile is where.
Then queried on tiles near a point, and should return all the data for tiles near that point.

Problem:

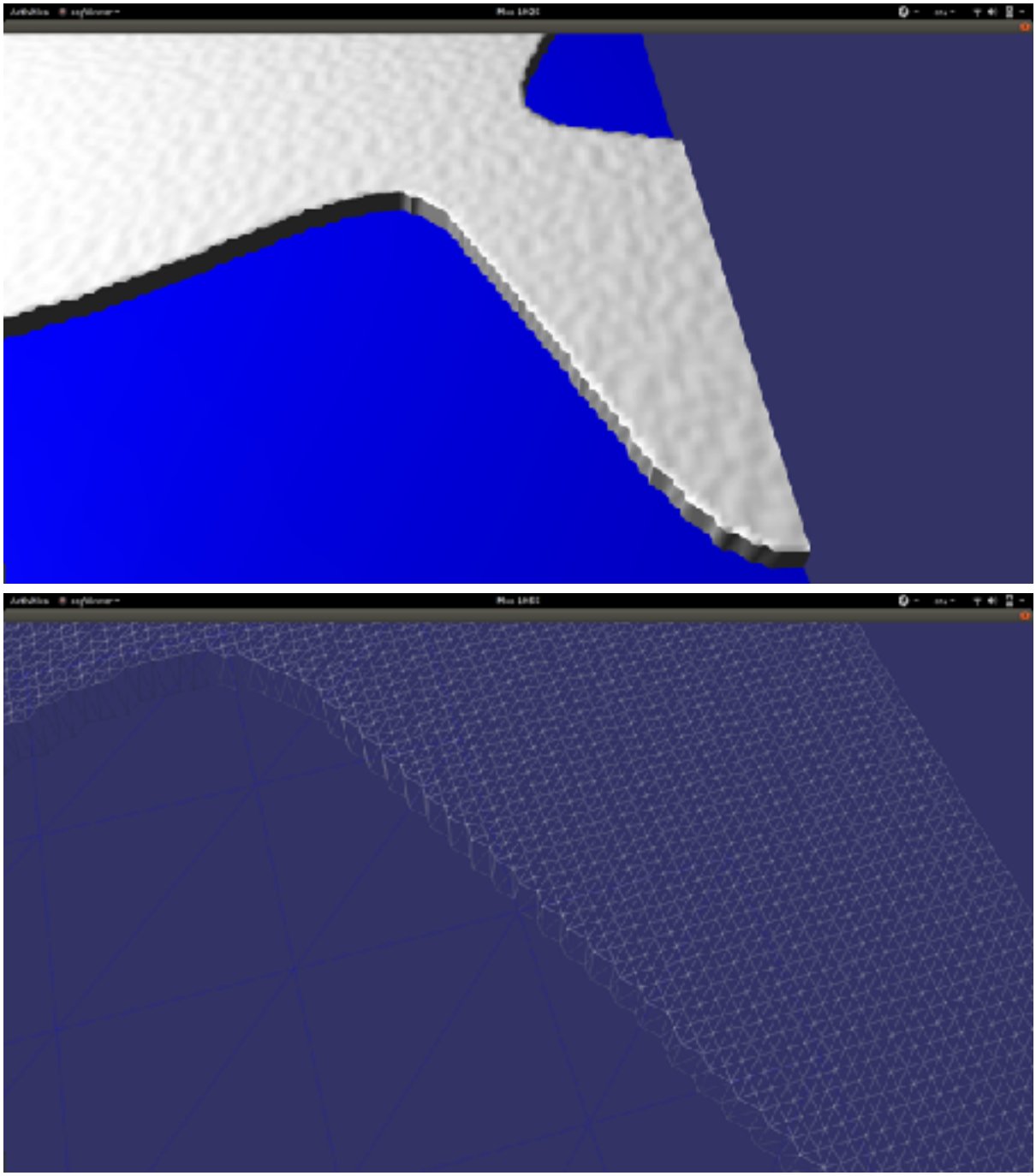
A heightmap for generic terrain can be represented as a grid of equally spaced heights, which can then just be converted into a grid of vertices.

- ~~When mapping terrain of the ice surface, this is ok, but gives noticeable lighting effects~~: This is a problem that can be sorted out by the renderer.

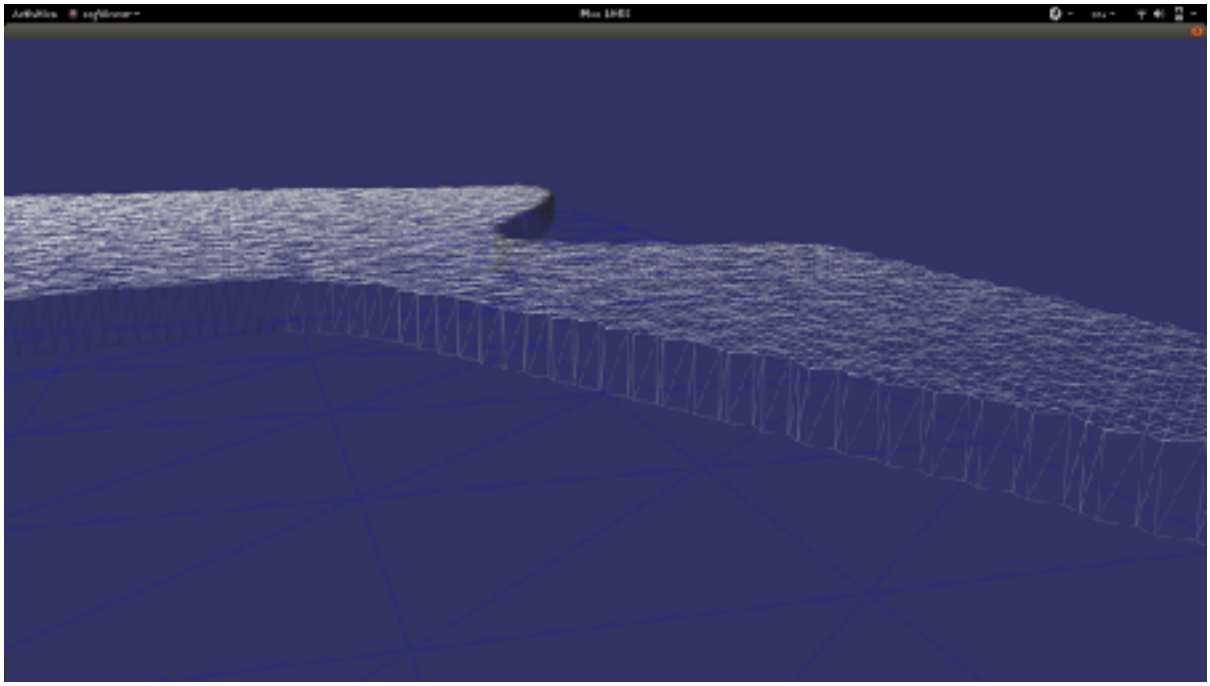


(this is good for the time being, as we're pressed for time, and the effects are more pronounced because I'm using simplex noise so it's very rough and random terrain.)

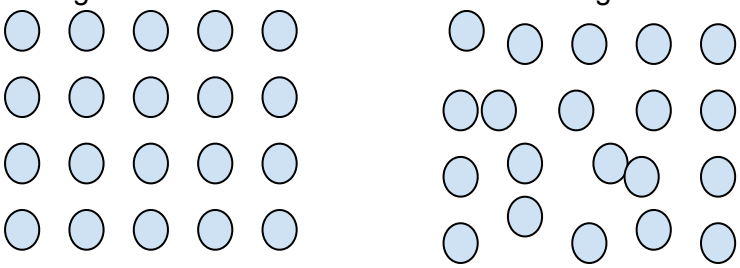
- Mapping edges is a little bit harder, as you can see by using a grid of heights the edges of the ice (here generated with simplex noise) are pretty rough. Heres another angle:



(Another problem with grid-height map can be seen here, in that it’s quite wasteful in renderer, but that is a problem that can be sorted by the renderer later)



An easy solution is to have, for edges such as these, a grid of vertices instead, so, on the left below is how the heights are arranged currently, and on the right, is an alternative arrangement which allows some smoother edges. But it's not really ideal for a crack



For the crack itself, we need to store a collection of vertices. Not really sure what the best way to store those is. Could have a sparse list of vertices, forming a surface (renderer does rest of work), in this sparse list, triangles would be formed by adjacent vertices (in xy plane (horizontal).

Current proposition for data store

- Store ice surface as grid of heights
- For edge of ice, sparse list of vertices
- For crack, sparse list of vertices
    - They change over time;
        - Store how they move
        - Vertices should be able to split or merge
        - Store time slices, a time slice shows what the vertices do in that time, where they move, how they split or move