Hierarchical Curve Skeletons of 3D objects on GPU

By: Rohan Juneja (2014156)
Alind Khare (2014011)

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY **DELHI**

Algorithm In a nutshell

1. After making 3D object (without holes), **boundary voxels** of the 3D object are identified as the source of the repulsive force field.

2. Compute the **repulsive force function** (Potential Field) at each object voxel.

3. Detect the **critical points** of the vector field and connect them using path-lines by integrating over the vector-field.

4. Compute the **divergence of the vector-field** at each voxel. Points with low divergence values are selected as new seeds for new skeleton segments.

5. Compute the **curvature at every boundary voxel** to compute the skeleton.

http://coewww.rutgers.edu/www2/vizlab/NicuCornea/Skeletanization/skeletanization.html

How Repulsive Force is Calculated?

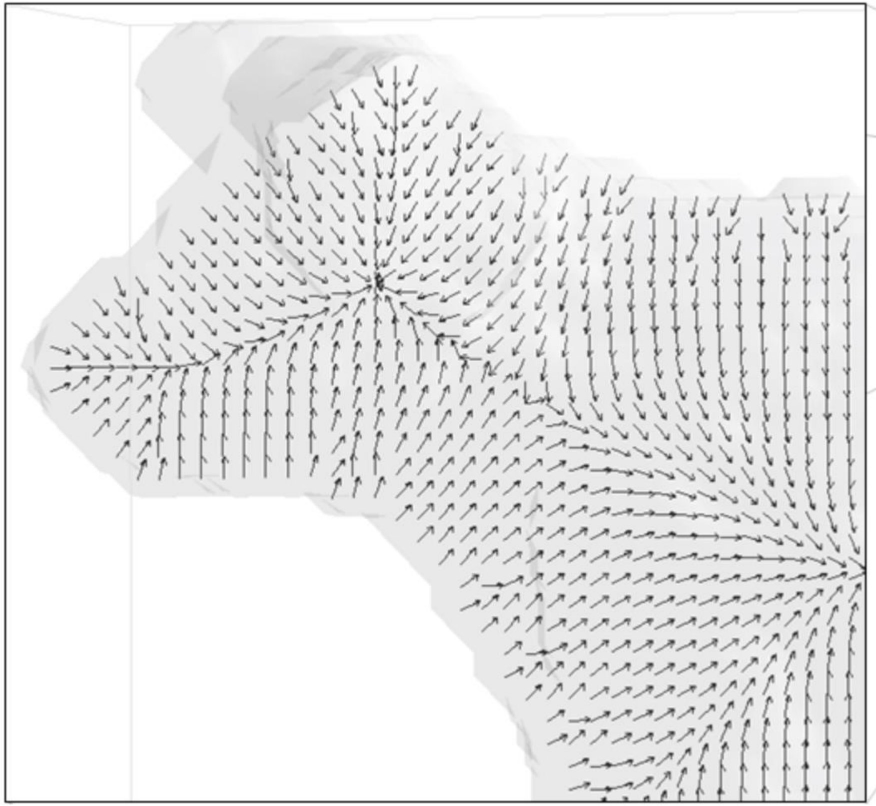
Repulsive force at a point due to a nearby point charge is defined as a force pushing the point away from the charge with a strength that is inverse proportional to a power of the distance between the point and the charge.

$$F_{pc} = CP/R^m$$

where CP is the normalized vector from C to P which gives the direction of the force, R is the distance between P and the charge C

Boundary voxel is considered as a point charge and repulsive force is calculated at each voxel.

Repulsive Forces



What are Critical Points?

Critical Points are points where the magnitude of the force vector vanishes.

The force field value is evaluated at each of the eight corners of a grid cell (voxel cell) using tri-linear interpolation. Cells containing both positive and negative values for every vector component (*x*, *y* and *z*) are potential candidates for containing critical points

There are three types of critical points:

attracting nodes (where all vectors are pointing towards the critical point)

repelling nodes (where all the vectors are pointing away from the critical point)

saddle points (where some vectors are pointing towards the critical point and others away from it)

Skeleton Formation

Saddles are a special type of critical points for the purpose of extracting the curve-skeleton from a 3D object

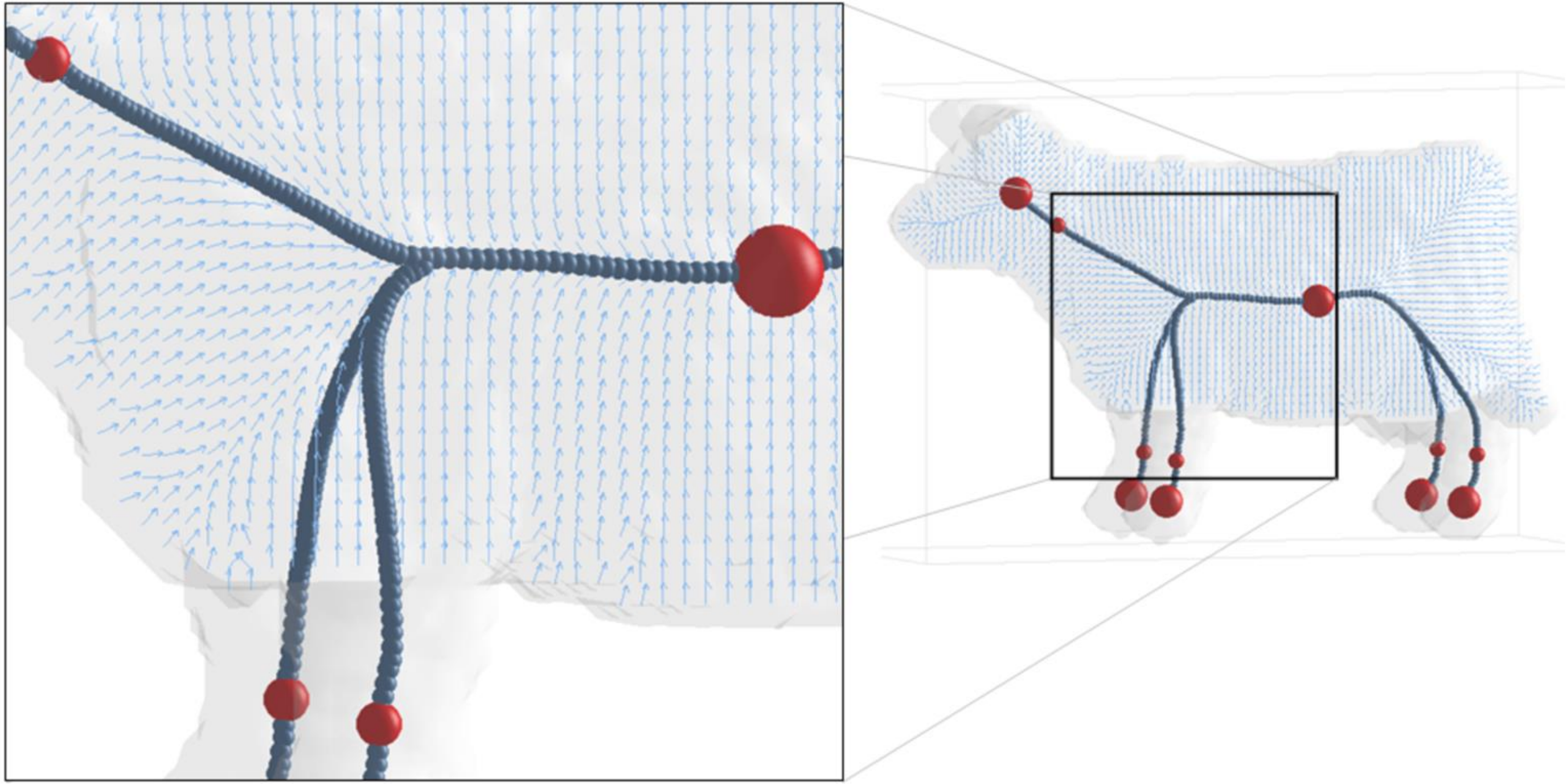
Path-lines are “seeded” from saddles in the direction of the eigenvectors corresponding to the positive eigenvalues.

Next, a path- line force-following algorithm is applied, which stops at another critical point or when it arrives at a previously visited location.

Samples taken along the integration path started at a saddle point form a *skeleton segment.*

Connecting all skeleton segments forms the *core skeleton*.

Critical Points and Skeleton formation



Initial Stats of Execution Runtime

Reading volume data	0 ms
Make Solid Volume	21 ms
Calculating Boundary voxels	2 ms
Calculating Potential Field	15687 ms
Detecting Critical Points	15 ms
Connecting Critical Points	1 ms

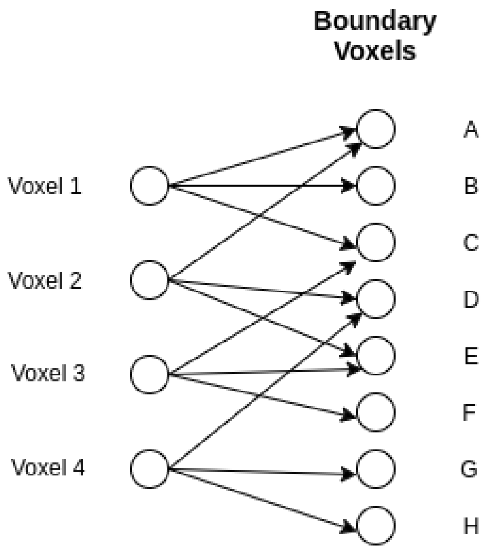
Potential field

Phases of Potential field

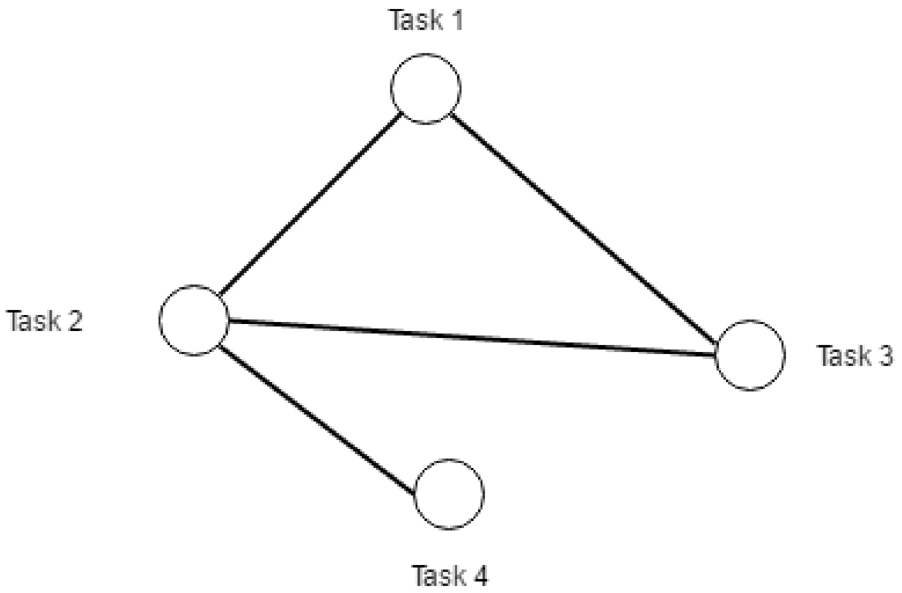
Finding the boundary voxels
Sorting the boundary voxels
Computing potential field for inside voxels
Normalizing force vectors for inside voxels
Computing potential field for boundary voxels

Parallelising code by providing each voxel to each thread.

Potential Field



Edge denotes neighbor of the voxel

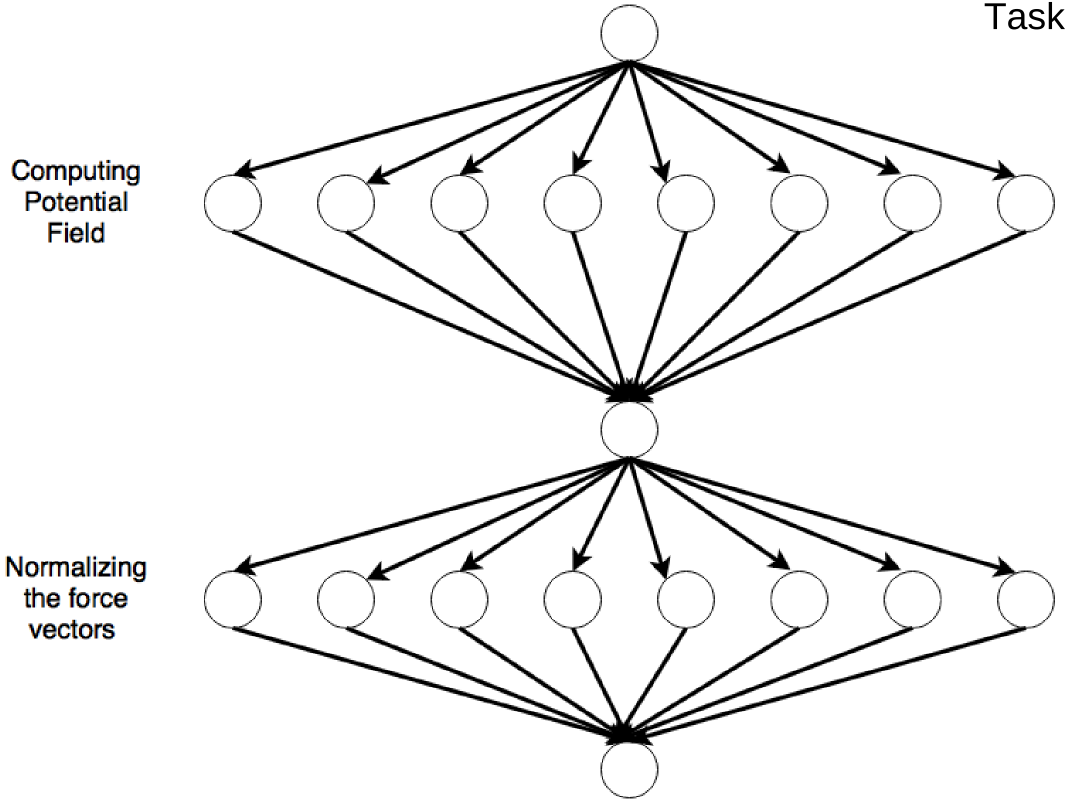


Task interaction graph

- Computing potential and normalisation of each vector is done independently by different threads.
- As per the figure, since voxel 1 and voxel 2 have the same neighbours, they interact while finding the potential field.

Potential Field

Task Dependency Graph



Sorting

Serial Code Sorting:

First the array of voxels gets sorted by z-plane values.

Then points having similar z-plane values gets sorted by y-plane values.

The points having same z-plane and y-plane values get sorted by x-plane values.

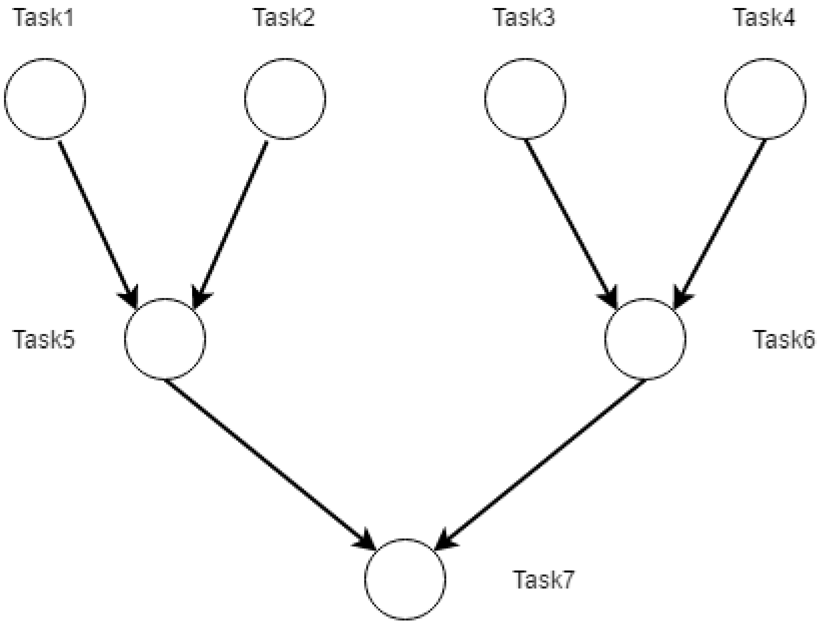
Parallel Comparator Function :

```
struct My_Comparator {
  __host__ __device__
  bool operator()(const VoxelPosition& o1, const
VoxelPosition& o2) {
      if( (o1.z-o2.z)!=0 )
              return o1.z < o2.z;
      else if( (o1.y-o2.y)!=0 )
              return o1.y < o2.y;
      else
              return o1.x < o2.x;
  }
};
```

Optimisations

- Boundary voxels are computed parallely by keeping the neighbours in shared memory.
- Also, for storing boundary voxels, constant memory have been used as boundary voxels are just been read several times in the code.
- Force calculated by each voxel on every other voxel is calculated in parallel by using dynamic parallelism.

Calculating High Divergence



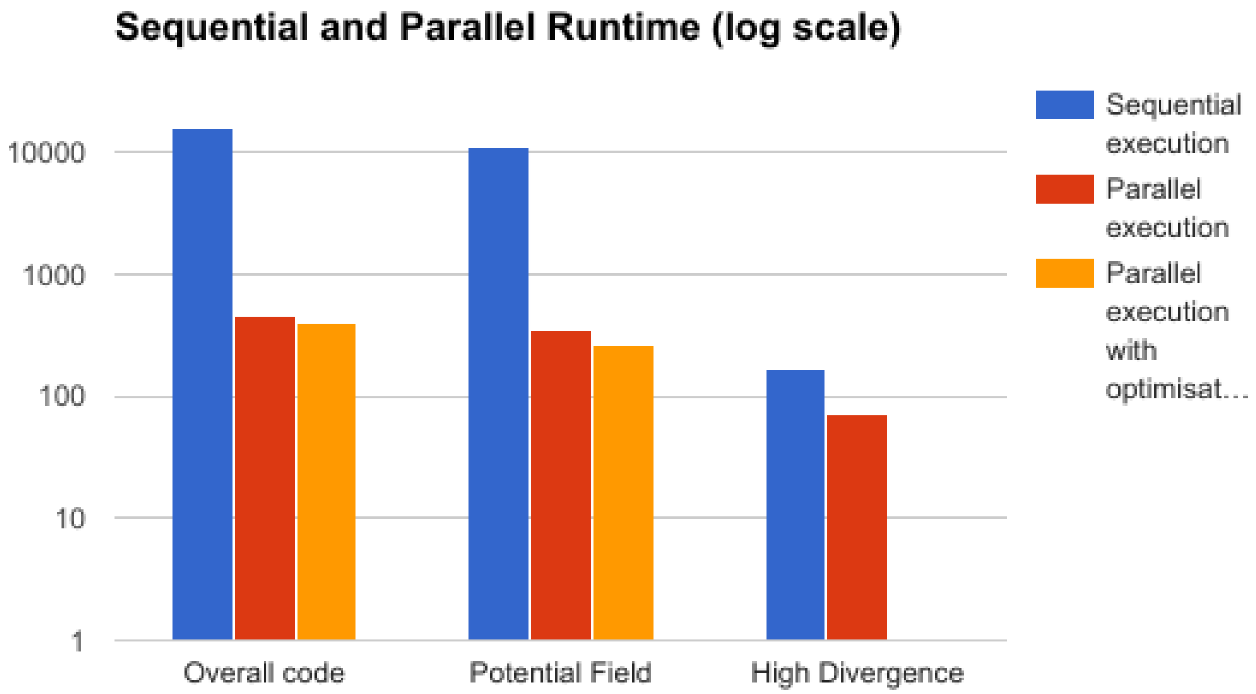
Task Dependency Graph

- Initially, maximum and minimum values are calculated.
- Using these values, threshold is calculated.
- Points for which divergence is less than the threshold value are considered as divergence points.

Runtime of Serial vs Parallel code

	Serial Runtime (in ms)	**Parallel Runtime (in ms)**	**Parallel runtime with optimisations (in ms)**
Total execution time	15727	469	404
Calculating Potential Field	11332	354	265
Calculating High Divergence points	170	72	
Speedup		33.53	38.98

Results (log scale)



THANK YOU