Computing Hierarchical Curve Skeleton Of 3D Objects On Gpu

Alind Khare(2014011) Rohan Juneja(2014156)
Indraprastha Institute Of Information Technology Delhi , India

Abstract

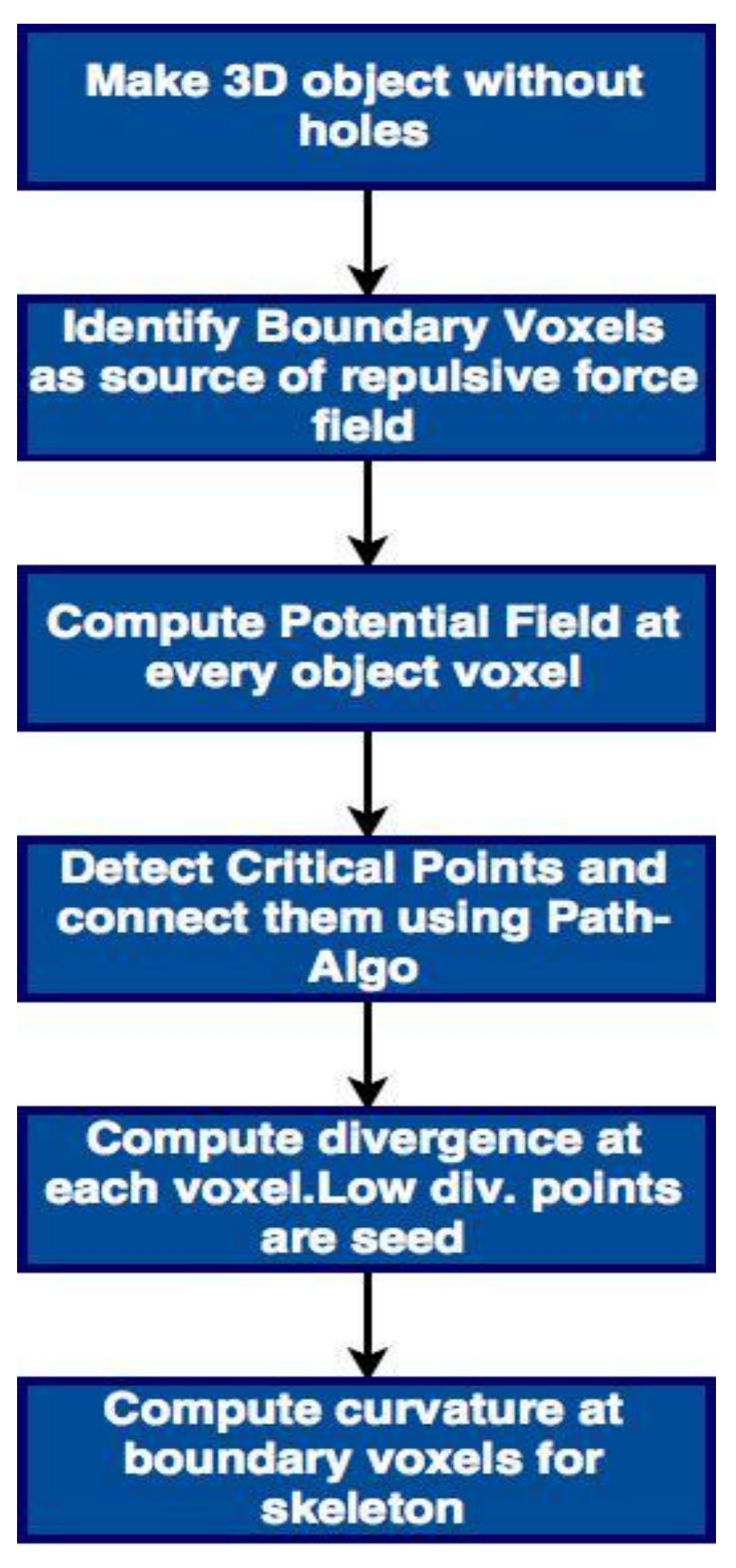
The Algorithm used in this paper is based on computing hierarchical skeletons. This is done by calculating a repulsive force field for every voxel of the discretized 3D object. Based on the topological characters such as critical and divergence points the skeleton is made by connecting such points.
We aim at parallelizing such Algorithm on GPU. Many Computations in the algorithm are embarrassingly parallel and when done on GPU can reduce the execution time.Such Implementation will help in extracting skeletons on huge 3D objects.

Introduction

Skeleton are used widely used for abstracting the topology in multi-dimensions.Skeleton is used to defy a line-like representation of 3D object. This line-like representation is also called curve skeleton. Skeletons have varied applications in real world such as visualization, computer animation, shape matching, curved planar reformation. Here are few properties that a curved skeleton should follow:

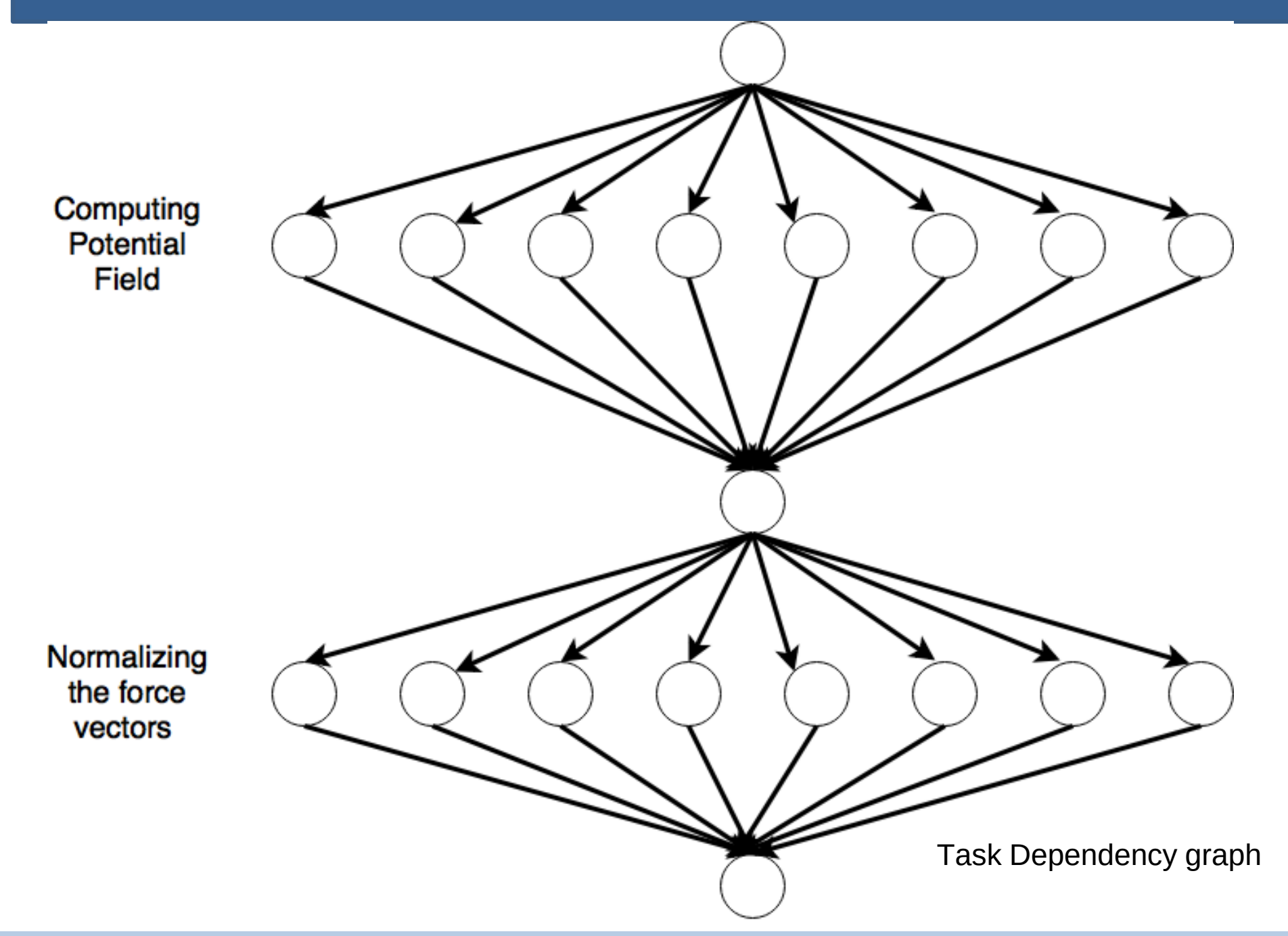
- It should be one voxel thick.
- It should capture essential shape of the object.
- It should be centred around the object.
- It should be connected.
- Different segments of the skeleton should be distinguishable.
- Every interior boundary point should be visible from the skeleton.

Algorithm In A Nutshell

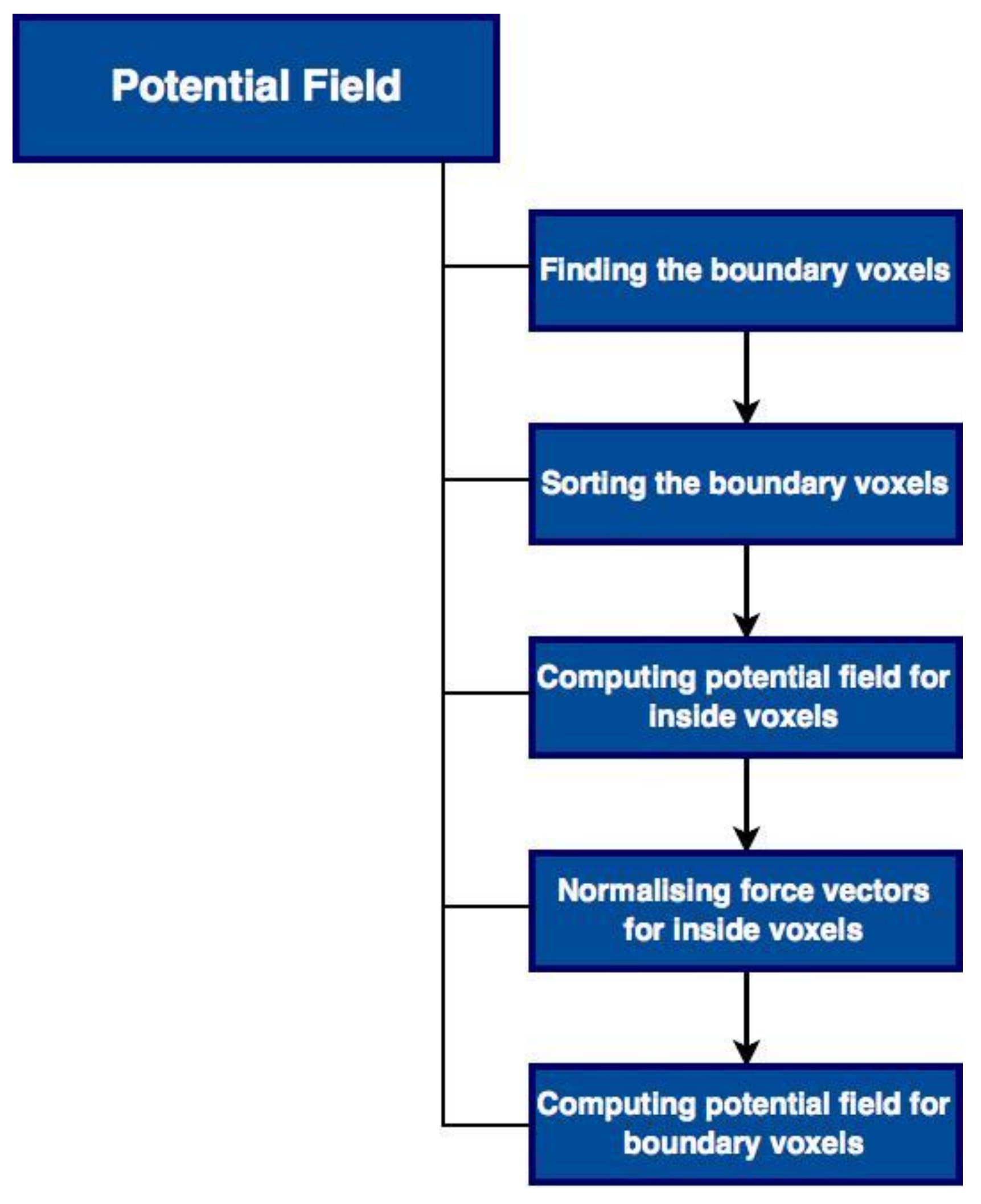

Make 3D object without holes
Identify Boundary Voxels as source of repulsive force field
Compute Potential Field at every object voxel
Detect Critical Points and connect them using Path-Algo
Compute divergence at each voxel.Low div. points are seed
Compute curvature at boundary voxels for skeleton

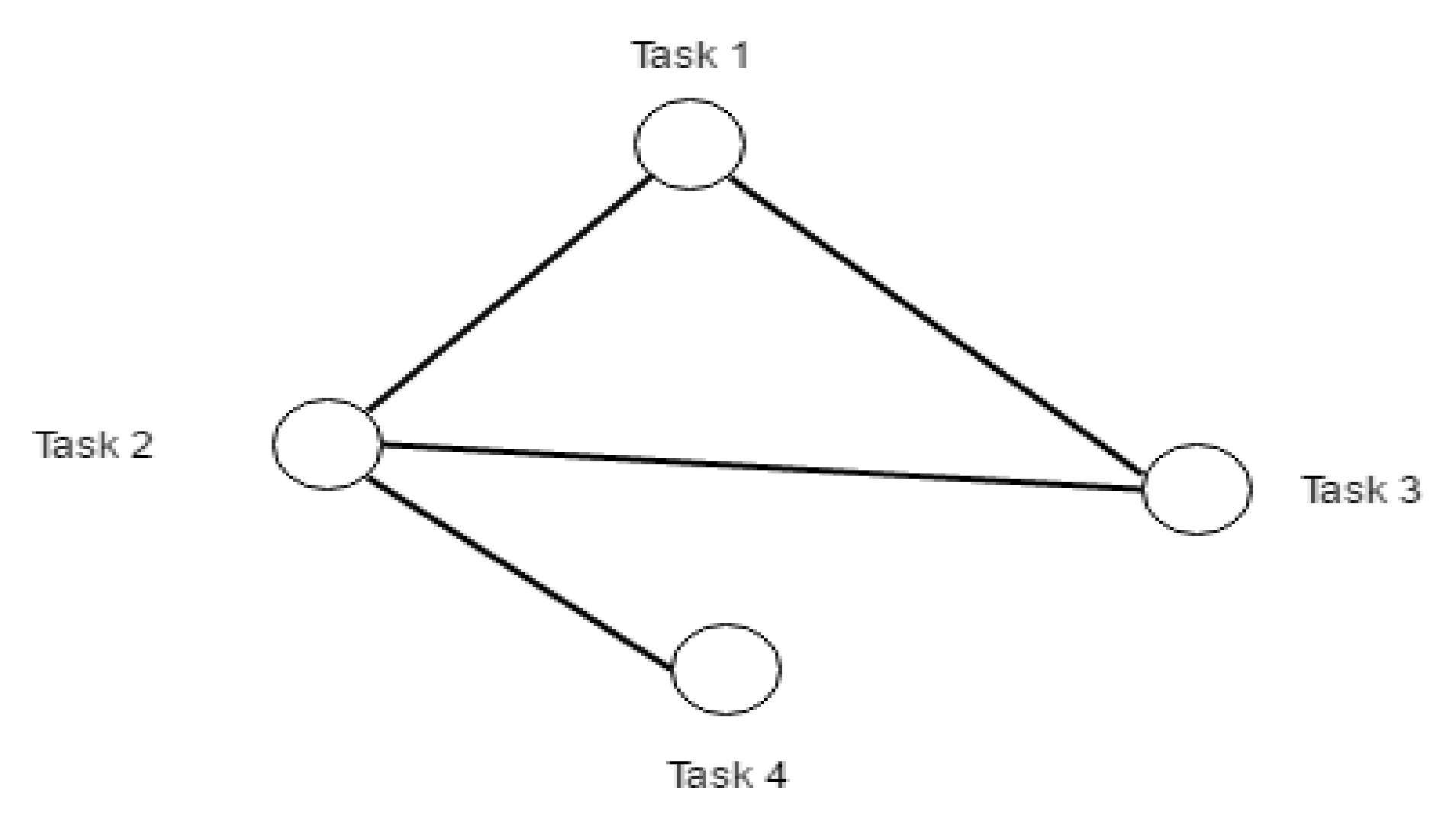
The algorithm starts with filling the holes in the object and converting the input to binary. Repulsive force is calculated by taking boundary point as charges and applying the force function.Based on heuristics critical points are calculated which act as seed to path following algorithm.Connecting Critical points generates Level-0 skeleton. Connecting Low divergence points generates Level-1 Skeleton.

Potential Field

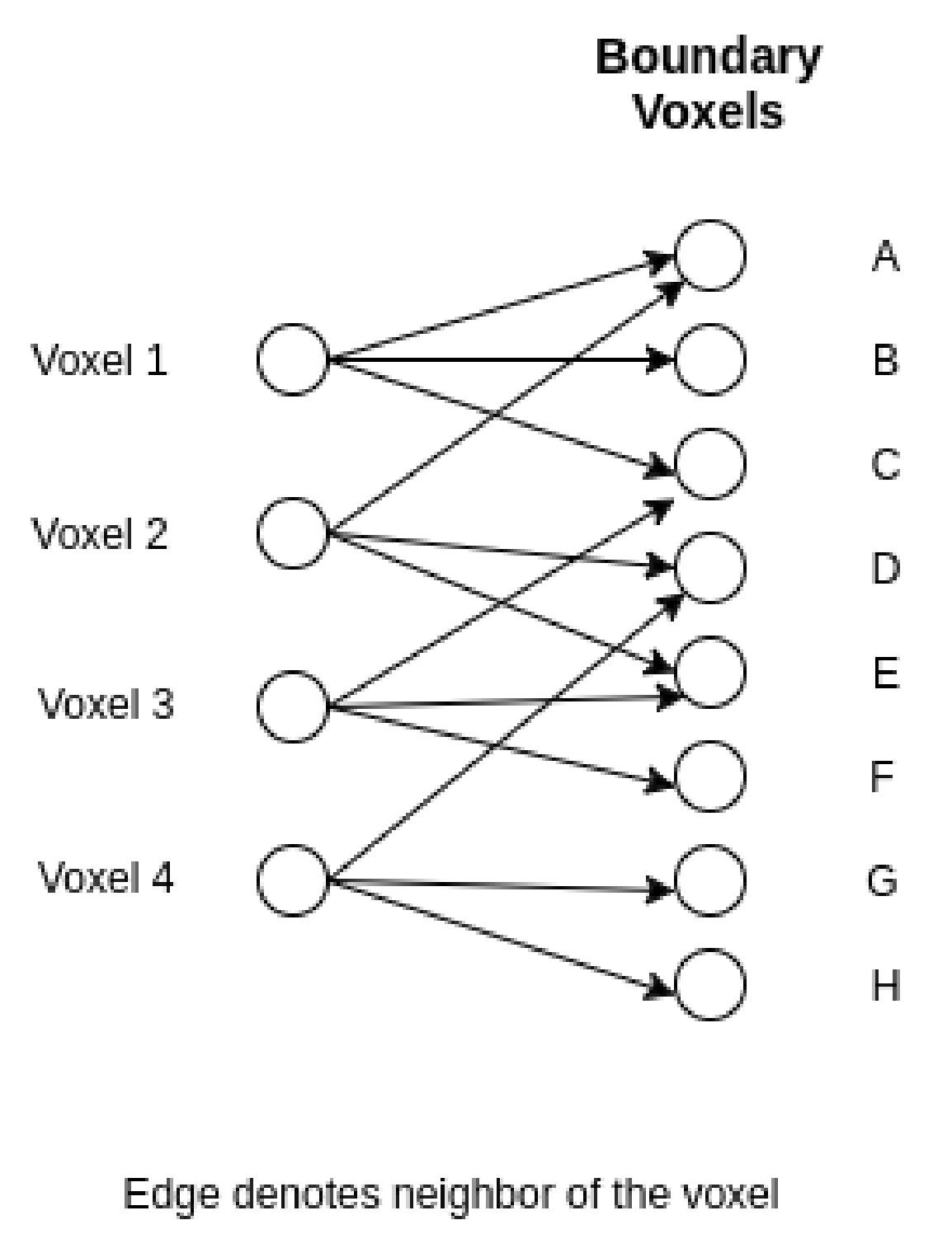


Potential Field Phases


Potential Field
Finding the boundary voxels
Sorting the boundary voxels
Computing potential field for inside voxels
Normalising force vectors for inside voxels
Computing potential field for boundary voxels

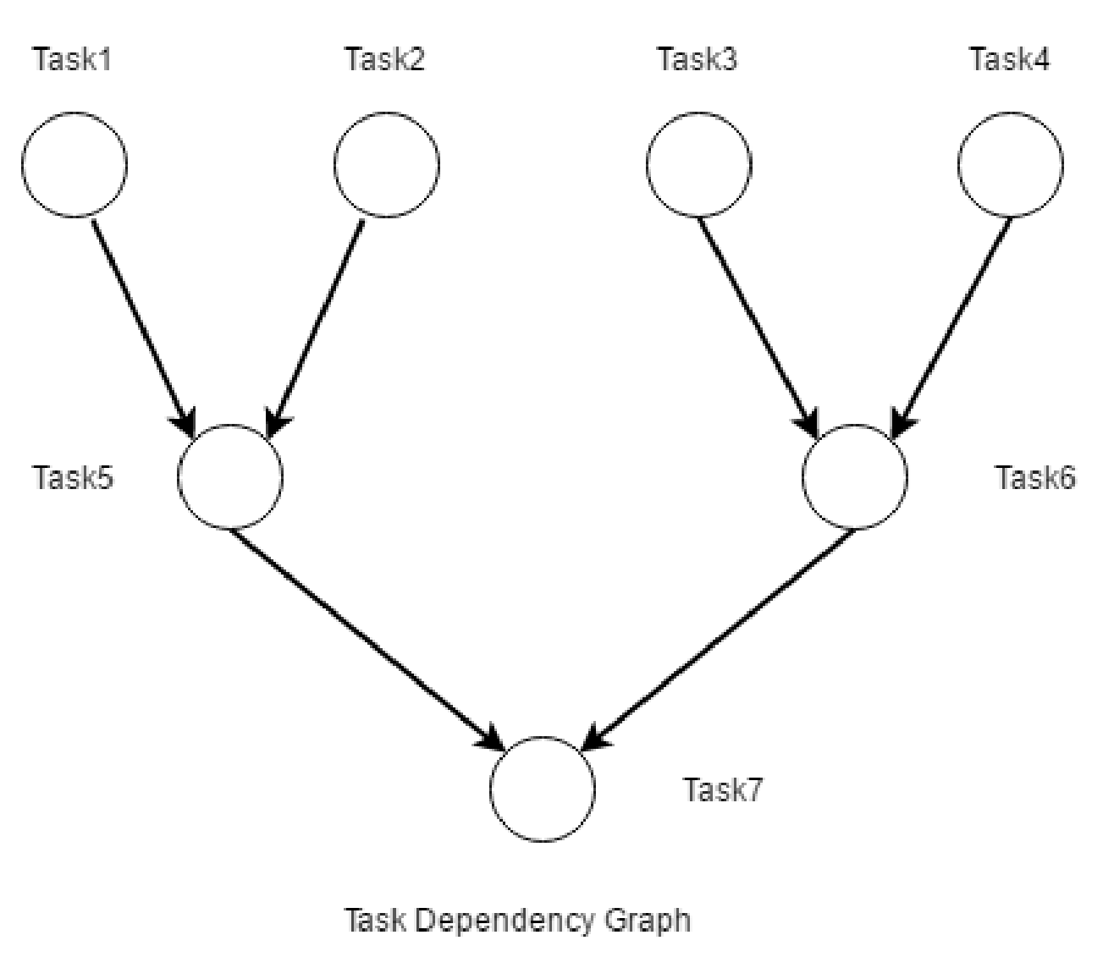


Task interaction graph



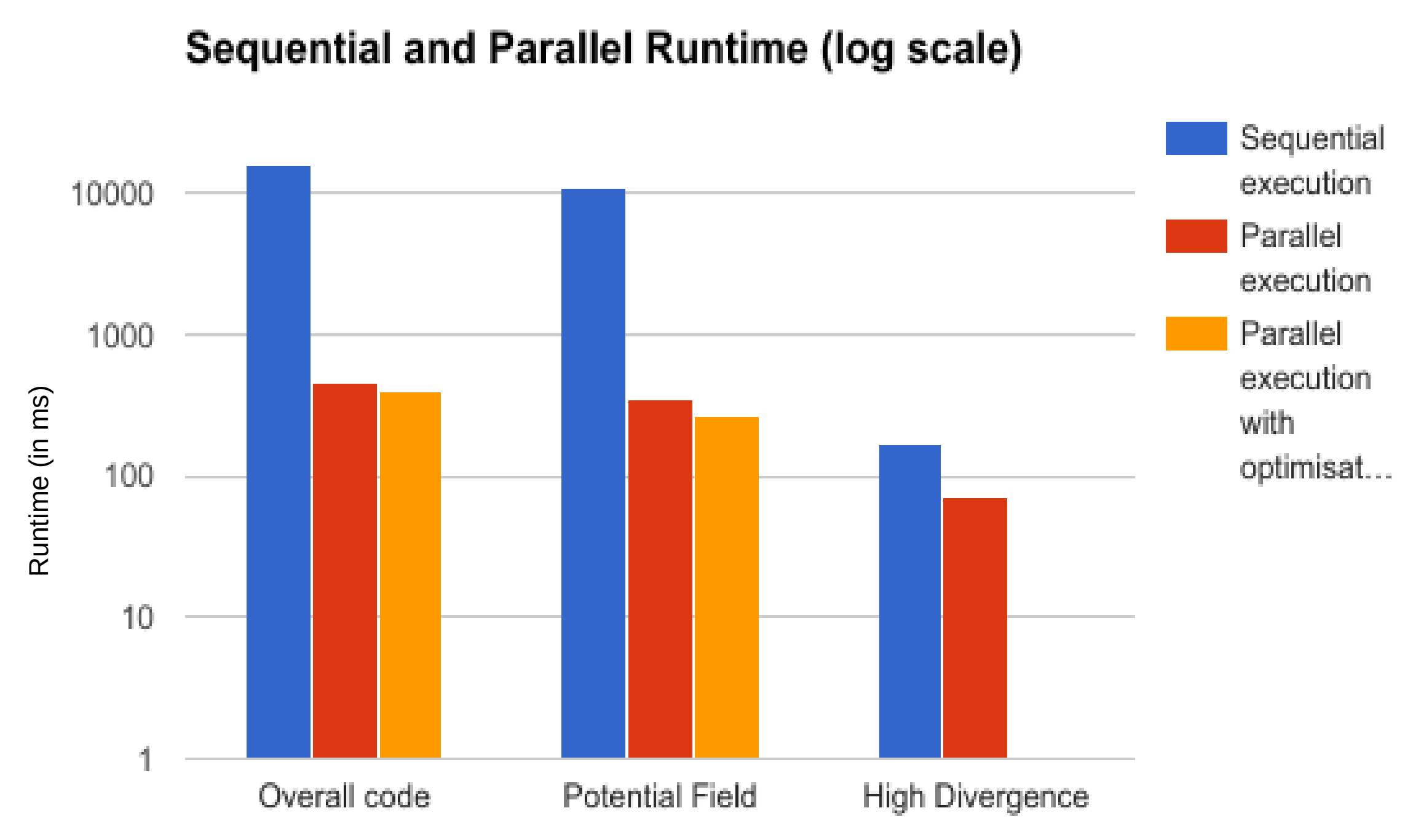
- Computing potential and normalisation of each vector is done independently by different threads.
- As per the figure, since voxel 1 and voxel 2 have the same neighbours, they interact while finding the potential field.

High Divergence Points



- 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.

Results

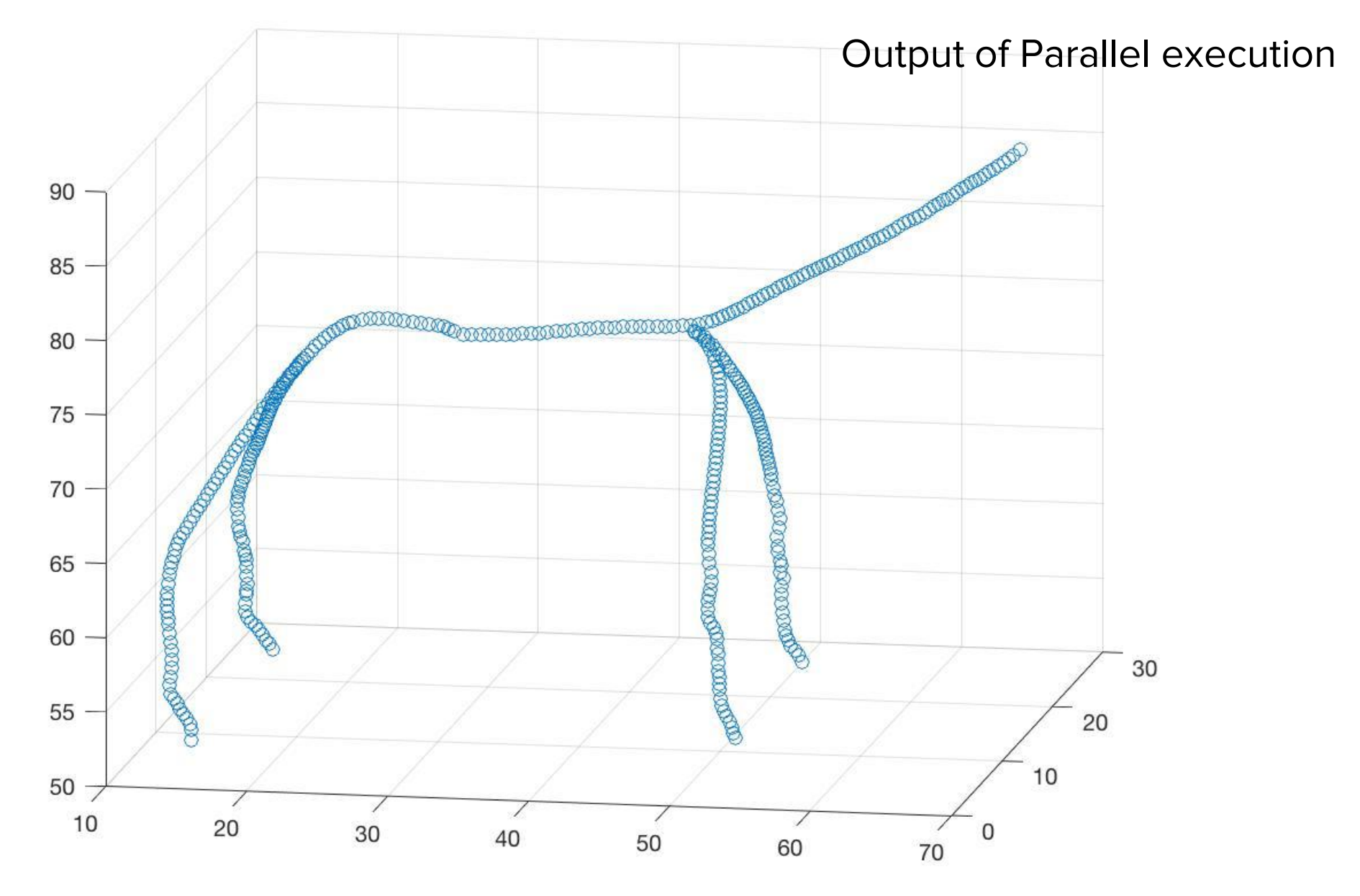


- The overall execution time has been reduced from 15.72 sec to 404 ms, thus giving a speedup of around **38.98.**
- Potential field takes almost most of the execution time, which has been reduced from 11.33 sec to 265 ms, hence giving a speedup of around 42.76.
- Runtime for calculating High Divergence points has been reduced from 170 ms to 72 ms, giving a speedup of around 2.36.

Optimisations

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

Output



Conclusions

- CUDA can be used to bring the execution time less by a huge factor.
- GPU is particularly apt to use for parallelizing the for loops in which each iteration is independent of the other.
- Using shared memory decreases the global memory access which helps reducing the time further.
- Creating a buffer and using it repeatedly for future purposes may help in bringing the execution time of the algorithm less.
- Analyzing task interaction graphs helps optimize the code a lot.

References

1. Sequential code from Rutgers University: http://coewww.rutgers.edu/www2/vizlab/NicuCornea/Skeletanization/skeletanization.html
2. Nicu D. Cornea, Deborah Silver, Xiaosong Yuan, Raman Balasubramanian., 2005, “Computing hierarchical curve skeletons of 3D objects”, This Visual Computer, v. 21, p. 945-955.
3. Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, Tong-Yee Lee.,2008, “Skeleton extraction by mesh contraction”, ACM SIGGRAPH article No. 44.