Computing skeleton of a 3D model

ROHAN JUNEJA, 2014011
ALIND KHARE, 2014011

1 INTRODUCTION

Converting 3D objects into a skeleton is a very fundamental problem with its in application in computer graphics. Various approaches have been proposed to do this, the two approaches discussed being the most popular. One of which is a Hierarchical approach based algorithm which converts the 3D voxelized objects into a repulsive force field (vector field). It then uses topological characteristics of the force field, mainly critical points, to extract the skeleton.

And other which works on a surface mesh, which contracts a mesh to a zero-volume skeleton and then centralise the skeleton.

2 DETAILED DESCRIPTION

An algorithm converts the 3D voxelized objects into a repulsive force field (vector field) .It then uses topological characteristics of the force field, mainly critical points, to extract the skeleton.

The force field over the entire 3D object is computed. This results in a 3D array where each voxel contains a vector value. Critical points, low divergence points and high curvature points are then used to extract skeleton hierarchy.

The vector field is calculated by generating a force field inside the object by the charging object's boundary. The boundary elements are taken to be point charges since these elements are also voxels. Hence, boundary voxel is an object voxel which has empty neighbour. Interior voxel is an object voxel whose neighbours are all object voxels.

Every boundary voxel is taken as a point charge and the force at each interior voxel is computed by summing the influences of all the point charges.

Next when the resulting field is calculated, the critical points are identified running various heuristics. After extracting the critical points, they are classified into various types. The classifications are as follows: attracting nodes (where all vectors point to critical point), repelling node (where all nodes point away from critical point) and saddle points (where some vectors point to and some point away from critical point).

After the critical points are detected. They are connected by path-lines by integrating over the vector field (using some path following algorithm). This computes the core skeleton (level-0 skeleton).Here seed points are saddle points and Path following algorithm is started from seed points.

Next divergence at each voxel is calculated from the vector field. Low divergent points are selected as seed points. Changing the divergence threshold, level-1 skeleton is computed.

Also, the curvature at boundary voxels is calculated and based on the curvature threshold (given by the user) new seed points are selected. Hence computing level-3 skeleton after running the path following algorithm.

But most of the existing methods to extract skeleton, including the above one, require a volumetric representation of the model.

A research group of Hong Kong University gave an idea of extracting skeleton from the surface model directly by contracting the mesh to a zero-volume skeleton. They use a process which includes a contraction term based on the Laplace operator which removes geometry details along the normal directions and an attraction term which retains the geometry information required by using mesh vertices as anchor points. This process does not alter the mesh connectivity. Then a connectivity surgery is done to remove all the collapsed surface and get a 1D skeleton. And to make sure the centeredness of the mesh, each node is moved to the center of its corresponding region.

3 MILESTONES

Milestone 1:
Forming vector field and classifying the the critical points as attracting nodes, repelling node and saddle points

Milestone 2:
Joining the critical points by path following heuristic algorithm with saddle points taken as seed points.

Milestone 3:
Implementing the rest of the hierarchical algorithm and testing the algorithm on a cow model, 3D K model.

4 ANALYSIS OF SERIAL AND PARALLEL CODE

Make solid volume

Initially any holes are covered by applying flood fill algorithm on z-planes followed by y-planes and then x-planes in the order as specified.
Here flood-filling means floodfill the outside of the object in the z-direction, y-direction and x-direction with values.
If one wants to assign values (say 1) to volume in Z direction, the algorithm goes as by initializing a value (1) to one point, then assigning the same value to its neighbours (neighbours are calculated by some criteria). Then one of its neighbours gets picked up and the value to its neighbours gets assigned and so on the algorithm continues assigning the value to points, hence covering any holes if there. This algorithm makes the volume binary.

For parallelising the flood fill algorithm in Z direction, each z plane is provided to one thread. Now in every z plane, it iterates in y and x direction, filling all the holes in that plane. Similar is the case for flood fill algorithm in Y and X direction. Though flood fill algorithm in each direction will run sequentially.

Calculating Potential field

It has following parts:

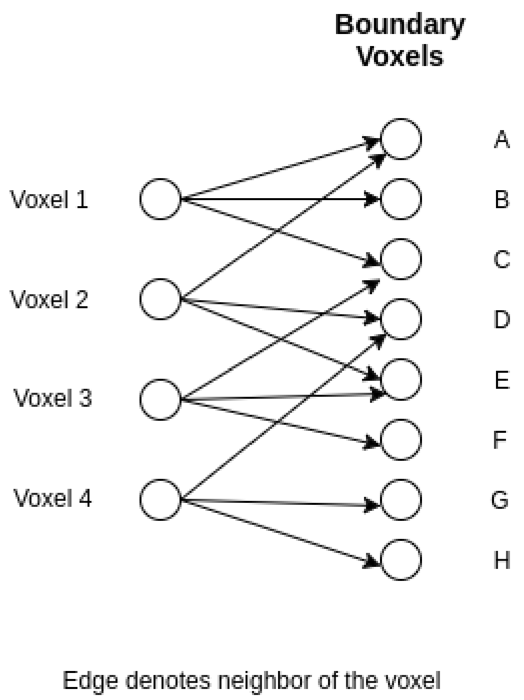
1. Finding the boundary voxels
2. Sorting the boundary voxels
3. Computing potential field for inside voxels
4. Normalizing force vectors for inside voxels
5. Computing potential field for boundary voxels

Initially boundary voxels are calculated by iterating over all the six neighbours of each voxel, if any of the neighbour value is zero then this voxel is added to the boundary voxels array. This array is then sorted based on z-axis values, followed by y-axis values and then x-axis values. Now potential field (vector field) at every voxel is calculated. For a voxel, its neighbouring boundary voxels are searched and force value is calculated by taking all neighbouring boundary voxel as point charges. Neighbours are then calculated by taking boundary voxels into consideration which are 100 units (threshold) in x, y and z direction. These boundary voxels act as point charges and hence force is calculated at each voxel. Now force at each voxel is normalized.

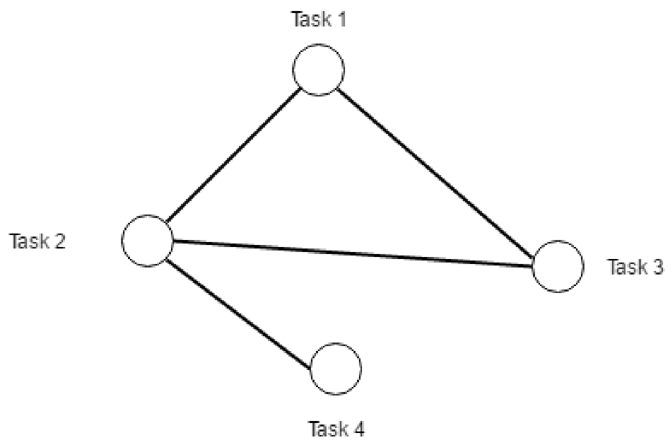
For parallelizing the sorting of boundary voxels, thrust api of cuda can be used. For this we need to come up with our own comparator function for the sorting which matched the sorting given in the code. The comparator functions first compares the values of the z-axis, followed by values in y -axis and finally x-axis values in lexicographic order.

Then calculating potential for inside voxel can be parallelised by taking each voxel independently.
The idea is simple to compute the potential field for each voxel independently. Hence the grid for the kernel was decided as follows: The blocks in a grid were selected as 2-D with x dimension being number of planes in z-axis(for 3-D image) and y dimension being number of planes in y-axis(for 3-D image).The number of threads per block were 1-D and equal to the number of planes in x-axis(for 3-D image).As for each voxel the potential field is calculated independently hence there is no data race and hence parallelisation can be done.

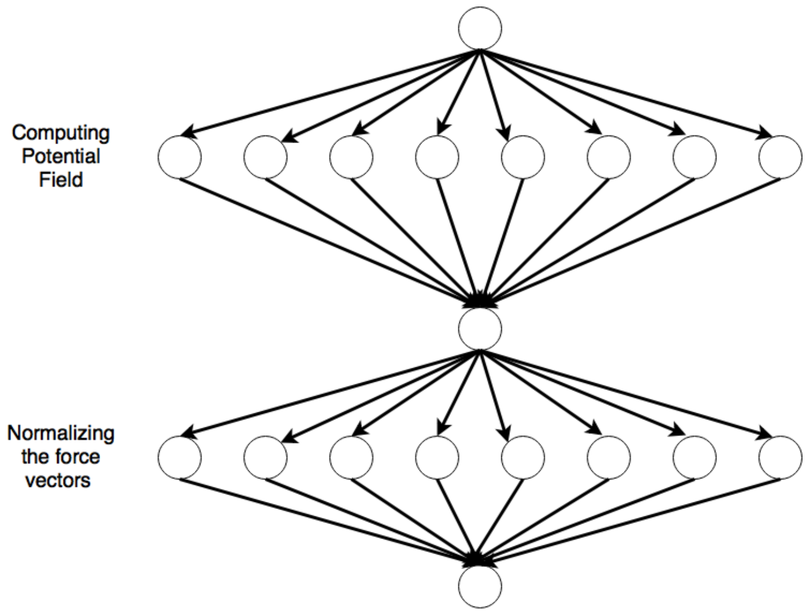
Since normalizing is done for each voxel independently. The computation of each inside voxel can be done independently. Hence, we came up for the kernel which does the same. The grid was same as that of the grid for computing potential field for inside voxels.



The above figure shows the boundary voxels of inside voxels to which they are connected to. Since voxel1 and voxel2 have same neighbors, therefore they interact while finding the potential field as shown below by the task interaction graph.



Task interaction graph

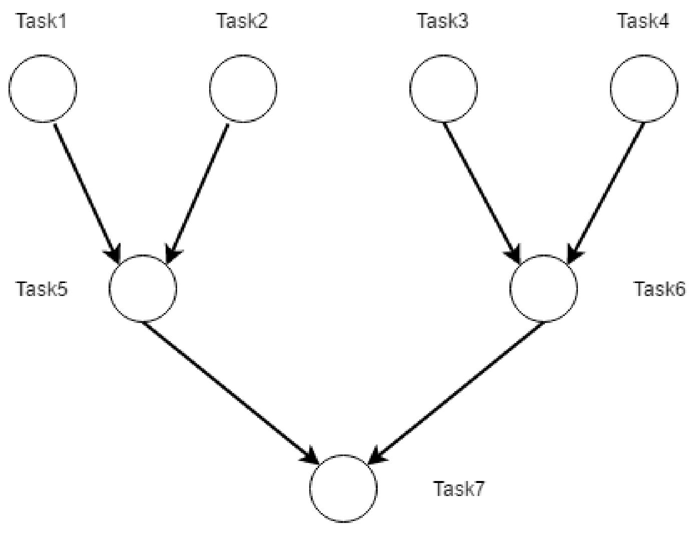


Task Dependency graph

Finding High Divergence Points of a vector field

In High Divergence points, it is initially calculating maximum and minimum value of divergence by iterating over the force values of each voxel. Using the maximum and minimum values, it calculates a threshold value, and if the divergence of the force value of any voxel is less than the threshold, then it is considered as a divergence point.

For finding the minimum and maximum values, max function of thrust library can be used.



Task Dependency Graph

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).

Making streamlines from critical points and seed points

A Jacobian Matrix of vector-field is calculated at every voxel. From this jacobian matrix eigenvalues and eigenvectors are calculated. Critical Points are classified as : attracting nodes, repelling nodes and saddle nodes. Saddle nodes are the points where the there are some positive and some negative real eigenvalues and they serve as seed points. A force following algorithm follows the streamlines which is it follows the direction of eigenvectors having positive eigenvalue. This happens until another critical point is reached or already visited point is reached. This leads to creation of a segment. And by joining segments a core skeleton gets formed.

5 IMPLEMENTATION DETAILS

- For the Potential field, all the phases except finding boundary voxel has been parallelized.
- Parallelization is done by assigning independent CUDA thread to each voxel and computing the potential field value around it.

- For divergence values calculation, a common buffer (C++ struct) is used to store divergence values for every voxel. Each voxel has been assigned an independent CUDA thread to calculate the divergence value.
- Same buffer is used again to identify low divergence points which later serve as seed points to generate Level-1 skeleton.

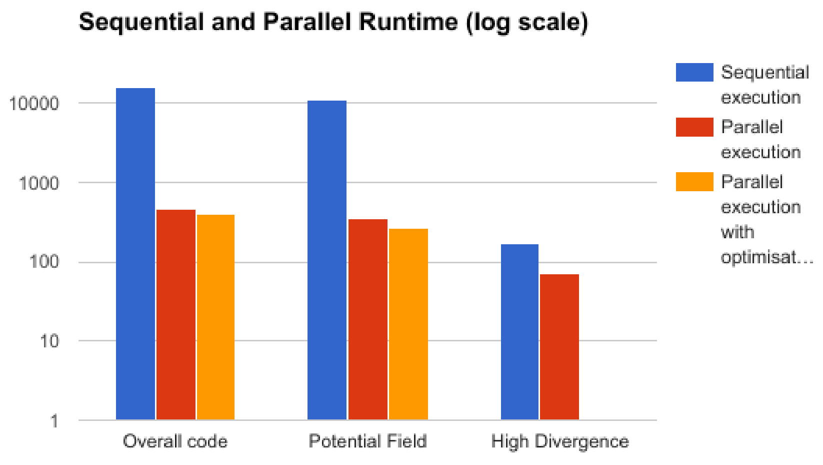
6 OPTIMISATIONS

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

7 RESULTS

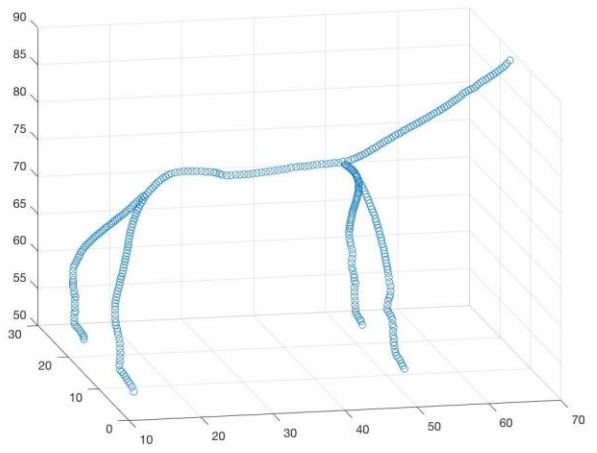
Total speedup achieved

CPU Execution time: 15727 ms
GPU Execution time: 404 ms
Speedup ~ 38.93

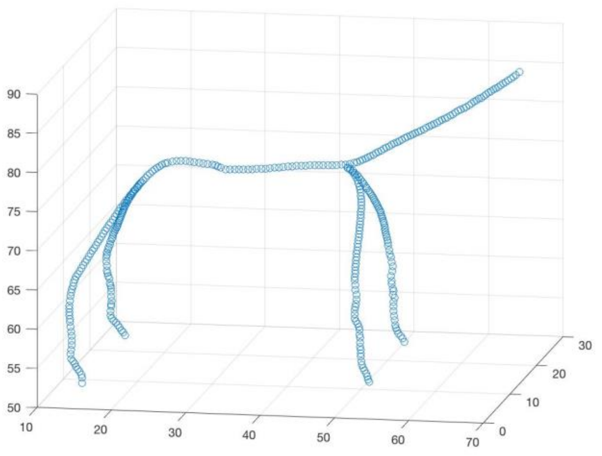


Sequential and parallel execution time of parallelised code (in log scale)

8 OUTPUT



Output for Serial execution



Output for Parallel execution

8 REFERENCES

[1] Serial code by 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.