# Visualizing the Inner Structure of N-Body Data using Skeletonization

**Edward Dale**
Department of Computer Science
Rochester Institute of Technology
Rochester, NY, USA
scompt@scompt.com

**Hans-Peter Bischof**
Department of Computer Science
Rochester Institute of Technology
Rochester, NY, USA
hpb@cs.rit.edu

**Abstract** ***N-body simulations solve the n-body problem numerically and determine the trajectories of the n point masses. The result of these calculations is a huge amount of data detailing the positions and other properties of each body such as mass and velocity. To effectively draw conclusions from these data, one must employ scientific visualization to create images and movies that illustrate the structure of the data. We show that the computer animation technique of skeletonization can be applied to the volume data produced by n-body simulations in order to visualize the inner structure of the data. This novel application is compared to traditional rendering methods in terms of its ability to show this structure.***

*Keywords:* N-body, Skeletonization, Spiegel, Volume , Visualization

## 1 Introduction

The increasing speed of computers means that scientists are able to run situations that were previously impossible, producing torrents of data that need to be somehow analyzed to draw a conclusion. How to distill this large amount of data down to a form that can be digested is potentially a problem just as difficult as the initial simulation, but visualization is a useful tool.

N-body simulations result in a dataset on the order of tens of gigabytes. Examining these data without a visual representation is very difficult. To extract some kind of meaning, we borrow the concept of skeletonization from the field of computer animation and apply it to scientific visualization. Skeletonization (also known as line thinning) is a technique that, when applied to a dataset, yields a "thinner remnant that largely preserves the extent and connectivity of the original region" [1]. It has the immediate benefit of reducing the amount of data being processed. We show it can also be used as an effective visualization for n-body data. This technique is compared to more traditional point- and volume-based rendering.

The rest of this paper is organized as follows. In Section 2, a background summary of the concepts and subject matter being used is presented. Section 3 explains the data structures used and algorithm implemented. Results, including pictures, are presented in Section 4. Finally, some conclusions are made about the viability of skeletonization as a visualization technique and what future work is possible.

## 2 Background

### 2.1 N-Body Simulations

The n-body problem is the problem of finding the "trajectories in time of $N$ point masses whose only interaction is Newtonian gravitation" [2]. This is a very computationally expensive problem ($\mathcal{O}(n^2)$) as each body exerts a force on every other body according to Newton's law of gravity. A closed solution only exists for the $n = 2$ case. For all $n > 2$, numerical approximations must be used. Typically, an n-body simulation is used to model another process. We have visualized simulations of models of the evolution of galaxies consisting of hundreds of thousands of stars, molecular clouds, gas, and dark matter over time. The results of these simulations and the input to the algorithms described below is a sequence of files containing the properties of each body at each timestep.

## 2.2 Skeletonization

### 2.2.1 Introduction

Skeletons are a useful and intuitive structure to work with because they capture the essential topology of an object. The extraction and manipulation of skeletons from volume data is well-researched and there are a number of techniques, however, we will be focussing on the ones presented in [3].

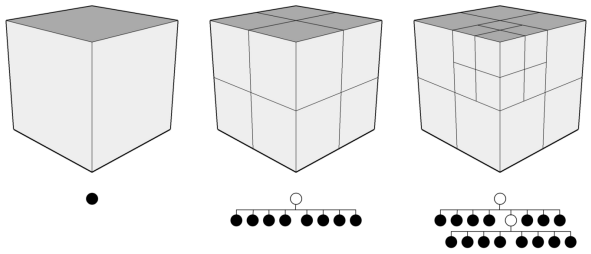
Gagvani et al. define two types of skeleton structures: skeletal voxels and the skeleton tree. Skeletal voxels are the set of voxels that form the medial surface (centered with relation to the boundaries) of the volume. The skeleton tree is a connected tree structure consisting of the skeletal voxels and connecting line segments. Figure 1 shows an object and it's skeletal voxel and skeleton tree representations. Notice that the main components of it's structure are preserved: the four corners and lengthened body.

Figure 1: A hexahedron and it's associated skeletal voxels and skeleton tree.

### 2.2.2 Algorithm

Extracting the skeletal voxels from a volume is done by thinning that volume until a desired thinness is reached. This thinness is the thinning parameter *tp*. The method described in [3] uses a weighted distance transform that eliminates the need for costly floating point operations. The algorithm proceeds by propagating boundary distances inward until there are no new voxels. The distance transform of a voxel $q$ needs to be within *tp* of the mean of all 26 neighbors of $q$ in order to be a skeletal voxel. For low values of *tp*, the distance transform of a voxel need be only a little higher than it's neighbor to be a skeletal voxel, yielding a large set of skeletal voxels. A higher value creates a much smaller set [1].

To create the skeleton tree, a graph theory approach is taken. A completely connected graph is created with the skeletal voxels as the vertices. The edge weights are a linear combination of spatial distance between two voxels and the difference in their distance transform values as shown in Equation 1. The $\alpha$ adds yet another parameter to the process.



(a) Level 0 (b) Level 1 (c) Level 2

Figure 2: Octree decompositions of a cube.

The minimum spanning tree of this graph is the skeleton tree of the original volume.

$$EW_{V1\leftrightarrow V2} = \alpha * DIST_{V1\leftrightarrow V2} + (1-\alpha) * \|(DT_{V1} - DT_{V2})\| \quad (1)$$
$$, 0 \leq \alpha \leq 1.$$

## 2.3 Spiegel Visualization Framework

Spiegel is a Java visualization framework for large- and small-scale systems that uses a pipeline approach similar to that of Unix to create a visualization. Functional blocks are assembled in the pipeline using a graphical programming environment. A visualization problem is solved by implementing new and reusing existing blocks to solve individual tasks [4]. The data structures and algorithms described below were implemented in Spiegel by using and extending the constructs it provides.

# 3 Implementation

## 3.1 Data Structure

When dealing with a three-dimensional space containing particles numbering in the hundreds of thousands, the data structure used needs to be sufficiently advanced to mask the inherent complexity. The ideal data structure divides the input space into manageable, regular subspaces. Such a subdivision of a space can be achieved with a region octree such as the one displayed in Figure 2.

A space can be decomposed into infinitely many cubes, so a criterion for decomposition must be established. Different criteria yield octrees with different behaviors, which may be better suited to a particular use. The octree implemented for the skeletonization algorithm below is the MX octree.

The MX (MatriX) octree [5] divides a space into regions of constant size. When inserting a point into an MX octree, the space will be recursively decomposed until the desired size is met. In a tree representation of an MX octree, the tree will have a set height and every point in the space will exist in the lowest level.

The initial step of the algorithm is to insert all of n-bodies into an octree, thereby decomposing the space and voxelizing the data. Some positional error can be introduced at this point, but as discussed in [6], it is negligible.

### 3.2 Skeletonization

The skeletonization algorithm described in Section 2.2 had to be modified slightly because of the characteristics of the volume data being processed. The volume data that the algorithm was designed for is a space containing a continuous block of object voxels, the rest of which are background voxels (*e.g.* Figure 1). However, because of the scatteredness of n-body data and the initial voxelization step, the data will be regular, but there won't necessarily be a single border between object and background voxels.

The determination of whether a voxel is part of the object or not is critical as too many or too few initial voxels will make accurate skeleton extraction impossible. Two more parameters were added to tune this step. The first parameter controls the height, and thereby the resolution, of the input octree to the skeletonization algorithm. The second is the connectedness that a voxel must have to be an object voxel.

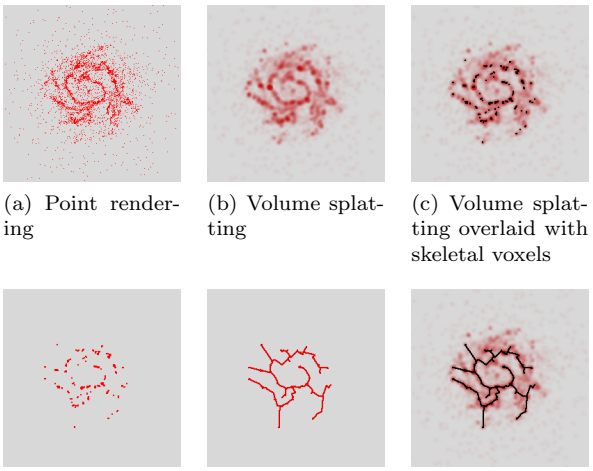
### 3.3 Skeleton Visualization

The output of the skeletonization algorithm is either a set of points (the skeletal voxels) or a set of line segments (the skeleton tree). To visualize either of the representations, they were simply sent to the video card as primitives to be rendered.

## 4 Results

### 4.1 Rendering Method Comparison

Figure 3 shows two different existing rendering methods applied to the gas particles of an isolated spiral galaxy compared with the two skeleton visualizations. Figure 3a renders each gas particle as a point. Figure 3b voxelizes the gas particles and then renders the volume using volume splatting. Figures 3d and 3e show the skeletal voxels and skeleton tree rendered as points and line segments respectively.



(a) Point rendering (b) Volume splatting (c) Volume splatting overlaid with skeletal voxels

(d) Skeletal voxels (e) Skeleton tree (f) Volume splatting overlaid with skeleton tree

Figure 3: Different rendering techniques applied to gas particles.

In the first two figures, a structure can be observed, which should be present in the skeleton representations. By overlaying the skeletons on the volume splatting representation (Figures 3c and 3f), the structure can be visually verified as being present.

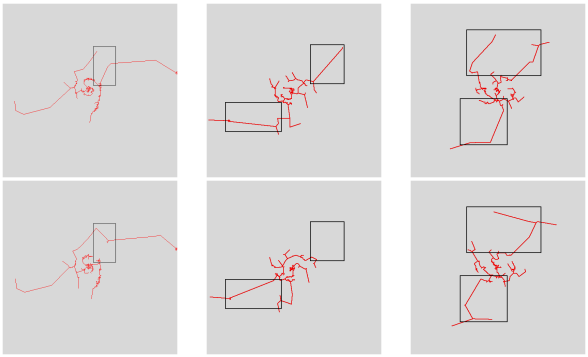
Having shown that the skeleton visualizations capture the structure of the data, the advantages and disadvantages can be discussed. The two skeleton methods have the benefit of completely eliminating noise outside of the main body of the data. However, they both also lose some of the structure of the data, in particular the absolute center and the top of the data. The skeletal voxels do not provide a very useful visualization of the data unless presented with some other structure (*e.g.* skeleton tree or splatted volume).

### 4.2 Skeletonization Thinness Parameter

Sometimes the skeletonization parameters can hide complex features. In Figure 4, the resulting two arms of two colliding galaxies can be revealed or hidden, depending on the value of the thinness parameter chosen. This highlights the highly data-dependent nature of the skeletonization algorithm.

(a) $tp = 2.0$ (b) $tp = 1.0$

Figure 4: The thinness parameter $tp$ can hide complex features.



(a) $t = 605, 610$ (b) $t = 790, 795$ (c) $t = 1070, 1075$

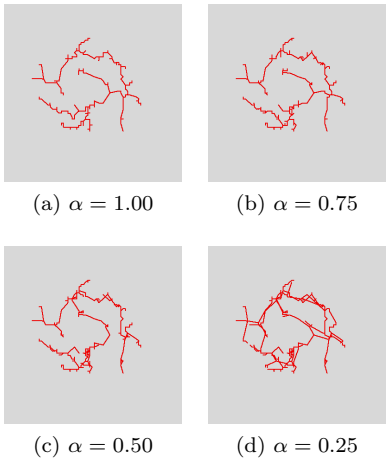
Figure 5: Pairwise consecutive skeleton trees illustrating "jumpiness" between timesteps.

### 4.3 Skeleton Tree Artifacts

The changing positions of the stars leads to very noticeable artifacts in the skeleton tree when examined over time. At any one timestep, the skeleton tree looks like a reasonable approximation of the structure of the galaxy, as seen in Figure 3. Over time, though, the skeleton's structure can change greatly enough that a "jumpiness" effect appears when viewed as an animation. Pairs of skeleton trees at consecutive timesteps can be seen in Figure 5. Note the changes to the line segments in the boxes between timesteps.

### 4.4 Skeleton Tree $\alpha$ Parameter

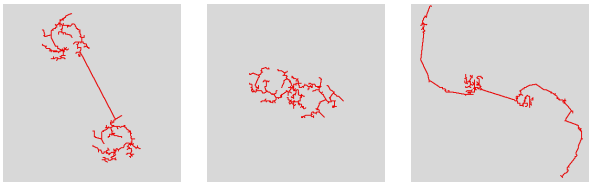
The $\alpha$ parameter determines the contribution of each of euclidean distance and distance transform to the graph from which the skeleton tree is generated. An $\alpha$ value close to 1 means the skeleton is generated based more upon the distance between two particles. This can be seen in Figure 6a. As $\alpha$ decreases, the skeleton becomes based more upon the difference in distance transform between two particles. This has the effect that particles from different sides of the dataset can become connected, a feature that can be seen in Figure 6d and which



(a) $\alpha = 1.00$ (b) $\alpha = 0.75$

(c) $\alpha = 0.50$ (d) $\alpha = 0.25$

Figure 6: Effects of $\alpha$ parameter on skeleton tree.



(a) $t = 0$ (b) $t = 100$ (c) $t = 300$

Figure 7: Skeleton trees of colliding galaxies.

is generally undesirable because particles close to each other are more likely to be related.

### 4.5 Skeleton Trees of Colliding Galaxies

The skeletonization algorithm generates a potentially undesirable artifact when the data is divided into more than one physical group, such as the case of two galaxies before collision. The skeleton tree generated from this data contains a skeleton for each galaxy and an undesirable line connecting them as seen in Figures 7a and 7c. For the time steps where the galaxies are joined, as in Figure 7b, the line is no longer conspicuous and need not be removed.

## 5 Discussion and Future Work

We have successfully shown that skeletonization is a valid technique for displaying the inner structure of n-body data. This success is, however, only limited to skeleton trees; skeletal voxels do not appear to be a useful visualization technique. Once the particles become disperse enough, the skeletal voxel representation is too sparse to derive any meaning

visually. Some structure needs to be placed over these voxels, usually in the form of the skeleton tree. The skeletal voxels could also be used as input to another algorithm.

At discrete points in time, the skeleton tree is able to show a reasonable approximate of the structure of the data, as verified by point and volume rendering. However, significant artifacts emerge when the skeleton tree is animated over time. One solution would be to incorporate the skeleton of the previous timestep into the skeletonization algorithm using some form of feature tracking, as mentioned in [1]. Another possible solution is to create a spline-type skeleton representation that is less sensitive to changes. This runs the risk of losing touch with the actual structure of the data, but has other benefits such as smooth automatic navigation, as also mentioned in [1]. Smooth navigation would be useful to generating an animation that "tours" the interesting structural pieces of the n-body data over time.

The determination of the skeletonization parameters is currently a trial-and-error process. Adding some more intelligence to the algorithm to allow it to determine the correct parameters would make the results more repeatable across different datasets.

In the case of multiple disjoin structures in the data, the algorithm should detect the situation and create multiple skeletons accordingly.

Moreover, the skeleton tree has only been shown to be useful when looking at the position of particles. No attempt was made to skeletonize based on other particle attributes.

# References


[1] N. Gagvani and D. Silver. Parameter controlled skeletonization of three dimensional objects, 1997.

[2] Jean Kovalevsky. *Introduction to celestial mechanics*, volume 7 of *Astrophysics and space science library*. Springer-Verlag, New York, 1967. Translated by Express Translation Service; 25 cm; Bibliography: p. [127]; Translation of Introduction la mecanique celeste.

[3] Nikhil Gagvani, D. Kenchammana-Hosekote, and D. Silver. Volume animation using the skeleton tree. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 47–53, New York, NY, USA, 1998. ACM Press.

[4] Hans-Peter Bischof, Edward Dale, and Tim Peterson. Spiegel - a visualization framework for large and small scale systems. In *MSV '06: Proceedings of the 2006 International Conference of Modeling Simulation and Visualization Methods*, pages 199–205, Las Vegas, USA, 2006. MSV'06/ISBN #:1-60132-010-8/CSREA.

[5] Hanan Samet. *Applications of spatial data structures: Computer graphics, image processing, and GIS*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[6] Edward Dale. Visualizing the inner structure of n-body data using splatting and skeletonization. Master's thesis, Rochester Institute of Technology, June 2006.