Visualizing the Inner Structure of N-Body Data using Splatting and Skeletonization

by

Edward Dale

A Thesis Proposal Submitted in Partial
Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Supervised by

Graduate Coordinator Hans-Peter Bischof
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York
March 2006

Approved By:

Hans-Peter Bischof
Graduate Coordinator, Department of Computer Science
Primary Adviser

Stefan Harfst
Astrophysics Group, Department of Physics

TBD
TBD

Abstract

The primary goal of this thesis is to research the fundamental concepts involved in the rendering of volume data and develop an acceleration technique that emphasizes the inner structure. The volume data that will be visualized is the result of n-body simulations. This new acceleration technique consists of finding the "skeleton" of a volume and rendering the voxels closer to the skeleton with a higher priority. Both the novel and existing visualization techniques will be compared in terms of speed and image quality. All of the techniques will be implemented in Java for the Spiegel visualization framework.

Contents

1. Overview

1.1 N-Body Simulation

N-Body simulations calculate the forces acting between a high number (N) of bodies. This is done by computing the force of body i applied on body j and using integration to calculate positions and velocities. This must be done $(N-1)$ times for each of N particles yielding a $\mathcal{O}(N^2)$ computation time. The end result of these calculations is the position, velocity, and acceleration of each body. [7]

1.2 Skeletonization

Skeleton extraction and manipulation is a very intuitive technique for volume animation. The main benefit being the large body of algorithms and techniques for dealing with articulated skeletons stemming from the research done regarding character animation. Skeletons also capture the essential topology of an object. For the purposes of this thesis, a skeleton consists of the set of voxels that form the medial surface (centered with relation to the boundaries) of the volume. A centerline is curve-like representation of the medial surface. The skeleton and centerline need not be completely connected, however, a later step in the algorithm discussed below ensures this. Both of these representations have a single parameter that determines how thin the resulting object should be [2].

Extracting a skeleton from a volume is done by thinning that volume until a desired thinness is reached. This is the thinning parameter that was mentioned earlier. The method described in [2] 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. For each voxel, the average of all its neighbors is calculated and compared with the thinning parameter

to determine if it is a skeletal voxel.

The previous step yields a collection of voxels that likely do not form a connected skeleton. This is resolved by creating a completely connected graph with the voxels as vertices and a linear combination of spatial distance and distance transform difference as the edge weight. The connected skeleton is then the minimum spanning tree of this graph.

Determining the centerline from disconnected skeletal voxels is done using a recursive midpoint algorithm. It is initiated using the two desired end-points of the centerline and finishes when the distances fall below a *fineness* threshold.

1.3 Volume Rendering

Direct volume rendering methods map voxels directly to pixels. Creating the mapping from sampled value to opacity and color is called classification and is likely the most difficult step in rendering a volume. The rendering algorithm would only need to be written once, but the classification would need to be tailored to the dataset being rendered.

The classification mapping can occur a number of ways, however this thesis will deal only with splatting. Splatting can be thought of as taking each voxel and throwing it, as one would a snowball, at the image plane. Each voxel is projected onto the image plane using a gaussian kernel function scaled by the distance of the voxel from the image. The volume is traversed from front to back yielding a rough approximation that is progressively refined as more slices are processed [6]. This is the "regular" splatting that will be implemented for this thesis.

Another technique that will be implemented and compared uses a hierarchical data structure based on either an octree or principal component analysis clustering. The hierarchical approach allows the visualization resolution to be lowered for interactive manipulation and raised for final rendering. The multiple resolutions can also be combined so that very relevant parts of the image higher resolution.

Improvements were made on the memory requirements by only storing coordinates of points relative to the centroid of the cluster they are in. This allowed Hopf *et al.* to use bytes and shorts instead of floats. Additionally, for time-varying data, spline control points can be stored instead

of coordinates, further reducing memory requirements while maintaining accuracy. These accelerations allowed for real-time manipulation of 16 million particle datasets. These additional optimizations may or may not be implemented here.

2. Architectural Overview

The entire thesis will be implemented in Java for the Spiegel visualization framework [3]. Because Spiegel encourages a separation of visualization tasks into a number of discrete boxes. The division of this project is described in figure 2.1.

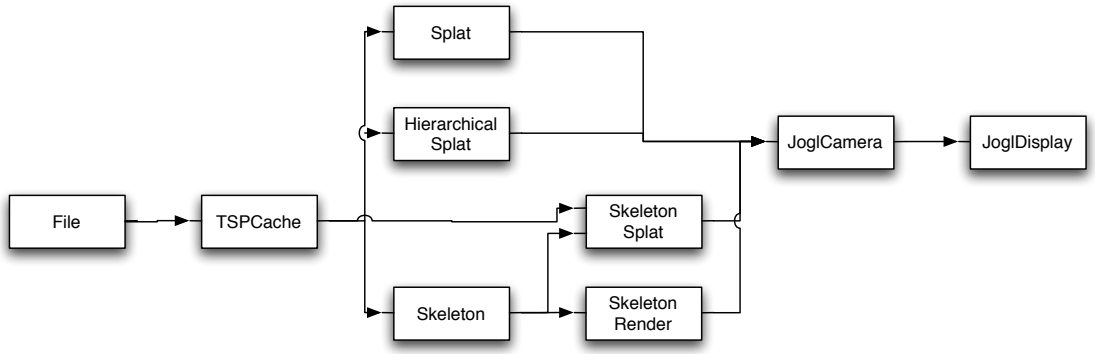

Figure 2.1: Project breakdown

File Reads the input files in the format described in section 3.2. This will be an adapted version of the current File input component that can read a series of files.

TSPCache This will potentially be a caching block that stores the input files in a format that is easier to read in Java. The results of the parsing will be stored in some kind of binary format (possibly serialized objects) the first time that they are read and used for subsequent visualizations. The format could be a type of Time-Space Partitioning (TSP) tree [8].

Splat This component will do a non-optimized splatting of the input data [9].

Hierarchical Splat Renders the input volume using a hierarchical data structure for acceleration [5].

Skeleton Determine the skeleton of the volume data. This will output a `Skeleton` object that will contain the minimum spanning tree of the skeletal voxels [1].

Skeleton Splat This will be the novel algorithm developed in this thesis. It will use the skeleton as the acceleration structure for rendering the volume.

Skeleton Render An auxiliary component to render the skeleton of the volume.

JoglCamera All of the above rendering components will produce an object conforming to a `Drawable` interface that will be rendered by this component. This component will be developed as part of another project being done by the author concurrently with this thesis.

JoglDisplay This is the matching component to the `JoglCamera` that actually displays the rendered image on the screen.

3. Functional Specification

3.1 Usage

The Spiegel components developed as part of this project will be used in the same way that Spiegel is currently used. They will need to be laid out in a manner similar to figure 2.1. One of the three rendering methods (regular, hierarchical, skeleton) will normally be used at a time, but more than one is possible. A file will be selected using the File input component and a time will be selected using the Clock component. The Update button will then need to be pressed to begin the visualization.

3.2 Input Data Format

The n-body data will be the result of calculations from [4] and will come in the form of a number of text files, each representing a snapshot of the system at a certain. Each file has a three line header containing the snapshot number (starting at zero), the number of total particles as well as the number of each type of particle (star, cloud, gas, dark), and the time of snapshot.

Following the header is N lines in the format defined in figures 3.1 and 3.2 and table 3.1.

```
id mass x y z vx vy vz pot h u rho T tob ipt icomp igal icol
```

Figure 3.1: Data line format

```
1    1.7145E-05   -1.6669E+01 -4.6704E+00  2.3354E-01
  7.9363E-02 -2.5869E-01  1.7513E-02   -2.0836E-01
  0.0000E+00  0.0000E+00 -1.0000E+00 -1.0000E+00
  -4.3960E+03     1   1   1     1
```

Figure 3.2: Sample data line

Table 3.1: Data line format

	Description
id	Particle index starting from 1 (Not unique)
mass	Particle mass (Not constant)
x y z	Particle position
vx vy vz	Particle velocity
pot	Potential
h	Particle radius (Undefined for stars and dark matter but gives a true radius for clouds and the smoothing length (kernel size) for gas (SPH) particles)
u	Internal specific energy (Only definded for gas particles (energy per unit mass))
rho	Mass density of the gas at the position of particle (Only defined for gas and cloud particles and possibly stars)
T	Temperature, otherwise same as rho
tob	Time of birth (When a particle was introduced into the simulation, only meaningful for stars and clouds)
ipt	Particle type: 1=stars, 2=clouds, 3=gas(SPH), 4=dark matter
icomp	Galaxy component: 1=disc, 2=bulge, 3=halo
igal	Galaxy identifier if two or more galaxies
icol	The same as ipt

4. Deliverables

All deliverables will be made available online at http://www.cs.rit.edu/∼erd4819 and eventually http://www.cs.rit.edu/∼grapecluster.

- Thesis (technical paper):
  - Estimated size: 20 - 30 pages
  - Includes sample input data and design documentation
  - Includes comparison of existing techniques and novel skeleton technique presented here
- Source Code:
  - Committed to the Spiegel CVS repository
  - Full Javadocs documenting the code
- Usage instructions posted to the Spiegel website
- Images and movies generated from n-body data simulated by Stefan Harfst

5. Detailed Schedule

Table 5.1: Detailed Schedule

Date	Milestone
17 March 2006	Proposal submitted
7 April 2006	Volume rendering with regular splatting
21 April 2006	Volume rendering with hierarchical splatting [5]
19 May 2006	Volume splatting with skeleton acceleration
9 June 2006	Defend

6. Current Status

Research has been done, but no code has been written. Two research papers have been written on volume rendering and animation that will likely form the introduction and background of the final paper.

Bibliography

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

Description: The authors present a technique for thinning a volume to a parameterized point of thinness. The procedure is reversible and uses primarily integer math for performance.

[2] 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.

Description: The authors present a technique for extracting the skeleton from a volume. This is accomplished by thinning the volume to a certain point and then creating the skeleton from the minimum spanning tree of the fully connected graph with the skeletal voxels as the vertices.

[3] The grape cluster project. http://www.cs.rit.edu/∼grapecluster/.

Description: The GRAPEcluster project is the source of the Spiegel visualization framework that will be used to implement the three rendering algorithms.

[4] Stefan Harfst, Christian Theis, and Gerhard Hensler. Modelling galaxies with a 3d multi-phase ism, 2005.

Description: The authors present a new method for modeling the evolution of galaxies. It is simulation data from galaxies simulated with this method that will be visualized in this thesis.

[5] Matthias Hopf and Thomas Ertl. Hierarchical splatting of scattered data. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 57, Washington, DC, USA, 2003. IEEE Computer Society.

Description: The authors present a method of render large uncorrelated point sets. The data is stored hierarchically as quantized relative coordinates. The hierarchical structure allows a trade-off between rendering speed and image quality.

[6] Michael Meißner, Jian Huang, Dirk Bartz, Klaus Mueller, and Roger Crawfis. A practical evaluation of popular volume rendering algorithms. In *VVS '00: Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 81–90, New York, NY, USA, 2000. ACM Press.

Description: A summary and comparison of a number of different volume rendering algorithms including splatting.

[7] N-body/particle simulation methods. http://www.amara.com/papers/nbody.html.

Description: A summary of n-body methods.

[8] Han-Wei Shen, Ling-Jen Chiang, and Kwan-Liu Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 371–377, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

Description: The original proposal of the Time-Space Partitioning tree, which can effectively capture both the spatial and the temporal coherence from a time-varying field.

[9] Lee Westover. Footprint evaluation for volume rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 367–376, New York, NY, USA, 1990. ACM Press.

Description: The original volume splatting paper. Volumes are represented as overlapping basis functions scaled by the voxel value.