Visual Debugging of Visualization Software: A Case Study for Particle Systems

Patricia Crossno
Sandia National Laboratories

Edward Angel
University of New Mexico

ABSTRACT

Visualization systems are complex dynamic software systems. Debugging such systems is difficult using conventional debuggers because the programmer must try to imagine the three-dimensional geometry based on a list of positions and attributes. In addition, the programmer must be able to mentally animate changes in those positions and attributes to grasp dynamic behaviors within the algorithm. In this paper we shall show that representing geometry, attributes, and relationships graphically permits visual pattern recognition skills to be applied to the debugging problem. The particular application is a particle system used for isosurface extraction from volumetric data. Coloring particles based on individual attributes is especially helpful when these colorings are viewed as animations over successive iterations in the program. Although we describe a particular application, the types of tools that we discuss can be applied to a variety of problems.

CR Categories: D.2.5 [Software]: Software Engineering – Testing and Debugging.
Keywords: Visual debugging, algorithm animation, program animation, program visualization, particle systems.

1 INTRODUCTION

Starting with William Reeves' work in 1983 [8], particle systems have been used within the graphics community in a variety of applications. Particle systems have been used to model "fuzzy objects" such as clouds, fire and water [8][4][11]. Hierarchical particle systems have been used to model trees and grass [12][7]. The combination of particle systems with behavioral rules, known as *behavioral animation*, has been used to model the flocking behavior of birds, herds, and schools of fish [9]. Particle systems have also been used for implicit surface modeling [14][16] and as the basis for meshing algorithms [1][10][17]. We have used particle systems to extract isosurfaces from volume data [3].

As particle systems have been applied to more sophisticated problems, the software has become increasingly more complex. Not only has the size of the systems increased but the particles can be programmed to have complex behaviors and particle systems can include multiple types of particles. Software implementations include a variety of data structures to store and manipulate the particles. The evolution of other visual systems both for specific applications and in general-purpose systems, such as VTK and AVS, has shown similar properties.

During the course of our work, we quickly decided that conventional debugging tools were inadequate for dealing with something as complex and dynamic as a particle system. Our solution was to build a number of graphical tools so that we could debug our code visually. The human eye can instantly recognize patterns or mistakes that hours of examining particle information would not reveal. These tools were invaluable in situations where the sheer number of particles would have precluded the examination of each of the particles individually.

The literature provides examples of earlier work in visual debugging. There are papers on general algorithm visualization [6][2][13]. More closely related to our work are papers on visual debugging of geometric algorithms. Tal and Dobkin present a system, GASP, for visualizing geometric algorithms from computational geometry [15].

Wolfenbarger, et al. describe a framework called CoMeT that provides a graphical debugging environment for their meshing algorithms [17]. Using this tool they can watch the evolution of the mesh with respect to particle positions and links between particles.

Huang, et al. developed a stochastic energy minimization approach for untangling knots [5]. In their paper, they used two different coloration schemes to represent energy at various positions within the knot's curve. Although, Huang et al. never explicitly claimed to use these colorings as visual debugging tools, the information about the energy distributions would have been useful for interpreting the behavior of the algorithm over time.

This case study describes the tools that we developed for visually debugging a particle system that does isosurface extraction. We used Advanced Visual Systems scientific visualization system, AVS 5.0, as our software platform. We wrote a particle system module and used AVS's visual programming paradigm to input an AVS field data type from their read field module and to output AVS's geometry data type to their geometry module. We used the various graphical user interface widgets that AVS provides to manipulate variables such as surface value and curvature factor within the particle system module. The geometry viewer not only provided controls for rendering the resulting surface models in a variety of useful ways, but also provided a user interface for flying around the models. Nevertheless, debugging the system would have been difficult with just these tools alone.

2 OVERVIEW OF THE ALGORITHM

To understand the sorts of problems we faced, it is important to have a general understanding of our particle algorithm. Initially, we place a number of particles on an isosurface. The particles move about on the surface and adaptively change their population until they reach a state of equilibrium. Depending on the surface, this can result in tens of thousands of particles evolving over dozens of time steps. Once the particle positions are fixed, they are used as vertices in a triangulation of the surface.

With each iteration of the system, particles can move, grow or shrink, and be born or die. Particle movement is based on forces exerted on each particle by its neighbors. The amount of force exerted by a particular particle is a function of its size and the distance between it and each of its neighbors. The size, in turn, is based on the curvature of the surface at the particle's current location combined with the repulsive forces from its neighbors. Birth and death are triggered by particles growing or shrinking past certain limits, indicating that the particle is either in an area that is under or over populated, respectively.

In calculating repulsion forces, we need to sum up the forces exerted on any one particle by those particles within a limited

neighborhood. To facilitate this operation, we created a data structure that links particles that are deemed to be close enough to impact one another's force calculations. Surface normal information is used to prevent linking particles that are spatially proximate, but that lie on non-adjacent surfaces.

In our system, surfaces can have edges when the volume boundaries intersect the isosurface. These boundaries create problems with respect to both the movement of particles near the edges and the triangulation of particles on an edge. To facilitate specialized handling of these particles, we created different particle types and subtypes.

To generate a surface from the particle positions after the system reaches equilibrium, we developed an advancing-front triangulation algorithm that makes use of the existing neighbor information and surface-normal information at each particle.

3 VISUAL DEBUGGING TOOLS

In this section, we describe the visual tools we developed to examine various aspects of our particle system. Each of the four sections relates to the tools used to visualize a particular data element or data structure in the system. The tools can be used statically to examine data values across all of the particles at a particular time, or they can be used dynamically to animate changing attributes of the system as it evolves over time.

3.1 Particle Placement and Size

For speed, we draw the particles as hexahedral approximations to spheres. The size of each sphere is set using the corresponding particle's radius of repulsion as the radius of the sphere. The particles are output at the end of each iteration so that we can follow their movement and the increases or decreases in population. We can superimpose the particles upon a conventionally produced isosurface of the data set to see how closely the particles follow the surface and whether their population density matches the curvature of the surface in that region.

In addition to seeing the particles, we want to be able to select a particular particle and know the contents of its associated fields such as the particle identifier, position, isovalue at that position, normal vector, curvature, and neighbor count. Knowing the particle identifier is especially useful because we can target a conventional debugger to *break* whenever that particle is accessed. Then we can follow the individual calculations for those particles that exhibit unexpected or unusual behavior.

For particle selection, we enable AVS's pick operation and we use the particle element's address as the value returned by the pick. We highlight the selected particle in green to provide visual feedback that we have picked the intended particle.

3.2 Neighbor Links and Surface Normals

To evaluate how particles are connected together in the web of neighbors, we draw a line between each pair of particles that share a neighbor link. Sometimes, it is useful to see each particle's normal vector, so we draw the normal as a straight line of length one starting at the particle's position as shown in Figure 1. The neighbor links and the normals are output at each iteration along with the particle hexahedra. Because the neighbor links, the normals, and the particles are created as three separate objects, AVS enables us to independently turn their display on and off and color them separately (though the hexahedra are colored by based on attributes, as is explained in the next section).

Sometimes the size of the particles obscures the neighbor links in areas of high particle density. If we wish to see just the particle location without the particle size information, we can turn the hexahedra off and darken the normals to distinguish them from the neighbor links (we typically use black normals with cyan neighbor links). By substituting the particle normal vectors for the hexahedra, we get a much better view of the dynamic neighbor link behaviors. We can watch the particles move about, breaking and forming neighbor links. Holes appear in the mesh of neighbor links when particles die. Later, neighboring particles around the hole move into the gap to fill it in. At any point in the simulation, we can disable the display update (which causes the simulation to pause) and turn on the hexahedral particle display to see particle sizes or other attributes that are encoded in each particle's coloration.

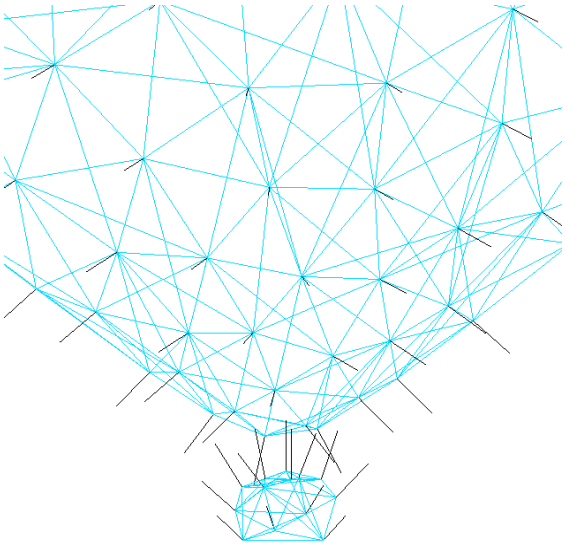

Figure 1: Normals for each particle are drawn in black, neighbor links in cyan.

3.3 Particle Attributes

There are a number of particle attributes that we are interested in tracking across the entire system. Initially, we tried to monitor these attributes by printing out statistics such as the low, high, and average values for these attributes. But without knowing the spatial distributions, the information was of limited use. So we decided to color each of the particles based on the particle's value for the attribute of interest.

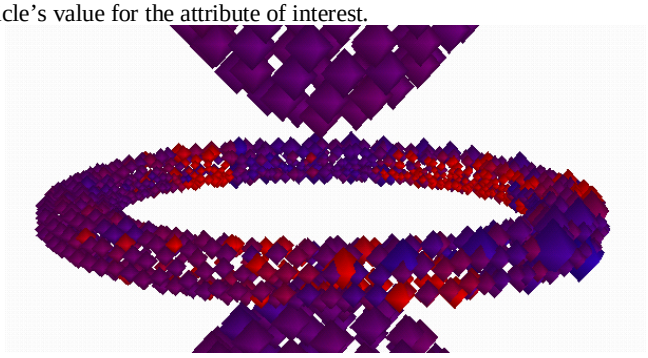

Figure 2: Particles colored by energy level.

We use set of radio buttons on the AVS control panel to select which attribute to use in coloring the particles during the

next iteration. These attributes include the particle energy, type, ratio of repulsion radius to desired repulsion radius, number of neighbors, and particles born within the last several time steps. In each case, the meanings of the colors are unique to the attribute.

We are interested in particle energy values because energy levels are involved in the particle system reaching equilibrium, and hence termination. Following Huang's example, we color energy along a continuum between blue and red, with blue representing no energy and red representing high energy, as is shown in Figure 2. The middle of the range, is the desired energy level for the system. So over the lifetime of the particle system the color of the particles should shift from blues and reds to a uniform purple. Because the energy range is artificially limited and the range can greatly exceed the specified maximum, we distinguish particles whose energy values are far outside the range by coloring them green.

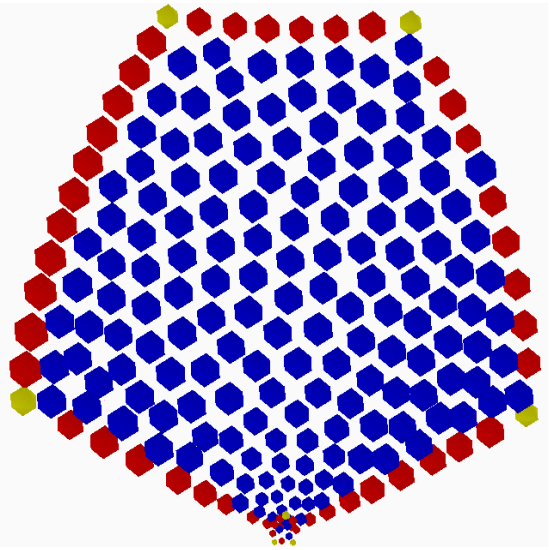

Figure 3: Particles colored by particle type.

Sometimes it is useful to color particles based on their types because particle type can change over the course of time. Then particle type transitions can be monitored and particle behaviors can be compared against types. When particles are colored by type, *interior* particles are colored blue, *edge* particles are red, *corner* particles (where more than one edge subtype intersect) are green, and *ghost* particles (similar to *edges*, but based on processor boundaries instead of a volume boundaries) are colored cyan.

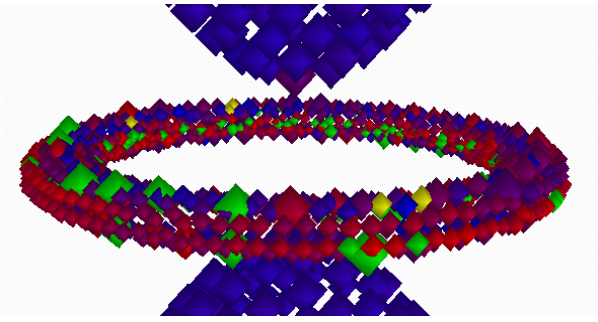

Figure 4: Particles colored by the ratio of repulsion radius to desired repulsion radius.

Each particle's size (repulsion radius) is calculated using a feedback equation that combines a curvature-based desired repulsion radius with the population pressures in the local neighborhood of the particle. Over time, we want the ratio of repulsion radius to desired repulsion radius to approach one. To get some idea of the number and location of particles that are approaching this ratio at various times during the evolution of the system, we created a particle coloration type based on this ratio. The ratio range from .5 to 1.5 is mapped into the blue to red color range so that the ideal ratio falls in the middle (purple). Particles whose ratios are less than .5 are colored yellow, and particles whose ratios are greater than 1.5 are colored green. This way, particles whose size is outside the expected range are highlighted and yet distinguishable as to whether their ratio is high or low.

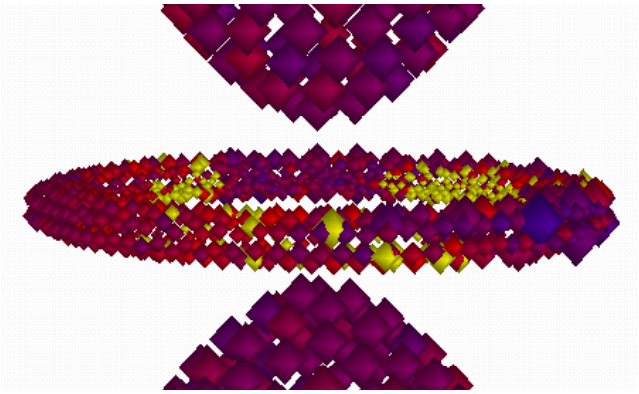

Figure 5: Particles colored by the number of neighbor links.

Another attribute of great interest is the number of neighbors for each particle. Much of the tuning of our particle system involved identifying how to keep the number of neighbor links low, while still preserving sufficient connectivity between particles to obtain good triangulations. Although the information is contained in the neighbor links, the number of particles combined with the difficulty of discerning in three-dimensions which links originated from which particles made another approach necessary. It is much simpler to color the particles based on the number of neighbors. Because the ideal number of neighbors is 6, we map the range from 0 to 12 into the blue to red color range. Particles with 6 neighbors are colored purple. Particles with neighbor counts in the range from 13 to 50 are colored yellow, and particles with more than 50 neighbors are colored green.

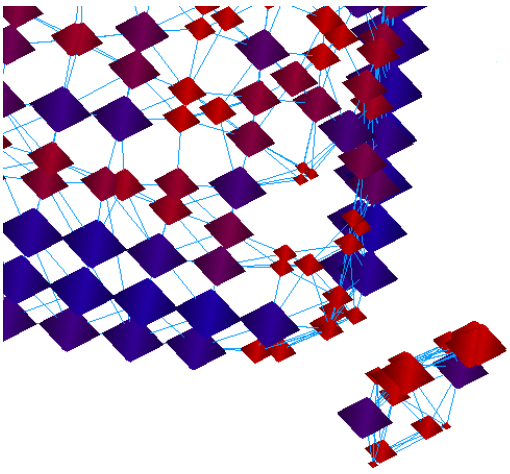

Figure 6: Particles colored by time steps since birth.

In adjusting birth/death parameters, we found that the sheer number of particles often makes it difficult to discern which particles have just been born, and which particles are older. Because the particle identifiers are assigned in counting order, we can keep track of the maximum particle identifier at a particular step. We only find it useful to distinctly color the previous three steps. Particles that have just been created are colored red. Particles created in the previous time step are colored magenta. Particles from the time step before last are purple, and all older particles are colored blue.

3.4 Triangle Generation Order

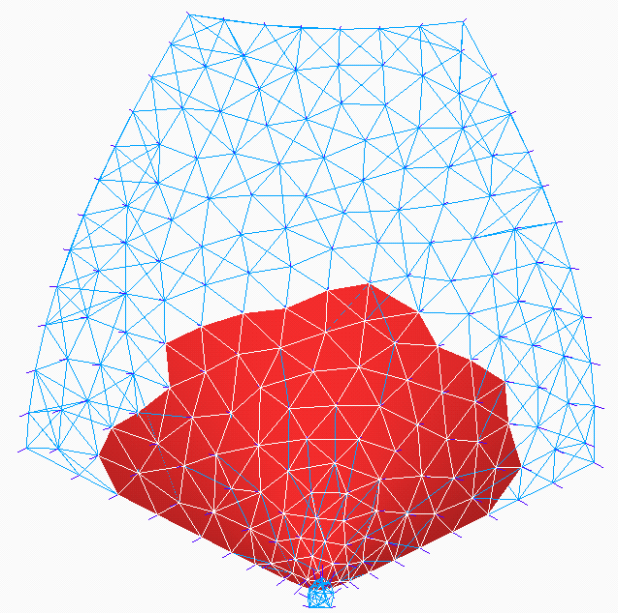

Figure 7: Incremental triangle output.

During the debugging of this algorithm we sometimes would end up with overlapping triangles or holes in the surface. In tracking down the underlying causes for these errors, we made use of an AVS rendering mode in which triangles are drawn outlined in white. We combined this rendering style with our own option to update the display of the triangulated surface after each section of triangles is generated. This technique enabled us to watch the triangulation evolve and to use the picking feature to identify which particles were involved when mistakes are made. We could then go back in the debugger and step through the triangulation code to the point when the erroneous triangles were created. This proved to be extremely useful in debugging the triangulation code.

4 RESULTS AND CONCLUSION

As visualization systems have evolved, the difficulties of debugging visualization codes have increased. In this paper, we examined a particular particle system visualization. While writing the code, we found that we had underestimated the difficulties inherent in implementing and working with complex dynamic systems of this sort. These complexities led to our development of a set of visual debugging methods.

As a result of this work we have reached three conclusions. First, for the very same reasons we develop visualization methods for extracting information from data, we should see visualization as a way of extracting information about our software systems. Second, visual debugging tools must be application specific. Third, incrementally adding visual debugging functionality to our code showed us exactly what we needed to see and was simpler and faster to implement than trying to learn to use existing visual debugging systems.

5 ACKNOWLEDGEMENTS

The DOE Mathematics, Information, and Computer Science Office funded this research. The work was performed at Sandia National Laboratories. Sandia is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. The use of AVS in this work is not a product endorsement by Sandia National Laboratories.

REFERENCES

[1] Bosson, F. Anisotropic Mesh Generation with Particles. Masters Thesis, Carnegie Mellon University, 1996.

[2] Brown, M. Exploring Algorithms Using Balsa-II. *Computer*, 21 (5): 14-36. May 1988.

[3] Crossno, P. and E. Angel. Isosurface Extraction Using Particle Systems. *Proceedings of Visualization '97*, pages 495-498. October 1997.

[4] Fournier, A. and W. Reeves. A Simple Model of Ocean Waves. *Computer Graphics (SIGGRAPH 86 Conference Proceedings)*, 20 (4): 75-84. August 1986.

[5] Huang, M., et al. Untangling Knots by Stochastic Energy Optimization. *Proceedings of Visualization '96*, pages 279-286. October 1996.

[6] Price, B., et al. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing*, 4(3): 211-266. September 1993.

[7] Reeves, W. and R. Blau. Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. *Computer Graphics (SIGGRAPH 85 Conference Proceedings)*, 19 (3): 313-322. July 1985.

[8] Reeves, W. Particle Systems – A Technique for Modeling a Class of Fuzzy Objects. *Computer Graphics (SIGGRAPH 83 Conference Proceedings)*, 17 (3): 359-376. July 1983.

[9] Reynolds, C. Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics (SIGGRAPH 87 Conference Proceedings)*, 21 (4): 25-34. July 1987.

[10] Shimada, K. Anisotropic Triangular Meshing of Parametric Surfaces via Close Packing of Ellipsoidal Bubbles. *6th International Meshing Roundtable*. 1997.

[11] Sims, K. Particle Animation and Rendering Using Data Parallel Computation. *Computer Graphics (SIGGRAPH 90 Conference Proceedings)*, 24 (4): 405-413. August 1990.

[12] Smith, A. Plants, Fractals, and Formal Languages. *Computer Graphics (SIGGRAPH 843 Conference Proceedings)*, 18 (3): 1-10. July 1984.

[13] Stasko, J. Tango: A Framework and System for Algorithm Animation. *Computer*, 23 (9): 27-39. September 1990.

[14] Szeliski, R. and D. Tonnesen. Surface Modeling with Oriented Particle Systems. *Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, 26 (2): 185-194. July 1992.

[15] Tal, A. and D. Dobkin. Visualization of Geometric Algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 1 (2): 194-204. June 1995.

[16] Witkin, A. and P. Heckbert. Using Particles to Sample and Control Implicit Surfaces. *Computer Graphics Proceedings, Annual Conference Series 1994*. 269-277. 1994.

[17] Wolfenbarger, P. et al. A Global Minimization-Based, Automatic Quadrilateral Meshing Algorithm. *7th International Meshing Roundtable*. 1998.

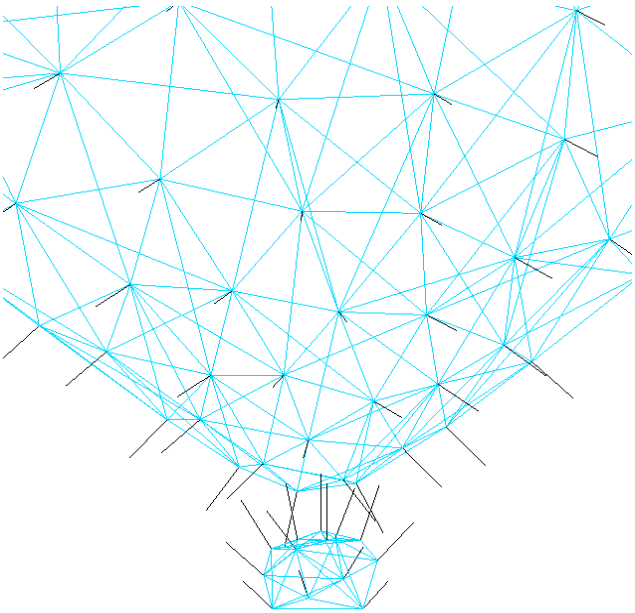

Figure 1: Normals for each particle are drawn in black, neighbor links in cyan.

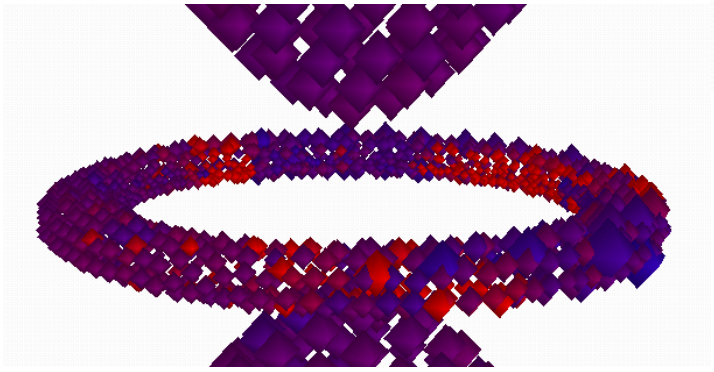

Figure 2: Particles colored by energy level.

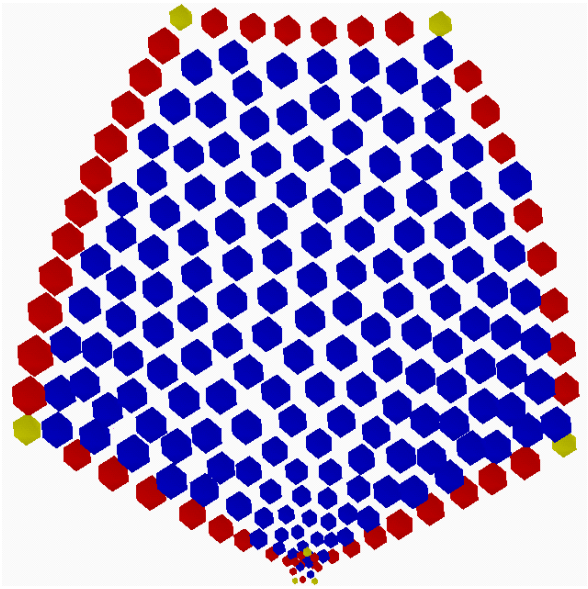

Figure 3: Particles colored by particle type.

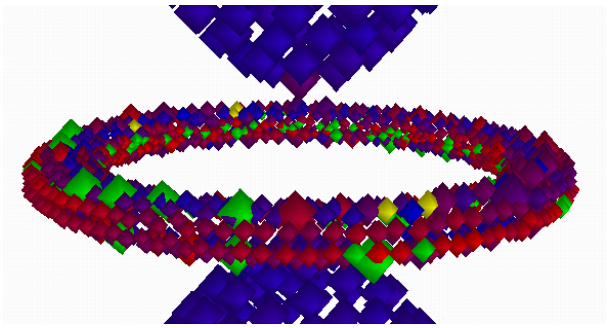

Figure 4: Particles colored by the ratio of repulsion radius to desired repulsion radius.

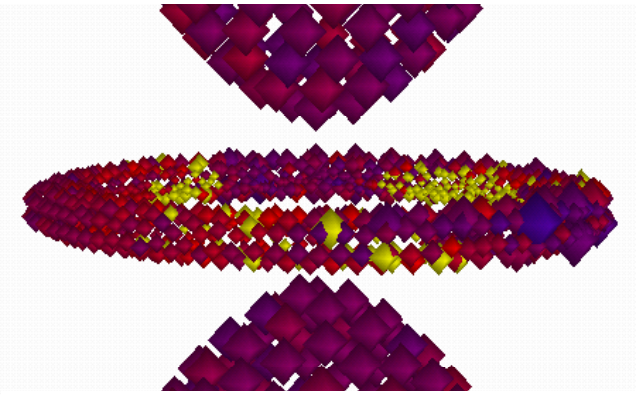

Figure 5: Particles colored by the number of neighbor links.

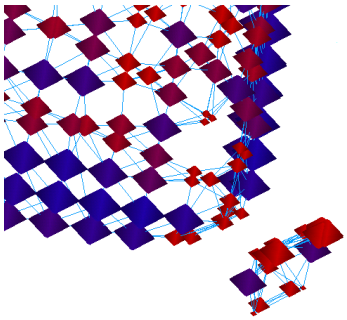

Figure 6: Particles colored by time steps since birth.

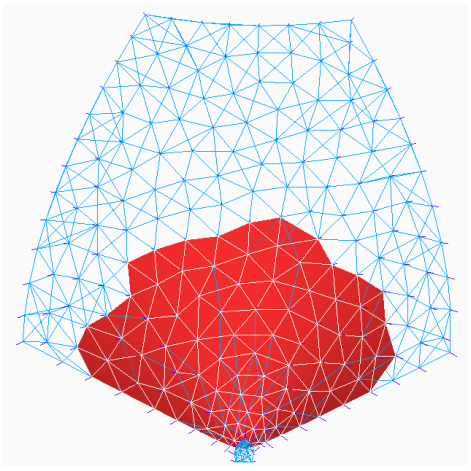

Figure 7: Incremental triangle output.