




Real-time Slicing of Data Space



Roger A. Crawfis¹

Lawrence Livermore National Laboratory
P.O. Box 808, L-301
Livermore, CA 94551
crawfis@llnl.gov

Abstract

Real-time rendering of iso-contour surfaces is problematic for large complex data sets. In this paper, an algorithm is presented that allows very rapid representation of an interval set surrounding a iso-contour surface. The algorithm draws upon three main ideas. A fast indexing scheme is used to select only those data points near the contour surface. Hardware assisted splatting is then employed on these data points to produce a volume rendering of the interval set. Finally, by shifting a small window through the indexing scheme or data space, animated volumes are produced showing the changing contour values. In addition to allowing fast selection and rendering of the data, the indexing scheme allows a much compressed representation of the data by eliminating "noise" data points.

Keywords

Isocontour, iso-surface, contour surface, volume rendering, splatting, animation, data partitioning, interactive, real-time, scientific visualization, compression.

Introduction

Computer simulations of fluid flow, the atmosphere, and other phenomena generate large amounts of data containing three-dimensional scalar fields. Typical techniques for examining these scalar fields are isocontour surfaces and volume rendering. Contour surfaces can be approximated by a set of triangles [15, 19] that are passed down for rendering. These triangles can then be stored and re-rendered for different view points. With accelerated graphics hardware, this rendering can be fairly interactive. Changing the contour value (the value of the scalar field defining the surface), however, requires traversing the data set and generating a new set of triangles.

This can be a costly operation. When the data is arranged according to its three-dimensional coordinates, calculating the contour surfaces requires examining each data cell. Avoiding this inherently slow traversal of the entire data set is one of the primary goals of this paper.

The geometric representation of an isosurface can produce a very large display list of triangles, normals, and associated information. Volume rendering [10] avoids the costly calculation and more importantly the storage of an isosurface. Several different techniques have been developed over the past several years for volume rendering. Baining Guo [8], Fujishiro, et. al. [7] and Max et. al. [18] describe techniques for segmenting a volume into discrete intervals that can then be volume rendered. Guo focuses mainly on extracting thin layers (interval sets) near an iso-contour surface using alpha shapes which can then be rendered using tetrahedral projection volume rendering techniques.

A rough approximation to a contour surface can also be achieved using volume rendering with a transfer function for the opacity that is non-zero only near the contour value. This paper presents a technique to rapidly shift such a transfer function across the possible data values, resulting in time-dynamic slicing of the data space. This paper merges ideas from iso-surface generation and volume rendering. The next two sections discuss the related work in each of these areas. A detailed description is then given for the fast volume rendering of data space slices. The mechanism used to animate these data slices is then presented, followed by some application results and conclusions.

Rapid Iso-surface Extraction

Several researchers have examined different aspects of generating efficient iso-contour surfaces. Here, I will summarize work developed to aid or avoid the traversal (search) of the data set for contour surface fragments. Wilhelms and Van Gelder[29] use octrees to aid in the search for iso-contour surfaces in regular or curvilinear grids. Each octree node stores the minimum and maximum data values of the subvolume below it. When calculating an iso-contour surface, the octree is traversed and branches whose minimum and maximum lie to one side of the contour value can be safely skipped. For data sets with large regions of empty space or uninteresting data

¹Current address:
The Ohio State University,
395 Dreese Laboratory,
Columbus, OH 43210-1277
crawfis@cis.ohio-state.edu

values, this can provide a substantial speed-up over the straight-forward approach of examining each data cell. For data varying widely or containing high frequencies, this approach is less-advantageous.

Saito [21] describes algorithms for real-time previewing of 3D scalar fields. He constructs prioritized lists of graphics primitives to represent a specific iso-contour surface. The data is resampled hierarchically leading to a prioritized set of points. The points are distributed uniformly in 3D space rather than based on data. Accurate rendering of a surface would require traversing the entire list of primitives, and insuring an adequate sampling of the 3D space.

Shen and Johnson [22] present a technique to accelerate the traversal of cells within a finite-element or unstructured mesh. They create two data structures for indexing the cells. The minimum and maximum data values for each cell (tetrahedra in their implementation) are first calculated. A minimum list is created that sorts the cells using the minimum data values. A sweeping list is also calculated and sorted using the maximum data values. The minimum list has pointer references to the corresponding cells in the sweeping list, while the sweeping list has a flag indicating whether the minimum is below the contour value. These flags are easily updated (incrementally) by traversing the minimum list until a cell with a minimum greater than the contour value is encountered. A binary search is used on the sweeping list to find the first cell with a maximum value greater than the contour value. All cells after this cell with their flag set (indicating the cell has a minimum below the contour value) are then candidates for contour surface calculations. This allows Shen and Johnson to quickly determine the cells that contribute to a contour surface, avoiding a full traversal of the data set. A high gradient, even in one cell, can still require a lengthy traversal of the maximum list.

Related Volume Rendering Background

Volume rendering can produce either density clouds or the appearance of shaded surfaces. For shaded surfaces, the voxels in a volume are colored according to a shading formula based on the gradient of the field. (See Levoy [13]). Once a shaded volume is generated there are many techniques for volume rendering it [6, 18, 20, 23, 24, 27]. Traditional volume rendering is computationally expensive, precluding real-time interaction with the data set. Several techniques have been explored to accelerate volume rendering. Work similar to that on octrees for isosurfaces has been investigated by several authors [5, 11, 12, 14]. Yagel and Shi [30] describe a technique for leaping directly to the first voxel that may contain the contour surface. All of these techniques preserve the three-dimensional lattice structure of their data. More efficient indexing schemes can be found by organizing the data according to the values.

Ihm and Lee [9] present an algorithm for efficient volume rendering of regular grids, that organizes the data according to its values, rather than their coordinates. Their algorithm is only for regular grids and targeted towards medical data where a few key data values or material classifications are used. The entire data set is represented three times: once for each major axis. Each axis is then subdivided into slice planes. Within each slice plane, the data values and their corresponding locations within the slice are ordered into discrete buckets according to the data values. For each view, the appropriate set of slices are traversed in a front-to-back or a back-to-front order and all data values within a specified range are splatted into the

image using the technique of Westover [28]. Since this splatting technique deals with each data point independently, no information about cell connectivity is required for data defined on a regular grid.

This paper presents a method to very quickly ascertain the data points in an interval set for volume rendering via splatting. Most volume rendering techniques require a back-to-front or front-to-back sorting of the data. In the case of ray-tracing, the sampling is either feed-forward or feed-backward. For regularly gridded data, a simple traversal of the 3D array in a back-to-front order is trivial. For this application, we are only concerned with interval sets with a relatively small thickness. Hence, only a fraction of the possible data sets may contribute to the overall image. For fast selection of the desired data points, the data needs to be ordered according to the desired data value as in Ihm and Lee.

One final observation is needed before proceeding with the algorithm presented in this paper. Max, et. al. [17] provide an algorithm for the fast generation and display of smoke through a vector field. For this application, they proved that given a homogenous color for the smoke, the order of rendering is immaterial. In other words, no back-to-front or front-to-back sorting is required for density clouds employing only a single color. Since we are interested in representing data "close" to a contour surface, a constant color is also a reasonable choice here.

Fast Volume Rendering of Data Slices

Combining the result of Max, et. al., previous work on hardware assisted splatting [3, 4] and the ideas presented in Shen and Johnson [22] and Ihm and Lee [9] leads us to a very simple algorithm for representing and rendering "thin" interval sets. The algorithm is simply to maintain a sorted list of data values along with their corresponding coordinate positions. Given any contour value, data points within a specified tolerance of this contour value can easily be determined using binary search on the sorted list. This usually small set of data points are then splatted to the screen using a constant color and opacity by the technique described in [3]. It should be noted, that there is nothing special about the contour specification or using thin slices. The data organization and splatting will work for arbitrary ranges of data values. The splatting will reconstruct and project the data values within that range. Since they are all a homogeneous color, the sorting order can safely be ignored. The slices can potentially limit the number of data points to a very small fraction (one-hundredth or less). This then allows for animation through even the largest data set. The rest of this section describes implementation issues and user parameters.

In constructing the list, the user specifies the desired range of values that are of interest. Since in many applications certain values are of little significance, these values can be discarded. The algorithm traverses the data set once, creating a sorted list of only those values that fall within the user's desired range.

This linear list is encapsulated into a C++ class, extending the Open Inventor [26] graphics classes. The sorted list is stored as an array of four component vectors. The first three components specify the 3D coordinates, while the last coordinate specifies the data value. The coordinate boundaries are also specified in the members, `bboxMin` and `bboxMax`, as

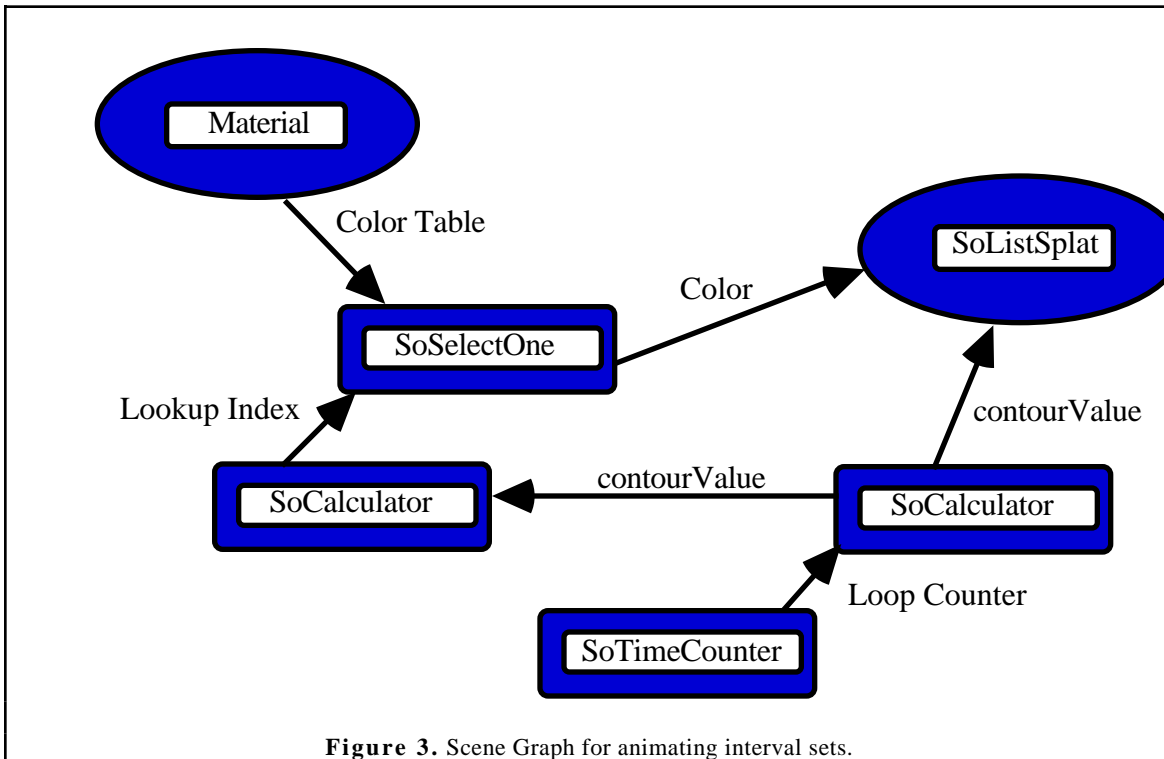


Figure 3. Scene Graph for animating interval sets.

two three-dimensional vertices. The desired contour value is also specified as is the thickness of the interval set.

On each redraw, four operations are performed. The viewing transformation is inquired and a new rotation matrix is determined to orient the volume with projection space. The rendering pipeline is then initialized for any global settings - blending functions, specification of the texture splats, etc. The list is then traversed. A binary search is performed to find the entries one delta before and one delta after the desired contour value. All of the entries within the resulting range are then individually rendered using a single texture mapped square containing the reconstruction function of Crawfis and Max [3] in the alpha channel. Each splat is drawn in the desired color, with the desired transparency and scaled by a user parameter, Splat Size below. All of these options can be set by the user of the C++ class, or by the user of the IRIS Explorer [1] module.

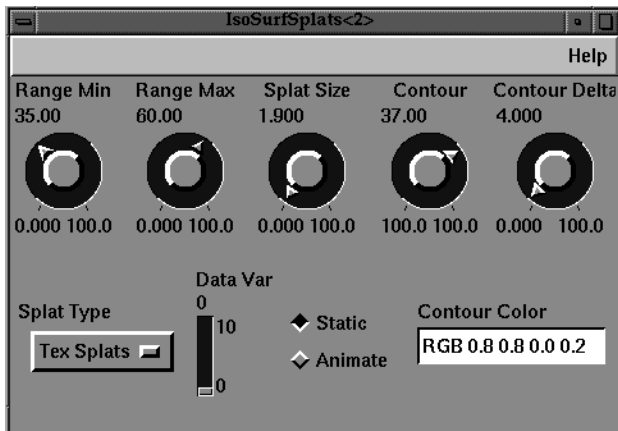


Figure 2. IRIS Explorer Control Panel

Figure 2. shows the user interface of the Explorer module.

Only changes to the parameters *Range Min* and *Range Max* requires traversing the data and rebuilding the list. The other parameters simply change the selection or redraw attributes and, as is evident in the video, take affect instantly. In the control widget of Figure 2, data values in the range 35.0 to 60.0 are of interest to the user. The contour value is set to 37.0, and the delta is set to a value of 4.0, selecting data points with values between 35.0 and 41.0. An arbitrary color and opacity can be specified in the textual box.

This leads to a fairly powerful and fast technique. For large data sets, the rendering of a moderate number of texture splats can require times greater than a half a second, disrupting the smooth rotation and analysis. Since for these very large data sets, the texture splats typically project to a small number of pixels, a quick and dirty solution is to use semi-transparent point primitives. A second *Splat Type* is provided for this function. In addition to having faster hardware rendering times, the points do not require any transformations to orient them parallel to the screen and can be cached for future redraws, hence they offer a substantial speed-up over the textured splats. Interactive exploration is thus possible with the point splats, while a more accurate representation can then be generated using the texture splats. The video clips show the use of point splats in slicing through a one million data point problem at interactive rates.

Finally, applying this technique to curvilinear and unstructured meshes allows a somewhat crude but useful exploration of the data space within those meshes. The reconstruction properties of splatting on a regular grid break down for these mesh topologies, and the user has to take care in examining these results. The ability to quickly index and represent the data allows for a rough examination of the data. Higher quality tools such as isocontour generators can then be

applied to focus on certain regions of contour values. Figure 10 shows the SoListSplat class applied to a finite-element calculation of fluid flow past a cylinder.

Animating the Interval Sets

As previously stated, one of the goals of this work is to allow rapid enough calculation and display of the interval sets for interactive exploration or real-time animation. Changes in a contour surface's shape from one value to another incrementally larger or smaller value is useful for understanding the 3D characteristics of the field. To allow for fast changes in the specified contour value, we employed Open Inventor engines [25]. It would be slightly more efficient to embed the animation controls into a new C++ subclass, but the flexibility of the Open Inventor engines proved more valuable. Sliding the contour "window" through the specified data range allows us to animate the data slicing. A SoTimeCounter engine connected to a SoCalculator engine will produce new contour values at specified time intervals. Connecting the output of the SoCalculator to the SoListSplat's contourValue member allows for smooth animation of the interval sets. Keeping the color constant throughout this animation makes it difficult for the user to distinguish the relative values at each time step. By providing a color table and an SoSelectOne engine, we can modify the color (and transparency) as the contour value changes. Figure 3. shows a resulting scene graph that will loop through contour values. The SoTimeCounter will generate a redraw command at periodic intervals.

The SoTimeCounter could be used as the Lookup Index for the color, provided the color table has the same number of entries as the steps in the SoTimeCounter. Simply looping through the data values leads to distracting jumps from the end of the range to the beginning. For smoother animations, the data values are made to "swing" by placing a cosine term in the Calculator for the contourValue. The color table can follow this nonlinear path through the data values by hooking the SoCalculator for the contour value to the SoCalculator for the Lookup Index. The second SoCalculator then maps the data values into the color table. This is shown by the thin line in Figure 3.

Figure 4 shows a modified control panel for the IRIS Explorer module similar to Figure 2 Here, the *Animate* button is selected and the color specification has been replaced with two new widgets. The color specification is now being taken from an input port on the module. By default, the timer has 256 steps



Figure 4. IRIS Explorer Control Panel - Animation set

(corresponding to a typical 256 entry color table). The *Frequency* dial specifies how many times per second the timer should loop through its steps. A value of 1.0 will force the timer to increment 256 times per second. A value of 0.1 only fires 25.6 times per second and takes ten seconds to complete a cycle (swing) through the data values. The *Step* widget allows the timer to count by increments larger than one. Hence, a *Step* of 8 and a frequency of 0.5 will try to update and redraw the list 16 times per second. The accompanied video shows several segments of real-time animation of the interval sets.

Memory Issues

The simple data structure used in this algorithm can lead to a significant reduction in the memory used for visualizing the data. For the percent cloudiness in a global climate simulation, keeping only the data values above twenty-five percent cloudiness selects roughly one tenth of the total data points. The sorted linear list requires the storage of the (x,y,z) or (i,j,k) components with each data value, hence the savings for regular grids may be diminished. For data sets having each dimension smaller than 256, the (i,j,k) coordinates can fit into three bytes. However, if the user selects the entire range of data values, the space requirements can be greater than the raw data for regular grids. In practice, the fraction of points is typically around one-tenth. Table 1 lists several data sets, number of nodes in the data set, the maximum and selected data ranges, the number of nodes within that range and the resulting percentage of nodes selected. The actual storage costs will depend on the data types and mesh topology. For regular grids, the i,j,k positions need additional storage, whereas for unstructured grids this storage is needed regardless.

The percentage figures in Table 1 support the claim that usually 90% of the data is uninteresting. For the HIPIP data, interesting data values lie both in the large negative range as well as the high positive range, accumulating upto 25% of the data. For the Climate Data, it should be noted that 90% of the data range gives a ten fold compression. For very large data set, where even the interesting data becomes unwieldy, this selection mechanism can be used to focus on small logically coherent portions of the data at a time.

Application Results

This algorithm has been applied to several data sets. With even the largest data set, the Global Climate model, real-time slicing through the data values is possible on our old 100MHz R4400 Onyx/VTX. Figure 5 shows a volume rendering an interval set of the flow magnitude through an idealized HEPA filter. Figure 6 represents the percent cloudiness in a global climate simulation. A rather large delta is used to give an overall feel for the scalar variable. Under interactive exploration, a sudden "popping" was noticed when we cycled through interval sets. By redefining the range and setting the delta to a very small increment, it was easily determined that there is a sudden increase in grid points with 100 percent cloudiness. This variable is also interesting in that isosurfaces generally do not produce very meaningful representations, due to the high frequency of the data. Figure 7 and 8 illustrate the technique applied to proton density. Figure 7 uses texture splats for the interval set whereas Figure 8 uses point splats. OpenGL [2] allows point sizes greater than a single pixel, and the points splats typically use sizes greater than 10 pixels wide. The data slicing brings out the regular grid of the

simulation, which can either be beneficial or distracting. Figure 10 shows high values of vorticity in a finite-element simulation of fluid flow past a post, on an unstructured grid.

The video accompanying this paper highlights the interactive speed of this technique. For the Global Climate model, a mesh of 320 by 160 by 19 is used generating close to a million data points. The algorithm can cycle through rather thick data slices using textured splats at slightly over a frame a second on an SGI Onyx/VTX running with R4400 at 100Mhz. We hope to soon upgrade this hardware to faster CPU's and graphics engines, allowing real-time exploration of this data set. This is our slowest application data. All of the other data sets listed above give real-time performance. Even the SOD data with slightly more data points than the climate model can be cycled through data slices in real-time.

Conclusions

An algorithm for rapid selection and representation of scalar field data defined on regular grids has been presented. It allows for the fast selection of new intervals within the data field. The algorithm is suitable for fast cycling through the data space of a simulation. Its simplified data structure allows for substantial compression of the data, and a rudimentary representation of non-regular grids or even scattered data sets.

The blazing speed of this technique and the compact nature of the sorted list are ideally suited for applications with high demands for interactivity, such as within a virtual reality setting. It is also useful for off-loading reasonable data sets from very large simulations on massively parallel supercomputers. After the user garners a quick overview with this technique, slower, more quantitative techniques can be used to render the data in more detail. These techniques can even be included in the same representation, as is evident from the images and the video, by performing the volume rendering last and testing but not setting the z-buffer during the splatting process. (See Crawfis and Max [3].

The use of animation to rapidly cycle through the data space has been shown to be a valuable technique in understanding a scalar field. Details that would be hard to discern from isocontour surfaces or even traditional volume rendering become clear using motion through the data space dimension.

While the algorithm represents data within a given range, it does so in discrete space rather than on a reconstructed continuous representation of the scalar field. Areas of high gradient can thus be missed with small intervals. As such, the algorithm does not truly represent interval sets or contour surfaces. It should properly be considered as a slice plane through data space. Its interactive speeds outweighs the disadvantages of the imprecise contour surfaces or interval

sets. Furthermore, the algorithm makes no assumptions about the data, representing only the raw data.

Future Work

The performance and useful results of this algorithm would be ideally suited for integration into a CAVE or virtual-reality environment. It is hoped that this goal can be accomplished in our next project. Time can be thought of as another dimension. The data structure presented here could be used to index through time as well. Here, a fixed range of values could be selected for each time slice, or the data could be arranged in an array of SoListSplat's with time as the array index. The textured splats are significantly slower than the point splats. A future goal is to look into the performance of the texture splats and optimize them. The Range Min and Range Max specification is used for the bounds in the animation. In practice, it would be better to specify the animation range separately. Furthermore, it would be better to allow different disconnected ranges to be specified for data sets like the Hipip data set. For curvilinear and unstructured data sets, the splatting algorithm used here is not accurate. The work by Mao, et. al.[16] should be incorporated to sample these topologies accurately enough for a correct representation. Finally, the quicksort routine, qsort, is used in creating the sorted lists. The efficiency of this routine looks promising enough to depth sort the selected points at each redraw and splat the points in back-to-front order. This would allow color and shading into the volume renderer.

Acknowledgments

I am indebted to the many people who provided the data used in the visualizations presented in this paper. The climate data is courtesy of Jerry Potter, the Program for Climate Model Diagnosis and Intercomparison at Livermore, and the European Centre for Medium-range Weather Forecasts. The finite-element data is courtesy of Mark Christon. The astrophysics shock wave data is courtesy of Richard Klein. The Hipip and Sod data are from the Chapel Hill Volume Rendering Test Data Set I and are courtesy of the Scripps Clinic. The HEPA filter data is courtesy of Bob Corey. I would also like to thank Nelson Max, Rebecca Springmeyer and Torsten Möller for many helpful comments. This work was performed under the auspices of the US Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-ENG-48.

Data Set	# of vertices	Data Min/ User Min	Data Max / User Max	# vertices w/in range	percentage
HIPIPH	262,144	-0.56/ 0.006	0.58/ 0.58	29,111	11%
HEPA	718,911	0 / 40	71.8 / 71.8	67,667	9.4%
Climate	972,800	0.0 / 0.1	1.0 / 1.0	94,798	9.7%
Post	16,096	0.0 / 2.0	16.8 / 16.8	1,498	9.3%
SOD	1,091,444	0.0 / 70.0	255 / 255	12,976	1.2%
Shockwave	1,728,000	0.0 / 10.0	71.05 / 71.05	8,356	0.5%

Table 1. Various compression ratios for different data sets.

References

1. *IRIS Explorer User Guide*. 1995: Numerical Algorithms Group Ltd.
2. Board, O.A.R., et al., *OpenGL Programming Guide*. Release 1 ed. 1993, Reading, MA: Addison-Wesley. 516.
3. Crawfis, R. and N. Max. *Texture Splats for 3D Vector and Scalar Field Visualization*. in *Visualization '93*. 1993. San Jose, CA: IEEE Computer Society Press.
4. Crawfis, R., N. Max, and B. Becker, *Vector Field Visualization*. *Computer Graphics and Applications*, 1994. **14**(5): p. 50-56.
5. Danskin, J. and P. Hanrahan. *Fast Algorithms for Volume Ray Tracing*. in *Proceedings of the 1992 Workshop on Volume Visualization*. October 1992. New York: ACM SIGGRAPH.
6. Drebin, R.A., L. Carpenter, and P. Hanrahan, *Volume Rendering*. *Computer Graphics*, August 1988. **22**(4): p. 64-75.
7. Fujishiro, I., Y. Maeda, and H. Sato. *Interval Volume: A Solid Fitting Technique for Volumetric Data Display and Analysis*. in *Visualization '95*. 1995. Atlanta, GA: IEEE Computer Society Press.
8. Guo, B. *Interval Set: A Volume Rendering Technique Generalizing Isosurface Extraction*. in *Visualization '95*. 1995. Atlanta, GA: IEEE Computer Society Press.
9. Ihm, I. and R.K. Lee. *On Enhancing the Speed of Splatting with Indexing*. in *Visualization '95*. 1995. Atlanta, GA: IEEE Computer Society Press.
10. Kaufman, A., ed. *Volume Visualization*. . 1991, IEEE Computer Society Press: Los Alamitos, CA. 479.
11. Laur, D. and P. Hanrahan, *Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering*. *Computer Graphics*, July 1991. **25**(4): p. 285--288.
12. Levoy, M., *Efficient Ray Tracing of Volume Data*. *ACM Transactions on Graphics*, July 1990. **9**(3): p. 245-261.
13. Levoy, M., *Display of Surfaces from Volume Data*. *IEEE Computer Graphics and Applications*, May 1988. **8**(5): p. 29-37.
14. Levoy, M. and R. Whitaker, *Gaze-Directed Volume Rendering*. *Computer Graphics*, March 1990. **24**(2): p. 217-223.
15. Lorensen, W.E. and H.E. Cline, *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. *Computer Graphics*, July 1987. **21**(4): p. 163-169.
16. Mao, X., L. Hong, and A. Kaufman. *Splatting of Curvilinear Volumes*. in *Visualization '95*. 1995. Atlanta, GA: IEEE Computer Society Press.
17. Max, N., B. Becker, and R. Crawfis. *Flow Volumes for Interactive Vector Field Visualization*. in *Visualization '93*. 1993. Los Alamitos, CA: IEEE Computer Society Press.
18. Max, N., P. Hanrahan, and R. Crawfis, *Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions*. *Computer Graphics*, 1990. **24**(5): p. 27-33.
19. Nielson, G.M. and B. Hamann. *The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes*. in *Visualization '91*. 1991. Boston, Massachusetts: IEEE Computer Society Press.
20. Sabella, P., *A Rendering Algorithm for Visualizing 3D Scalar Fields*. *Computer Graphics*, August 1988. **22**(4): p. 51-58.
21. Saito, T. *Real-time Previewing for Volume Visualization*. in *1994 Symposium on Volume Visualization*. 1994. Washington, D.C.: ACM Press.
22. Shen, H.-W. and C.R. Johnson. *Sweeping Simplices: A Fast Iso-surface Extraction Algorithm For Unstructured Grids*. in *Visualization '95*. 1995. Atlanta, GA: IEEE Computer Society Press.
23. Shirley, P. and A. Tuchman, *A Polygonal Approximation to Direct Scalar Volume Rendering*. *Computer Graphics (San Diego Workshop on Volume Visualization)*, Nov. 1990. **24**(5): p. 63-70.
24. Upson, C. and M. Keeler, *V-BUFFER: Visible Volume Rendering*. *Computer Graphics*, August 1988. **22**(4): p. 59-64.
25. Wernecke, J., *The Inventor Mentor*. Release 2 ed. 1994, Reading, MA: Addison-Wesley. 514.
26. Wernecke, J., *The Inventor Toolmaker*. Release 2 ed. 1994, Reading, MA: Addison-Wesley. 301.
27. Westover, L., *Footprint Evaluation for Volume Rendering*. *Computer Graphics*, August 1990. **24**(4): p. 367-376.
28. Westover, L. *Interactive Volume Rendering*. in *Proceedings of the Chapel Hill Workshop on Volume Visualization*. May 1989. Chapel Hill, NC.
29. Wilhelms, J. and A. Van Gelder, *Octrees for Faster Isosurface Generation*. *ACM Transactions on Graphics*, 1992. **11**: p. 201-227.
30. Yagel, R. and Z. Shi. *Accelerating Volume Animation by Space-Leaping*. in *Visualization '93*. 1993. Boston, Massachusetts: IEEE Computer Society Press.

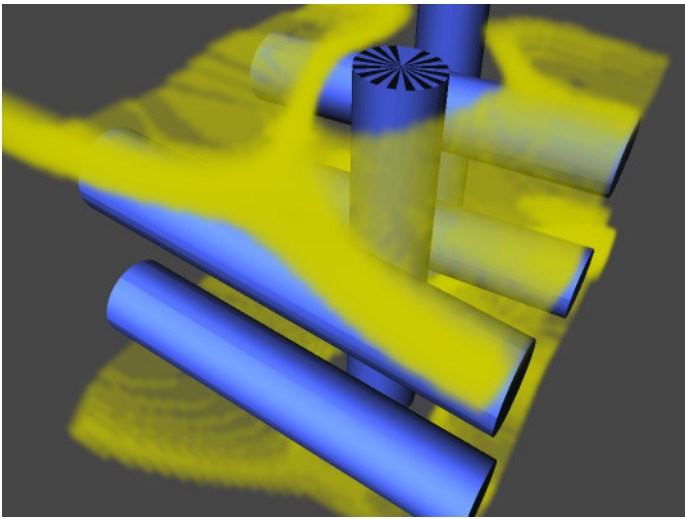


Figure 5. Interval set of velocity magnitude through idealized HEPA filter.

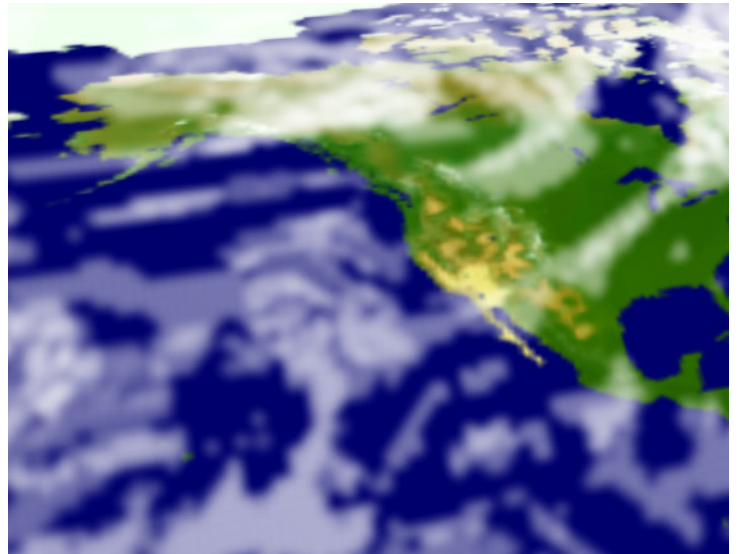


Figure 6. Interval set of percent cloudiness

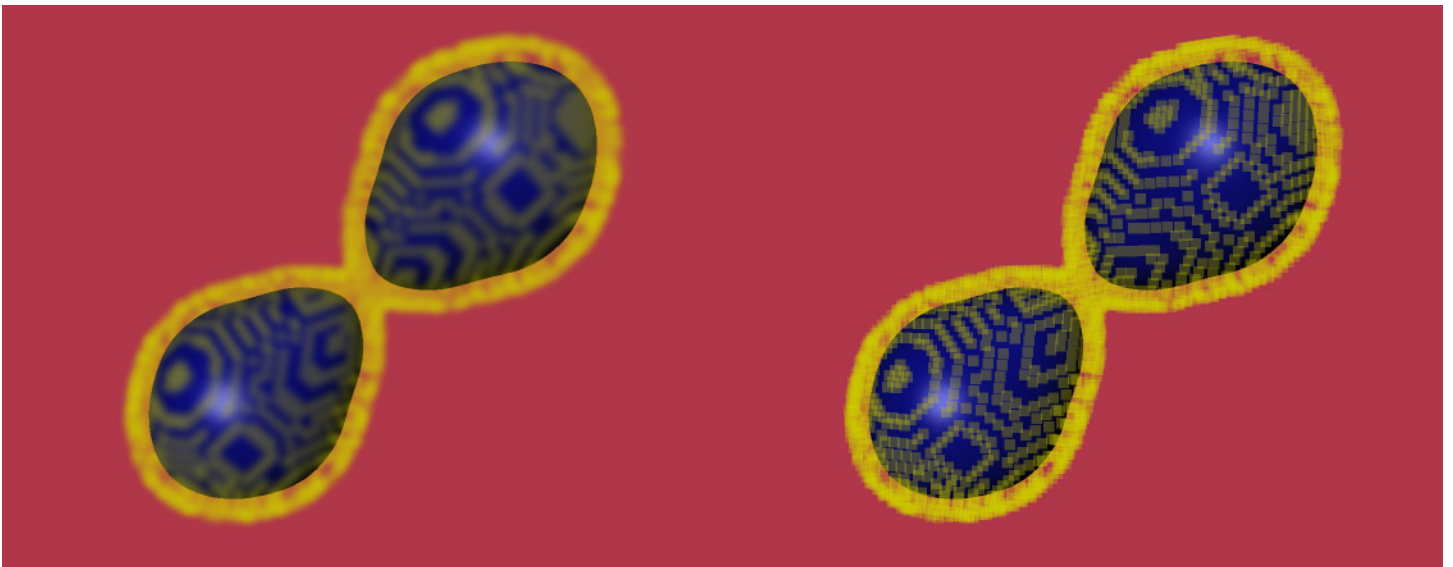


Figure 7. Interval set of proton density using Texture Splats

Figure 8. Interval set of proton density using Point Splats

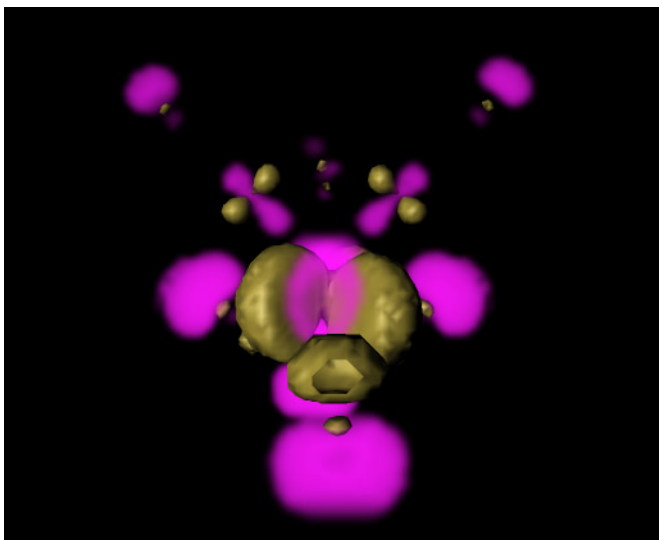


Figure 9. Interval set of HIPIP using Texture Splats

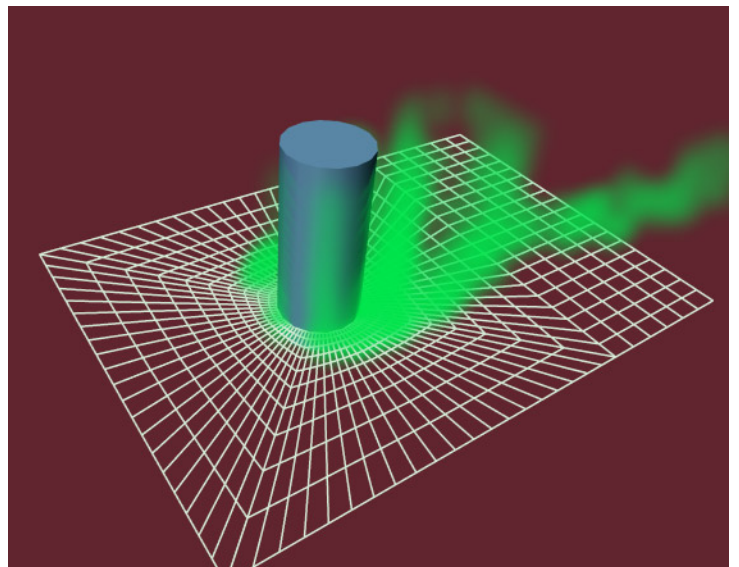


Figure 10. Animated slices through a finite-element mesh.