Volume Rendering of 3D Scalar and Vector Fields at LLNL

Roger Crawfis, Nelson Max, Barry Becker, Brian Cabral

Lawrence Livermore National Laboratory
P.O. Box 808
Livermore, CA 94551

Abstract

Simulation of complex 3-dimensional phenomena generate data sets which are hard to comprehend using conventional 2-dimensionally oriented visualization tools. One way to overcome this limitation is to employ various volume visualization techniques. While early volume visualization techniques worked well on simple scalar volumes they failed to exploit frame buffer hardware capabilities and handle higher order data such as vector fields. Work at Lawrence Livermore National Laboratory has centered on developing new techniques and extending existing techniques. This paper describes the various algorithms developed for volume rendering, and presents new methods of examining vector fields in a volumetric fashion.

Introduction

Computer simulations dealing with the atmosphere, the ocean, hydrodynamics, electro-magnetics, structural response, and environmental clean up, generate very large and complex data sets consisting of not only several scalar variables, but also vector and tensor quantities. Understanding an individual scalar field from a three-dimensional simulation is often difficult with current visualization tools. Understanding 3D vector fields and complex relationships is even more arduous. In order to understand the tens of gigabytes of data being generated from today's supercomputers, better visualization techniques and tools are needed. Better visualization tools are even more urgent as we progress to massively parallel architectures and larger more complex problems.

Volume rendering allows us to examine a substantial amount of a single 3D scalar field at a single time step. Research into visualizing 3D scalar fields has progressed over the past few years from iso-contour surfaces, to direct volume rendering of regularly gridded data, and then irregularly structured data, but all methods operate on a single scalar field. A minor amount of effort has been placed on understanding the interactions between several scalar fields. Even less effort has been directed towards understanding vector or tensor fields. Correcting these deficiencies has been the thrust of our research. This paper describes the various algorithms developed for volume rendering, and presents new methods of examining vector fields in a volumetric fashion.

Volume Rendering

The rendering of three-dimensional scalar fields has received much attention over the past several years. There are three common ways to visualize a scalar function in a volume (1) interactively move a color coded 2D slice through the volume [Nielson90], (2) draw contour surfaces [Lorensen87], or (3) integrate a continuos volume density along viewing rays [Max90]. The second method is a generalization of plotting contour lines in 2D. The other methods generalize the color coding of a 2D scalar field, although the 3D color field can only be viewed in either 2D slices or projections. The first two methods are generally termed *indirect* volume rendering, while the third method is termed *direct* volume rendering, or simply volume rendering. These terms imply the basic architecture of the techniques: indirect volume rendering converts the 3D field into geometric surface primitives which are then rendered to form an image, while direct volume rendering converts the field directly into the image. Henceforth, we will use the term volume rendering to refer strictly to direct volume rendering.

There has been much research on volume rendering algorithms over the past few years, leading to several competing algorithms. Four common approaches to volume rendering are (1) ray tracing, (2) analytical scan conversion, (3) hardware compositing, and (4) splatting. Various advantages/disadvantages exist with each of these algorithms. Many flavors of each of these algorithms also exist. We have implemented a version of each of these and extended them to

examine vector fields rather than or in addition to a single scalar field.

Before describing each of these algorithms and our extensions to them, some background information is useful. The basic premise of volume rendering is to simulate the absorption and scattering of light passing through a volume. There have been several computer graphics papers on the scattering, transmission, and shadowing of light propagating through clouds of particles. Kajiya and Von Herzen [Kajiya84], Rushmeier and Torrence [Rushmeier87], Blinn [Blinn82], and Max [Max86a] [Max86b] all suggest methods of correctly accounting for the shadowing, but the computation required is prohibitive. Instead, we chose to ignore the shadowing entirely, and only occlude the light on the way to the viewer, after a single scattering event. This leads to the following very simple illumination calculation. We model light as ambient illumination shining equally from all directions, and not shadowed along its path to any scattering particle. Under these assumptions, the result is the same as modeling glowing particles, as in the *density emitter* model of Sabella [Sabella88]. Sabella assumes that the volume density $\rho(x,y,z)$ glows with an energy $C\rho$ per unit length, and absorbs with an optical density of $\tau\rho$ per unit length, where C and τ are constants for any fixed material.

Consider a ray, R(t) = (x(t),y(t),z(t)), leaving the eye and passing through the volume. Then the total optical density of the cloud along the ray from the eye to a point R(t) is $\int_0^t \tau\rho(u)du$, so light starting from P(t) is attenuated by the transparency factor $\exp(-\int_0^t \tau\rho(u)du)$ [Blinn82]. The length dt of the ray glows with energy $C\rho(t)dt$, so the total glow energy reaching the eye is

$$I = \int_0^1 C\rho(t)e^{-\int_0^t \tau\rho(u)du}\,dt. \qquad (1)$$

These integrals can be calculated analytically when C, τ, and ρ are constant or linearly interpolated within a volume cell [Max90], [Williams92].

Current graphics hardware does not offer assistance in calculate this integral directly. It does however offer assistance in compositing or *alpha-blending* two colors together and offers support for texture mapping (placing a decal or picture on an object). We strive to take full advantage of these capabilities where appropriate.

Ray Tracing Volume Densities

Ray tracing is one method for solving the single scattering illumination integral given above. This technique requires casting rays from a viewpoint through a volume for each pixel in the output image. Various sampling and illumination techniques are employed to approximate and extend equation (1) (see [Upson88], [Levoy89], or [Danskin92]). In our implementation we use two types of sampling along a ray: one in which the samples are evenly spaced and one where the samples occur only on the faces of the rectilinear data cells. In this latter cases the samples are unevenly spaced. This approach to volume rendering makes it easy to use very sophisticated lighting or shading models at an increased computational cost.

One approach to visualizing volumetric vector fields is to map the vector field onto a scalar field and then volume render the result. We [Cabral93] have developed an algorithm, known as Line Integral Convolution (LIC), which performs this mapping. The technique can be thought of as a data operator which filters a 3-dimensional input image as a function of a 3-dimensional vector field, producing an output 3-dimensional voxel image.

For each each cell centered voxel containing a vector, or *vectel*, a local parametrically defined streamline is computed using a variable step Euler's method. (Other streamline computation techniques could be used, such as Runge-Kutta. We chose the technique described in [Cabral93] for reasons of efficiency and simplicity). This parametric curve, $p(x,y,z,s)$, is used to cut a 1-dimensional slice out of the input image, $F(x,y,z)$. The 1-dimensional slice can be thought of as a signal in the curve's parameter, s. This signal then filtered using a standard 1-dimensional convolution. The general form of this operator is given by the following integral

equation:

$$F'(x,y,z) = \frac{\int_{-L}^{L} F(p(x,y,z,s))k(s)ds}{\int_{-L}^{L} k(s)ds} \qquad (2)$$

where *k(s)* is the convolution kernel and 2*L is the length of the streamline. The denominator is a normalization term designed to keep the brightness of the output voxels in the range of the input voxels. The LIC integral, eq. (2), is performed once for every vectel, producing a scalar value for the corresponding output voxel.

If an isotropic, uncorrelated input image is used, such as white noise, the LIC operator can be thought of as correlating this noise image along local vector streamlines. This produces an image which on average is fairly homogeneous. Volume rendering this image without first weighting either the input or output image with another scalar field would result in only visualizing the outer surface of the volume. Figure 1 is a volumetric rendering of an electrostatic vector field consisting of two attractive charges. Here the input image is white noise weighted by the magnitude of the vector field. LIC is run on this input image and the resulting scalar volume is then rendered using the ray tracing approach described above.

Coherent Scan Conversion of Volume Densities

Max, Hanrahan and Crawfis [Max90] describe a technique for scan converting the back and front faces of an individual volume cell, and integrating the density of the rays passing through the front and back faces. The algorithm is exact for density fields which vary linearly. They also describe a general sorting algorithm for Delaunay triangulation, and discuss the intermixing of contour surfaces and the density clouds.

We have made several extensions to this algorithm, not documented in [Max90]. The contour surface can be color coded using a different scalar field than that used for the contouring. A list of data fields is processed for each vertex. One data field can specify the contouring, another the color of the density volume, and still another the opacity of the density volume. More general mappings can be used that take into account several data fields, the vertex positions, and information indicating which side of any contouring surfaces the vertex lies on.

We have extended this algorithm to allow for the contour surfaces to be covered with a three-dimensional texture. In particular, we applied a three-dimensional cloud texture to the contour surfaces of the percent cloudiness field resulting from a global climate simulation. We used the wind velocities to advect the texture coordinates given to this 3D texture, allowing the clouds to move with the wind where appropriate, and providing a sense of the wind blowing over the clouds in other areas. Figure 2 is a still from an HDTV animation done using this technique. The volumetric rendering was turned off for this image and an analytically computed atmospheric haze was added.

Hardware Projection and Compositing of Volume Densities

Shirley and Tuchman [Shirley90] describe an algorithm for dividing tetrahedral volume cells into up to four simpler tetrahedra. Each simple tetrahedron projects to a screen triangle with two vertices projecting to the same screen point A, and two faces perpendicular to the screen. Thus the thickness varies linearly from a maximum at A to zero at the opposite edges. Shirley and Tuchman compute the color and transparency at each vertex, and use the shading hardware inside the graphics rendering pipeline to interpolate these values at each pixel. Then they use the "alpha compositing" hardware to hide the background image behind the tetrahedron, according to the computed transparency, before adding in the new color.

Since these computations can all be done by the high speed hardware pipelines on modern workstations, they are quite fast. However, as we saw above, the correct transparency is an exponential function of the optical density, which is not linear. Thus, the density should be interpolated linearly across the triangle and then the exponential should be taken at every pixel. If instead, the transparency is computed only at the vertices, and then linearly interpolated across the triangles, visible artifacts will result.

It is not possible to compute an exponential per pixel inside current rendering pipelines, but an equivalent result can be achieved using texture mapping hardware. A 1D texture table is used,

which is addressed by the optical density, and returns the transparency. The texture coordinate is then specified as the optical density at each vertex, and will be interpolated across the triangle by the shading hardware, before being used as a texture address. This gives an accurate transparency for every pixel, and eliminates the artifacts.

We use this algorithm for a new vector visualization technique which we call flow volumes [Max93b]. We have extended the concept of stream lines or flow ribbons to their volume equivalent. A seed polygon is placed into the flow field under user control. This polygon acts as a smoke generator. As the vector field passes through the polygon, smoke is propagated forward, sweeping out a volume (Figure 5) which is subdivided into tetrahedra. Compression and expansion of the volume due to the flow can be taken into account by adjusting the opacity based on the tetrahedron's volume. As the flow volume expands, we employ an adaptive mesh refinement technique to ensure the curvature of the resulting volume is accurate. The complex topology of the flow volume would require a general sorting method to yield a valid back-to-front sort. However, for this application, we can require that the smoke or dye be a constant color throughout the volume. It can then be shown [Max93b] that the resulting integration of the volume density is independent of the order the volume cells are processed. Thus, no sorting is required. Since we are only rendering the smoke, and not an entire volume, and using the hardware for the rendering, we have achieved real-time interaction. We have also added additional features that allow the user to watch moving puffs of smoke, control the time propagation of the smoke, and combine opaque geometry with the smoke. Figure 4 shows an aerogel simulation, where the tiny cubes are the zones containing the aerogel, and the blue smoke represents a flow through the aerogel.

Hardware Splatting of Volume Densities

Westover [Westover89] used an idea called splatting, where each voxel is treated as a single sampling point and a continuos 3D signal is reconstructed. The contribution of each voxel point to the volume density is then determined by its reconstruction kernel, and these are composited into the image independently in back-to-front order. Westover [Westover90] later used an accumulation buffer to reconstruct an entire sheet's contribution before it is composited into the image. Since the individual splats overlap, this avoids problems in the sorting, however, a much more noticeable change in the volume density occurs when the arrangement into sheets must change.

Laur and Hanrahan [Laur91] extended the splatting technique to handle hierarchical representations of the sampled data field. They also approximated the gaussian used in the reconstruction by a small polygonal mesh and utilized the graphics hardware to render the splat. This allowed for a quick coarse representation of the data that evolved into a more accurate representation adaptively. The Explorer product from SGI uses a variant of this technique without the adaptive refinement.

In reconstructing the 3D signal, a gaussian function is usually used. An unattractive property of this is its infinite extent. Every splat theoretically contributes to the entire volume density. Some finite extent is usually chosen and the splat is either abruptly cut-off or forced to zero. Max [Max92] uses an optimal quadratic function with a limited extent for the reconstruction of 2D signals. We [Crawfis93] have extended this for a cubic function for the reconstruction of 3D signals, which is optimal for all viewing angles. Using this function as a hardware texture map on a simple square splat gives a more accurate rendering of the volume density, and for large problems consisting of many small splats is actually faster than using the polygonal mesh of [Laur91]. We have added this to the Explorer system.

We [Crawfis92] developed a methodology for integrating a vector representation into the scalar splat. This algorithm however uses a sampling and reconstruction approach (which we called a filter) and does not make use of the graphics hardware. Figure 3. illustrates this technique for the wind field of a global climate simulation. We [Crawfis93] have now merged this concept with volume rendering via splatting by extending the texture splats above to include an anisotropic representation of the vector field. The splat is not only oriented perpendicular to the viewing direction, but also rotated to align the texture with the projected vector direction. A table of textures

with different vector lengths is used to foreshorten the vector as its direction becomes parallel to the viewing direction. By using the appropriate controls of the texture mapping hardware, the volume density can be represented using one color scheme, and the vector field represented using another. Figure 6 shows the percent cloudiness field and the wind velocities for the global climate simulation. By cycling through a sequence of textures with changing displacements, we can make the textures move in the flow specified by the vector field.

Sorting

The scan conversion, hardware projection, and splatting methods of volume rendering all require a back to front sort of the volume cells. For rectangular lattices, the location of the viewpoint within the data volume (or of its projection onto a face or edge of the data volume) can easily be used to specify a sorting order. This sorting method extends recursively to octree or other rectilinear adaptive mesh refinements. For a more general mesh of convex data without holes, sorting can be done using a topological sort [Knuth73] [Max90] on a directed graph representing cell adjacency (with the direction of an edge specifying which of the two adjunct cells is in front). This sort may detect cycles, but it can be proved that cycles do not occur for Delaunay triangulations. [Edelsbrunner89], [Max90], or for the particular geometries which occur in our climate simulations [Max93a]. We are also working on a version of the Newell, Newell, and Sancha sort [Newell72] applied to volume cells instead of to polygons. This algorithm will split offending cells when cycles are detected. It also does not require the adjacency information, which may not be available in certain situations (for example, finite element simulations with sliding interfaces), and can deal with non convex volumes with holes. Williams [Williams91] also generalizes the directed graph method to non-convex data volumes, but his method is not guaranteed to be correct.

Conclusions

We have explored the use of volume rendering for scientific visualization and extended the algorithms for representing vector field data as well as scalar field data. The algorithms cover a wide gamut of techniques, from the very interactive flow volume generator for studying specific areas, to the ray-traced vector volumes and vector splatting techniques for representing global views of the vector field. These algorithms are a necessary step towards better visualization techniques for large data sets produced from complex three-dimensional simulations on supercomputers and massively parallel machines.

This paper has given a broad overview of our work. More details can be found in the references: [Max90], [Crawfis91], [Crawfis92], [Max92], [Cabral93], [Crawfis93], [Max93a], and [Max93b]. These papers also give other new material on the usage of volume rendering and our extensions to vector field rendering.

Future Work

Most of the extensions mentioned here are in their early development. Much work is still needed in applying them to various application data and making refinements. In particular, the proper choice of textures for the textured isocontour and the vector splatting techniques is an open area of research. Better subdivision and splitting algorithms are needed for the flow volume algorithm. We have also developed the basic framework for multivariate representations, but have only tried it out for very simple mappings. More complex mappings (presumably application specific) need to be studied, and a more efficient and flexible framework put into place.

Acknowledgments

The authors are indebted to the many people who provided the data used in the visualization in this paper. Tony Ladd and Elaine Chandler provided the aerogel data. 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 authors would like to thank Chuck Grant 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, with specific support from an internal (LDRD) research grant.

References

[Blinn82] Blinn, James, Light Reflection Functions for Simulation of Clouds and Dusty Surfaces.

Computer Graphics Vol. 16 No. 3 (Siggraph '82) pp. 21-29.

[Cabral93] Cabral, Brian and Casey Leedom, *Imaging Vector Fields Using Line Integral Convolution,*

[Crawfis91] Crawfis, Roger and Michael Allison. *A Scientific Visualization Synthesizer*. In *Proceedings Visualization '91*. Gregory Nielson and Larry Rosenblum eds., IEEE Los Alamitos, CA, pp. 262–267.

[Crawfis92] Crawfis, Roger and Nelson Max, *Direct Volume Visualization of Three-Dimensional Vector Fields.* Proceedings of the 1992 Workshop on Volume Visualization (October 1992), ACM SIGGRAPH New York. pp. 55-60.

[Crawfis93] Crawfis, Roger and Nelson Max, *Texture Splats for 3D Vector and Scalar Field Visualization.* In Proceedings Visualization '93 (October 1993), IEEE Los Alamitos, CA, pp. 261-266

[Danskin92] Danskin, John and Pat Hanrahan, *Fast Algorithms for Volume Ray Tracing.* Proceedings of the 1992 Workshop on Volume Visualization (October 1992), ACM SIGGRAPH New York. pp. 91-98.

[Edelsbrunner89]Edelsbrunner, Herbert *An Acyclicity Theorem in Cell Complexes in d Dimensions.* Proceedings of the ACM Symposium on Computational Geometry (1989) pp. 145-151.

[Kajiya84] Kajiya, James T. and Brain P. Von Herzen,. *Ray Tracing Volume Densities.*. Computer Graphics Vol. 18 No. 3 (SIGGRAPH '84) pp. 165174.

[Knuth73] Knuth, Donald E. *The Art of Computer Programming Volume 1: Fundamental Algorithms*. 2nd Edition. Addison-Wesley Reading, MA (1973).

[Laur91] Laur, David and Pat Hanrahan, *Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering.* Computer Graphics Vol. 25 No. 4 (July 1991, SIGGRAPH '91) pp. 285–288.

[Levoy89] Levoy, Marc. *Design for a Real-Time High-Quality Volume Rendering Workstation.* In *Proceedings of the Chapel Hill Workshop on Volume Visualization,* pp. 85-92.

[Loreensen87] Lorensen, Bill and

[Max86a] Max, Nelson, *Light Diffusion through Clouds and Haze.* Computer Vision, Graphics, and Image Processing Vol. 33 (March 1986) pp. 280-292.

[Max86b] Max, Nelson, *Atmospheric Illumination and Shadows.* Computer Graphics Vol. 20 No. 4 (Siggraph '86) pp. 117-124.

[Max90] Max, Nelson, Pat Hanrahan, and Roger Crawfis, *Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions.* Computer Graphics Vol. 24 No. 5 (November 1990, Special issue on San Diego Workshop on Volume Visualization) pp. 27–33.

[Max92] Max, Nelson, Roger Crawfis, and Dean Williams, *Visualizing Wind Velocities by Advecting Cloud Textures.* Proceedings Visualization '92, IEEE Los Alamitos, CA. pp. 179-184.

[Max93a] Max, Nelson, Barry Becker, and Roger Crawfis, *Volume and Vector Field Visualization for Climate Modelling.* IEEE Computer Graphics and Applications, (July 1993) pp. 34-40.

[Max93b] Max, Nelson, Barry Becker, and Roger Crawfis, *Flow Volumes for Interactive Vector Field Visualization.* In Proceedings Visualization '93 (October 1993), IEEE Los Alamitos, CA, pp. 19-24.

[Newell72] Newell, M. E., R. G. Newell, and T. L. Sancha, *A solution to the Hidden Surface Problem*. Proceedings of the ACM National Conference 1972, pp. 443-450.

[Nielson90]

[Rushmeier87] Rushmeier, Holly E. and Kenneth E. Torrance *The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium* Computer Graphics Vol. 21 No. 4 (SIGGRAPH '87) pp. 293302.

[Sabella88] Sabella, Paolo. *A Rendering Algorithm for Visualizing 3D Scalar Fields.* Computer Graphics Vol. 22 No. 4 (July 1988, SIGGRAPH '88) pp. 51–58.

[Shirley90] Shirley, Peter, and Allan Tuchman, *A Polygonal Approximation to Direct Scalar Volume Rendering.* Computer Graphics Vol. 24 No. 5 (November 1990, Special issue on San Diego Workshop on Volume Visualization) pp. 63–70.

[Upson88] Upson, Craig and Michael Keeler, *VBUFFER: Visibile Volume Rendering,* Computer Graphics Vol. 22 No. 4 (July 1988, SIGGRAPH '88) pp. 59-64.

[Westover89] Westover, Lee. *Interactive Volume Rendering.* In Proceedings of the Chapel Hill Workshop on Volume Visualization, May 1989, pp. 9–16.

[Westover90] Westover, Lee. *Footprint Evaluation for Volume Rendering.* Computer Graphics Vol. 24 No. 4 (July 1990, SIGGRAPH '90) pp. 367–376.

[Williams91] Williams, Peter L., *Visibility Ordering Meshed Polyhedra* ACM Transactions on Graphics Vol. 11 No. 2 (April 1992) pp. 103-126.

[Williams92] Williams, Peter, and Nelson Max, *A Volume Density Optical Model.* Proceedings of the 1992 Workshop on Volume Visualization, ACM, New York, pp. 61-68.

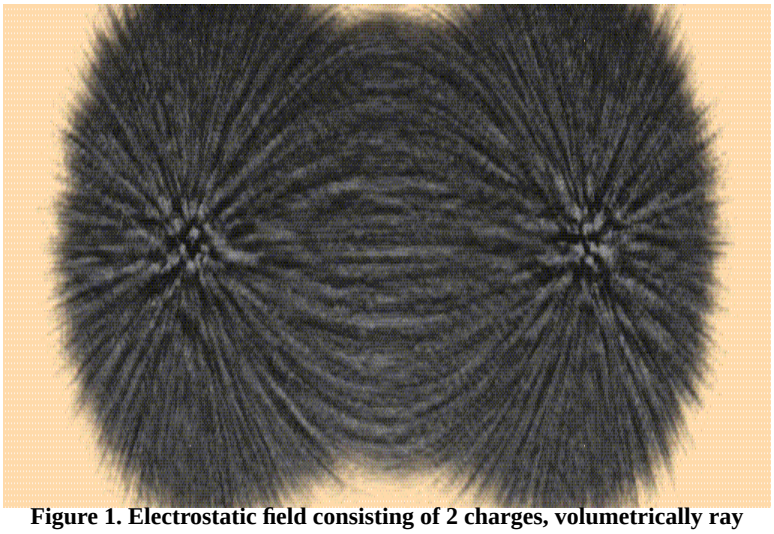

Figure 1. Electrostatic field consisting of 2 charges, volumetrically ray traced using the vector to scalar converter.

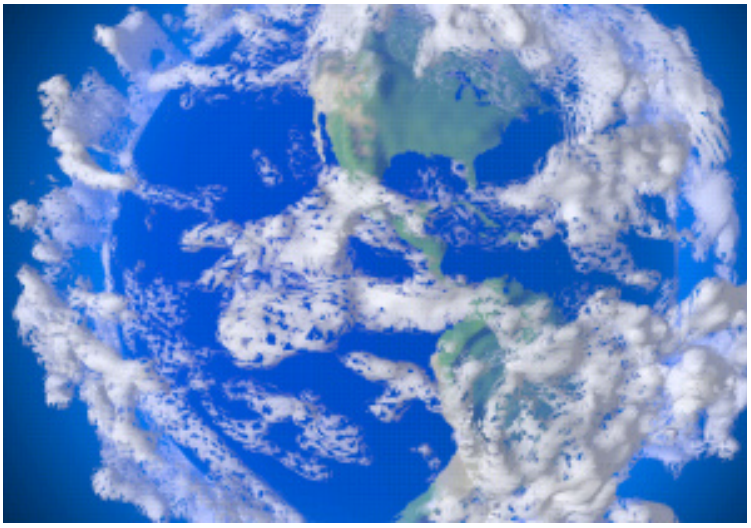

Figure 2. The percent cloudiness field rendered using a 3D texture advected by the wind field.

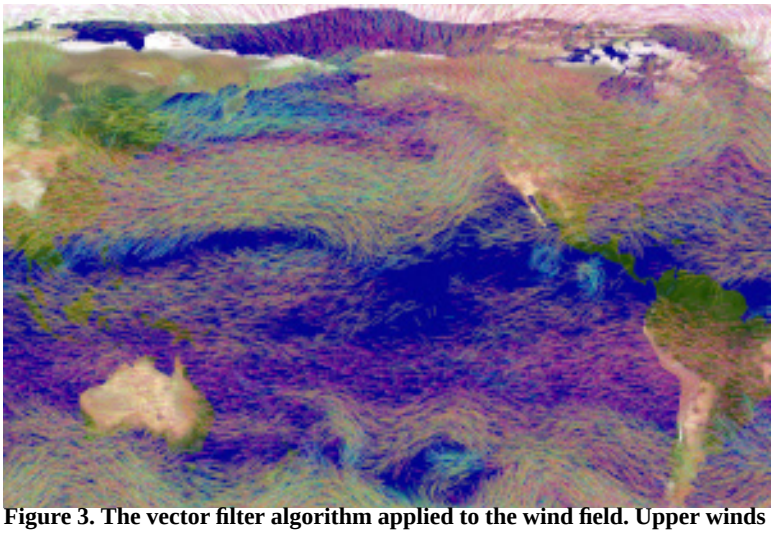

Figure 3. The vector filter algorithm applied to the wind field. Upper winds are in red, surface winds are in cyan.

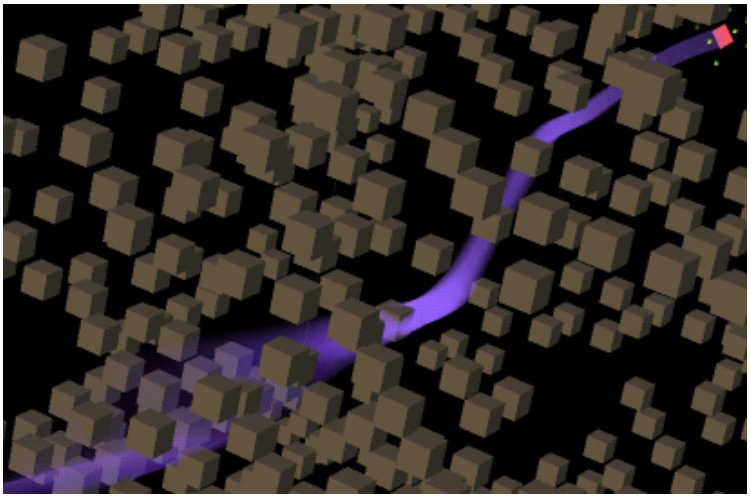

Figure 4. A flow volume weaving through an aerogel substance.

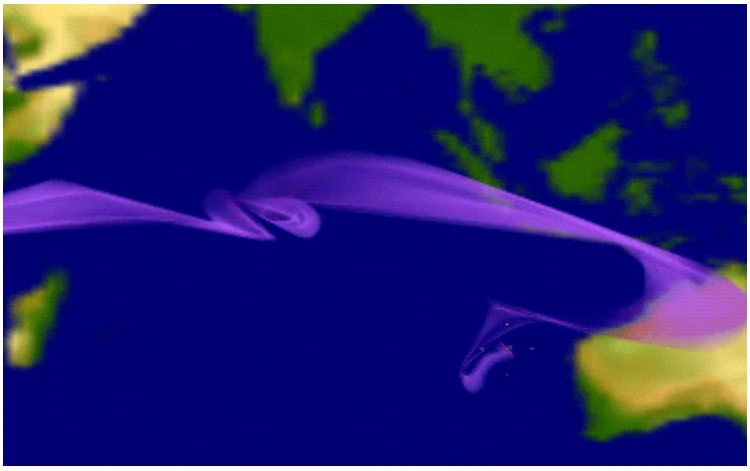

Figure 5. The Flow Volume Algorithm applied to the global climate data set.

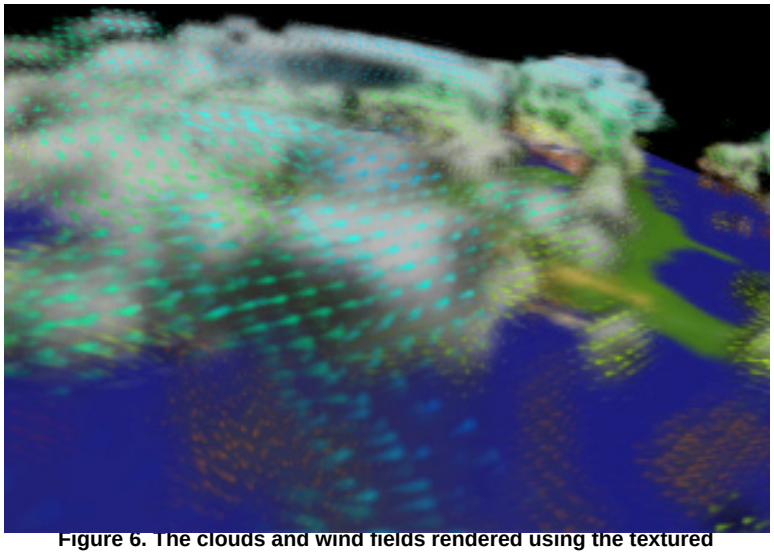

Figure 6. The clouds and wind fields rendered using the textured splats algorithm.