Direct Volume Visualization of Three-Dimensional Vector Fields

Roger Crawfis
Nelson Max

Lawrence Livermore National Laboratory
PO. Box 808 / L–301
Livermore, CA 94551
(crawfis@llnl.gov)
(max2@llnl.gov)

Abstract

Current techniques for direct volume visualization offer only the ability to examine scalar fields. However most scientific explorations require the examination of vector and possibly tensor fields as well as numerous scalar fields. This paper describes an algorithm to directly render three-dimensional scalar ***and*** *vector fields. The algorithm uses a combination of sampling and splatting techniques, that are extended to integrate the display of vector field data within the image.*

Additional Keywords: vector field, flow field, volume rendering, vector filter, compositing, scalar field, climate modeling.

Introduction

The rendering of three-dimensional scalar fields has received much attention over the past several years. These 3D scalar fields can be represented using either isocontour surface reconstruction algorithms, or as semi-transparent density clouds. With isocontour surfaces, intermediate geometry is produced and processed using the normal geometric pipelines developed during the last few decades. Thus, additional geometric objects such as axes, vectors or additional isocontours (from possibly different scalar fields) can be easily added to the display of the isocontour.

Unfortunately, this is not the case for volume density clouds. Shirley and Neeman [Shirley89] and Levoy [Levoy89] discuss the integration of separate geometric objects using raytracing. The sorting required at each sample point makes this algorithm infeasible for a large number of geometric objects, such as that produced by the display of many tiny vectors.

Max, Hanrahan and Crawfis [Max90] demonstrate how to incorporate geometric surfaces into their back-to-front compositing of volume polyhedra. This was limited to isocontour surfaces, and also required splitting the polyhedron up into several pieces and shipping each individual piece to the volume renderer. Lately, projective techniques have been developed that use a geometric description of the cloud density within each voxel [Westover90], [Wilhelms91], [Laur91]. While these techniques are geometrically based, they require passing the polygons to the geometry pipeline in a back-to-front order.

Research into the display of three-dimensional flow has also been explored over the past few years. Various algorithms to represent the flow via ribbons have been developed. Helman and Hesselink [Helman91] and Globus et. al. [Globus91] have developed algorithms to display critical points within the flow field. These algorithms and the standard vector or hedgehog plots have no direct way of being combined with the direct volume visualization methods recently developed.

Particle systems [Reeves85] can be used to represent both scalar ([Max90], [Sabella88]) and vector fields ([van Wijk91], [van Wijk90]). Vector fields require an advection of the particles for each time step, and usually involve creating and deleting particles as time progresses. Unfortunately, these algorithms are quite computationally intensive, and the number of particles required for this purpose would be prohibitive. Kajiya [Kajiya89] also made allusions to representing vector fields using algorithms developed for the display of hairy surfaces. This technique is limited to the display of the vector flow upon a surface and is very computationally expensive, requiring several hours of CPU time per image. Spot noise [van Wijk91] offers an interesting technique for visualizing flow fields, but is again limited to the flow over a surface. and is also computationally expensive.

Our goal was to render the relationship between turbulent flow fields and scalar density fields throughout a three-dimensional volume. This was driven by a requirement to visualize the cause and effect relationship of clouds and winds within global climate models [Potter91]. Global climate modeling produces a time history of data, each time step of which needs to be rendered. We have investigated the use of high frequency textures to represent vector fields in two-dimensions [Crawfis91]. Here, an anisotropic texture is derived from the vector field. Time dynamics are then created by simply regenerating an anisotropic texture at each time point. Since we recognize frequency, but not phase, in patterns and textures, a smooth flow is created that provides the illusion of motion in an animation, without requiring any advection. A method that does use advection on the same climate data is described in [Max92]. We have extended our research of two-dimensional vector filters into three-dimensions, and incorporated the integration of a scalar field with the compositing of the vector field to accomplish our goals.

Our technique for representing vector fields is to create a very fine-grained texture representation of the flow. Individual vectors, insignificant individually, combine to form a useful picture of the overall flow of the field. We have developed a filter which can be used to sweep through a volume image in back-to-front order. The kernel of this filter can be used to represent both scalar and vector quantities for two- and three-dimensional data sets. The basic algorithm to render a two-dimensional vector field passes a vector kernel filter across the resultant image. The kernel deposits an anti-aliased line over the width of its domain. Each individual line is composited in using the *OR* operator, and its orientation is based on a sampling of the vector field. By carefully controlling the movement of the filter, highly dynamic flow fields can be represented (Figure 5).

Filter Movement

A key criterion for good texture generation of the vector field is to avoid patterns caused by the regular movement of the filter, or the regular spacing of the data. Three options to overcome these patterns are available. Within the filter kernel, the center point (**P** in Figure 2) through which the vector passes is randomly chosen (Figure 3a). The major reduction in patterns comes from controlling the movement of the entire filter. The filter is moved with random jitters in its increment to prevent the regular spacing apparent from the clipped edges of the vectors (Figure 3b) Finally, the filter is moved in increments smaller than its width (Figure 3c). This allows vectors to overlap, and blurs the extent of each individual vector. While the differences between these three images may not be substantial, when we animate the images, very different results appear. What we are after is the illusion that the particles or the texture as a whole is moving, not individual flags waving in the wind. Whether all of these jitterings are necessary has yet to be determined, however none of them require additional resources of any significance. These jitterings were developed for two-dimensional filters and side or top views of three-dimensional data sets. Oblique views in three-dimensions will naturally break up regular patterns to some extent.

Vector Kernels

Vectors are represented on the image as line segments with varying color and opacity. As the filter moves along the output image, a *vector kernel* performs three tasks: determine the projection of the vector, calculate the color and opacities of the vector line segment and the scalar function, and composite this information into the image. The vector is projected onto the viewing plane by taking the dot product of the vector with two basis vectors defining the viewing plane:

$$ux = \vec{\mathbf{V}} \cdot \vec{\mathbf{i}}$$

$$uy = \vec{\mathbf{V}} \cdot \vec{\mathbf{j}}$$

where the projected vector, $\vec{u} = (ux \quad uy)^{\mathbf{T}}$

This projected vector, $\vec{u}$, is then normalized for use in future operations.

Once we have the projection of the vector onto the screen, we then need to determine for each pixel what fraction of the vector lies within the pixel and what the various attributes of the vector are at that pixel. Assuming circular pixels, the area of overlap with a thick line segment can be estimated by taking the absolute value of the dot product of the vector, $\vec{n}_p$, perpendicular to $\vec{u}$ with the vector from the center point to the current pixel (Figure 2). This gives us the perpendicular distance from the axis of the vector to the pixel. The function:

$$f(r) = \begin{cases} 1.0 & r \le r_1 \\ ar + b & r_1 \le r \le r_2 \\ 0.0 & r \ge r_2 \end{cases}$$

can then be used to produce smooth anti-aliased lines. The values of r1 and r2 control the thickness of line segments, and are specified by the user. These anti-aliased lines work better and are computationally easier than using cylinders.

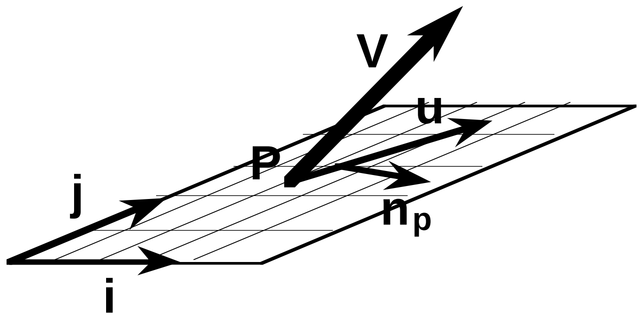

Figure 2. Determining Pixel values

The area is used as an opacity and color scaling factor in compositing the vector into the image. Several controls over the representation of the vectors are available. Depending on the kernel, an arbitrary color mapping scheme is offered. Current kernels will map the either the world z-coordinates or the screen z-coordinates (z-height) to a color in a user specified color table. The z-height can include both the relative position in the data set and the height increase of the vector across the filter. By heavily weighting this latter term, color can be mapped to show the vertical velocity component. Other color mapping schemes such as those proposed by Van Gelder and Wilhelms [VanGelder92], could easily be incorporated. This color is used as the base color or hue. By desaturating one end of the vector, we can add an indication of the signed direction of the vector (i.e., a vector head). Here, if we simply take the dot product of the projected vector and the vector to the pixel center, we will get a measure of where we are along the axis of the vector. Since we are only concerned with the pixels along the vector axis at this point, we can use this measure directly.

Finally, we adjust the vector's intensity by its magnitude. A depth cue can also be applied by adjusting the intensity based on the linear distance from the view point.

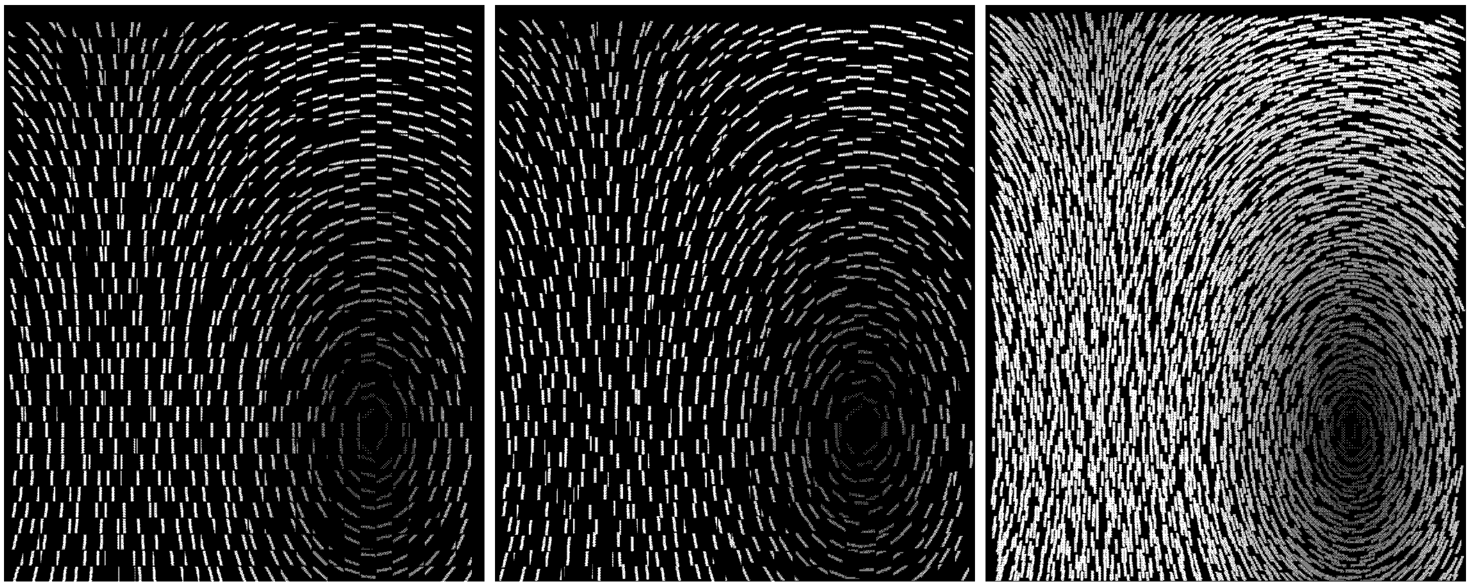
Figure 3. Vector Kernel Movement Effects: a) Jittering of the center point, b) Jittering of the stride length, c) Overlapping strides.

Scalar Rendering

For overlapping filters, a splatting-like algorithm works well. The ideal reconstruction filter described by Max [Max91] is used as the basis for the splat. This filter:

$$g(r) = \begin{cases} 1 - \dfrac{r^2}{\mathrm{st}} & 0 \le r \le \mathrm{s} \\ \dfrac{(\mathrm{t} - r)^2}{\mathrm{t}(\mathrm{t} - \mathrm{s})} & \mathrm{s} \le r \le \mathrm{t} \\ 0.0 & r \ge \mathrm{t} \end{cases}$$

with s = 0.48 and t = 1.37 for a filter stride of one, has a finite span, allowing the size of the kernel to be arbitrarily large. This allows long vectors, while limiting the effect of the splat to the stride taken in the filter. This does present the problem that the vector drawn with the splat is only affected by that splat and not neighboring splats that may overlap the vector. Overcoming this would require complicated neighborhood tests, and the handling of multiple vector segments within the kernel. Since these discrepancies in the renderer are not noticeable for the test cases we have run, we choose to ignore them. This implies that a perfectly acceptable solution would be to simply *splat* in a vector, and then splat in the volume over it for each kernel instantiation. A more accurate solution is described in the next section.

The addition of scalar splatting increases the computational time of the vector kernel substantially. The reason for this is twofold:

1) The pixels outside the projected vector must now be calculated and composited in.

2) Kernel calculations were skipped if the vector length was less than some user specified tolerance. These must now be drawn if the scalar field contributes to the image (i.e., the scalar field is greater than a certain threshold).

Compositing

Once we have the contribution due to the vector field and the contribution due to the scalar field at each sampled voxel, we can calculate the total contribution to each pixel. Consider a ray, of unit length, from the eye passing through a polyhedron within which we wish to render a vector (Figure 4). The segment of this ray passing through the polyhedron is broken up into three parts: that segment in front of the vector, the segment passing through the vector, and the segment behind the vector. Let dz represent the length of the front segment. If the vector has a thickness, dv, then the segments have lengths dz, dv, and (1-dz-dv). If we then assume a homogenous opacity and color, the intensity can be calculated using the equation:

$$I = \int_0^1 \Omega(t) e^{-\int_0^t \Omega(u)du} dt$$

$$= \int_0^{dz} \rho(t) e^{-\int_0^t \rho(u)du} dt$$

$$+ e^{-\int_0^{dz} \rho(u)du} \int_{dz}^{dz+dv} \nu(t) e^{-\int_{dz}^t \nu(u)du} dt$$

$$+ e^{-\int_0^{dz} \rho(u)du} e^{-\int_{dz}^{dz+dv} \nu(u)du} \int_{dz+dv}^1 \rho(t) e^{-\int_{dz+dv}^t \rho(u)du} dt$$

where $\rho(x)$ and $\upsilon(x)$ represent the scalar density and vector density distributions along the ray, and $\Omega(x)$ the total density distribution.

If we assume an infinitesimal thickness in the vector, and give it a fixed opacity, α_V , and color, I_V, then the equation simplifies to:

$$
\begin{aligned}
I = & \int_0^{dz} \rho(t) e^{-\int_0^t \rho(u)du} dt \\
& + I_V e^{-\int_0^{dz} \rho(u)du} \\
& + (1-\alpha_V) e^{-\int_0^{dz} \rho(u)du} \int_{dz}^{1} \rho(t) e^{-\int_{dz}^{t} \rho(u)du} dt
\end{aligned}
$$

The value dz can be calculated for each pixel ray from the plane consisting of the transformed vector and one of the basis vectors defining the viewing plane. By using the analytical integration proposed by Max, Hanrahan and Crawfis [Max90], this calculation requires only two exponential evaluations, or one additional exponential over the straight volume rendering.

The color times depth approximation, **C*D**, proposed by Wilhelms and Van Gelder [Wilhelms91], can be used to further simplify this equation. Here, three colors and opacities are computed for the vector, in front of the vector, and in back of the vector and composited together. If I_S and α_S are the color and opacity per unit length for the scalar field, the equivalent equation for the simplified **C*D** calculation is:

$$I = I_S dz + (1 - dz\alpha_S)I_V + (1 - dz\alpha_S)(1 - \alpha_V)I_S(1 - dz)$$

While this follows logically, it does not produce the desired result. Consider the case where a ray just grazes the edge of an antialiased vector, such that α_V is almost zero. The cumulative intensity is then:

$$I = I_S dz + (1 - dz\alpha_S)I_S(1 - dz)$$

but, the intensity of a neighboring pixel which does not intersect the vector is simply I_S. Thus for the **C*D** integration calculation, the formulas:

$$
\begin{aligned}
I &= I_S dz + (1 - dz\alpha_S)I_V + (1 - \alpha_V)I_S(1 - dz) \\
\alpha &= \alpha_S dz + (1 - dz\alpha_S)\alpha_V + (1 - \alpha_V)\alpha_S(1 - dz)
\end{aligned}
$$

should be used. These are then composited into the image.

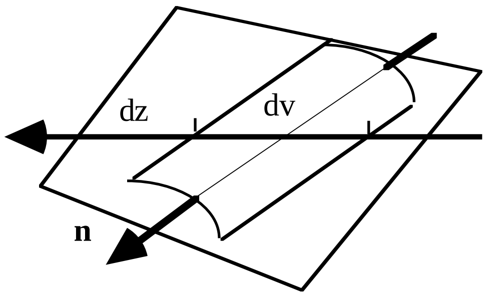

Figure 4. Integrating along the viewing direction.

Efficiency Considerations

At least three possible tests can be used to reduce computations and thereby improve the efficiency. The first is on the length of the vector. If the magnitude of the vector falls below a certain threshold, then the calculations needed to render it can be skipped. With this comes the second test, on the maximum contribution of the splat. If the opacity of the scalar field falls below some threshold, then the calculations to render it can be skipped. Finally, the biggest win comes when both the above conditions are true. In this case, the entire kernel can be skipped.

The size of the resulting image, the span of the filter, and the stride of the filter all have an effect on the performance of the filter. Smaller images and filter size and larger strides can improve the performance of the filter. The resolution of the image's z-space also has a significant impact on the performance of the filter. All of these variables are specified under user control.

The simplicity of a filter makes it a natural choice for vectorization and parallel processing. Each pixel within the filter requires the same arithmetic, allowing it to be computed in parallel on even a SIMD machine. For the 2D filter or a top down view with the 3D filter, several instantiations of the filter kernel can also operate in parallel.

Finally, since the filter samples both the vector and the scalar field, large amounts of memory may be necessary to maintain this data. However, the filter does process this data in a fairly sequential order.

Results

Figures 5 and 6 are taken from an HDTV animation presented at the SIGGRAPH '92 Film and Video show. Figure 5 illustrates the direct volume rendering of just the wind velocities, while Figure 6 illustrates the wind velocities and the percent cloudiness. All of this data was calculated from a global climate model with grid dimensions of 320 by 160 by 19. Figure 5 required 30 seconds to generate on a SGI Personal IRIS at NTSC resolution. Figure 6 required one minute. The simulated data consists of clouds and winds at every hour for ten days. Each day of the simulation generates 380Mb of data for the wind and percent cloudiness fields. Figure 7 shows an oblique view of a test function, simulating a tornado. Figure 8 illustrates the electric field around an within a small portion of a Boeing 737 jet, the avionics' bay.

Future Work

The above techniques provide an effective solution to the simultaneous display of a single scalar field and a single vector field. This allows the scientists to study the complex relationships between the winds and the clouds or the winds and a specific atmospheric heating term. However, the scientists still need to understand the complex dynamics between the winds and several scalar variables (i.e., percent cloudiness, incoming and outgoing radiation, percent humidity, etc.). This is a general research topic to be addressed in not only the vector domain, but the scalar domain as well.

We have simplified the problem here by flattening the terrain in the climate models and dealing only with a regular grid. In fact global climate models and many other grand challenge problems deal with irregular topologies which must be dealt with.

We have also concentrated our attention on the techniques and representations, rather than on efficient solutions. While the techniques are fairly efficient, improvements must still be made to achieve interactive levels. The use of table lookups as described by Laur and Hanrahan [Laur91], and Westover [Westover90] and the use of Gouraud shaded or hardware texture mapped polygons should be evaluated.

Finally, the technique outlined here does not take into account the overlap of the filters when drawing the vectors. This involves a trade-off decision between these inaccuracies and the complexity associated with keeping vectors consistent across splat or voxel domains.

Acknowledgments

We would like to thank Jerry Potter, Bob Mobley and Dean Williams of the Program for Climate Model Diagnosis and Intercomparison (PCMDI) at the Lawrence Livermore National Laboratory for providing us with the data and direction. The European Centre for Medium-range Weather Forecasting (ECMWF) provided the analytical model used to generate the data. Gene Cronshagen and Chris Anderson helped in generating and recording the animations. Steve Pennock provided the Boeing 737 data. Becky Springmeyer provided many useful comments on the draft.

Support for this research came from the Department of Energy's High Performance Computing and Communications Program through the effort on *Visualization For Global Climate Modeling.* This work was performed under the auspices of the US Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

[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.

[Globus91] Globus, A. and C. Levit and T. Lasinski. *A Tool for Visualizing the Topology of Three-Dimensional Vector Fields.* In *Proceedings Visualization '91.* Gregory Nielson and Larry Rosenblum eds., IEEE Los Alamitos, CA, pp.33–40.

[Helman91] Helman, J. and L. Hesselink. *Visualizing Vector Field Topology of Three-Dimensional Vector Fields.* IEEE Computer Graphics and Applications Vol. 11 No. 3, pp 36–46.

[Kajiya89] Kajiya, James T. and Timothy L. Kay. *Rendering Fur With Three Dimensional Textures.* Computer Graphics Vol. 23 No. 3 (July 1989, SIGGRAPH '89) pp. 271–280.

[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. 17–20.

[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.

[Max91] Max, Nelson, "An Optimal Filter for Image Reconstruction," *Graphics Gems II* (James Arvo, ed.). Academic Press, Boston.

[Max92] Max, Nelson, Roger Crawfis, and Dean Williams, *Visualizing Wind Velocities by Advecting Cloud Textures.* Proceedings of Visualization '92, IEEE.

[Potter91] Potter, Gerald. private communication.

[Reeves85] Reeves, William T. and Ricki Blau. *Approximation and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems* Computer Graphics Vol. 19 No. 3 (July 1985, SIGGRAPH '85) pp. 313–322.

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

[Shirley89] Shirley, Peter, and Henry Neeman *Volume Visualization at the Center for Supercomputing Research and Development.* In *Proceedings of the Chapel Hill Workshop on Volume Visualization,* pp. 17–20.

[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.

[VanGelder92] Van Gelder, Allen and Jane Wilhelms. *Interactive Visualization of Flow Fields.* 1992 Volume Visualization Workshop (this issue), Kaufman and Lorensen (eds), ACM SIGGRAPH, NY.

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

[Wijk90] Wijk, J.J. van, *A Raster Graphics Approach to Flow Visualization.* in Vandoni, C.E., and D.A. Duce (eds.), *Proceedings Eurographics'90,* North-Holland, Amsterdam, 1990, pp. 251-259.

[Wijk91] Wijk, J.J. van, *Spot Noise.: Texture Synthesis for Data Visualization* Computer Graphics Vol. 25 No. 4 (July 1991, SIGGRAPH '91) pp. 309–318.

[Wilhelms91] Wilhelms, Jane and Allen Van Gelder, *A Coherent Projection Approach for Direct Volume Rendering Visualization.* Computer Graphics Vol. 25 No. 4 (July 1991, SIGGRAPH '91) pp. 275–284.

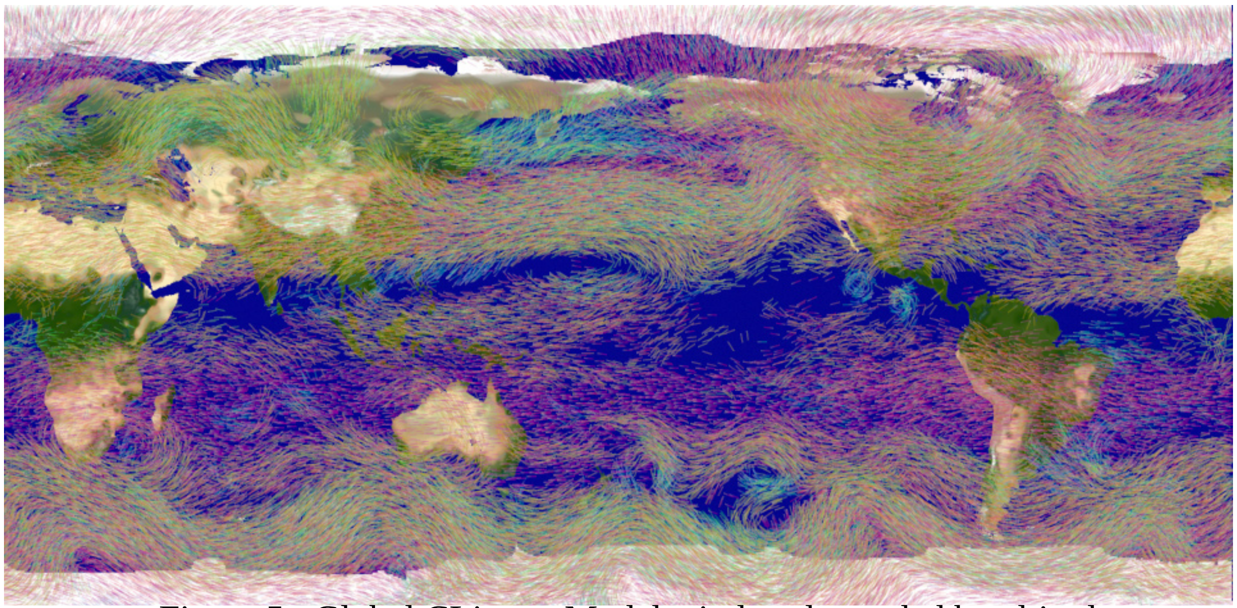
Figure 5: Global CLimate Model winds color coded by altitude

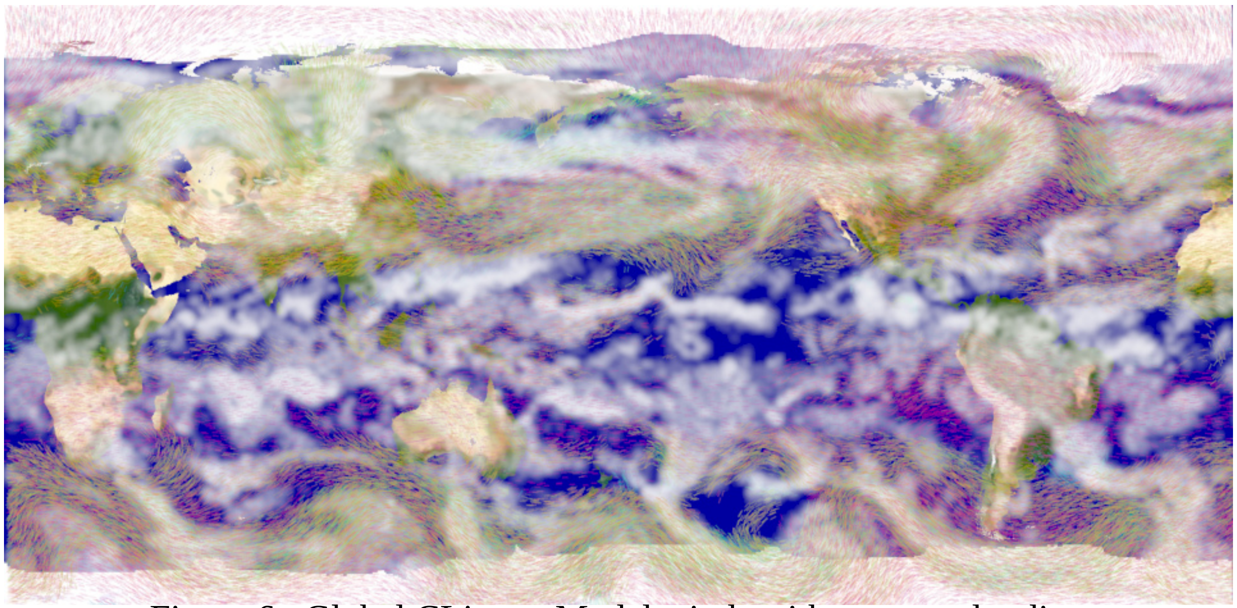
Figure 6: Global CLimate Model winds with percent cloudiness

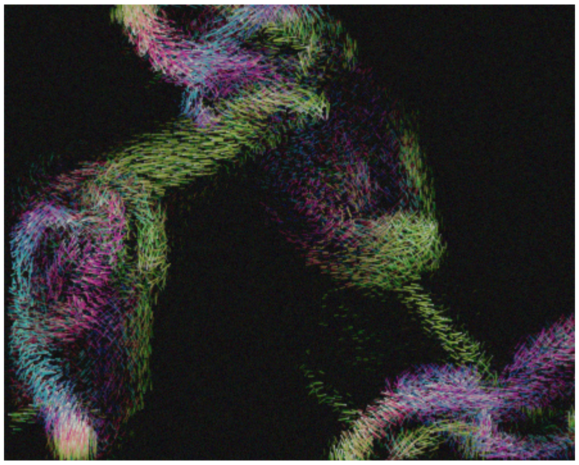
Figure 7 Vorticity of Fluid Flow
- not Test Tornado

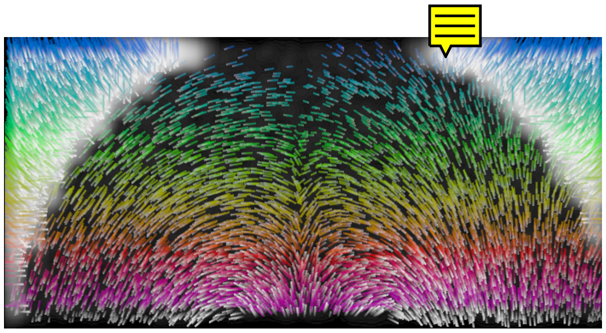
Figure 8: Avionics bay of a Boeing 737.
Electric field excited by an incident plane wave