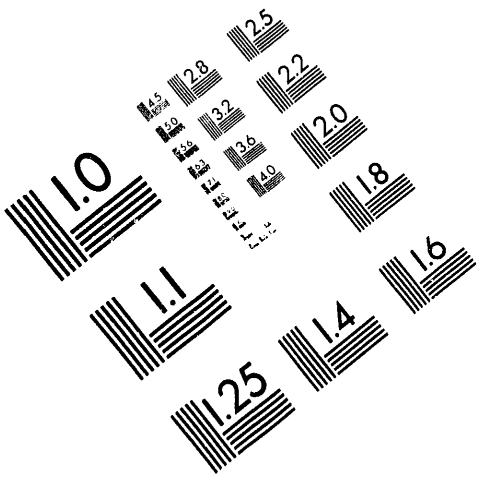
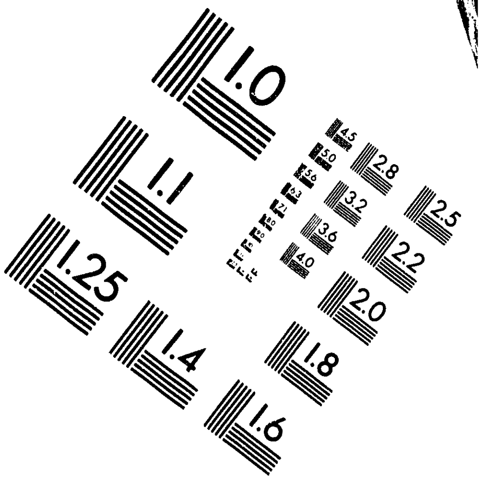


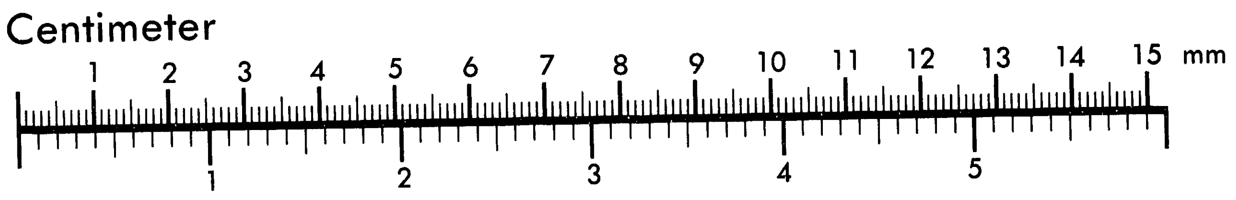
AIIM

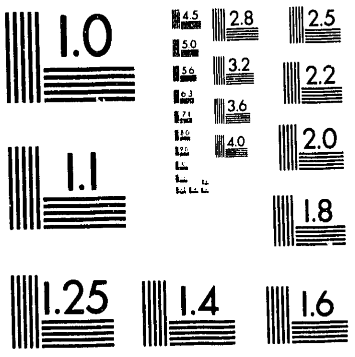
Association for Information and Image Management

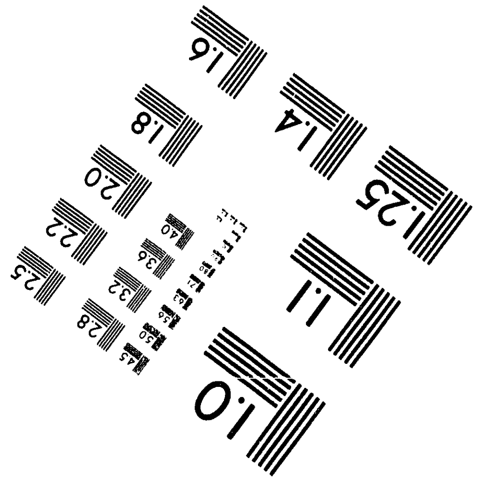
1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910

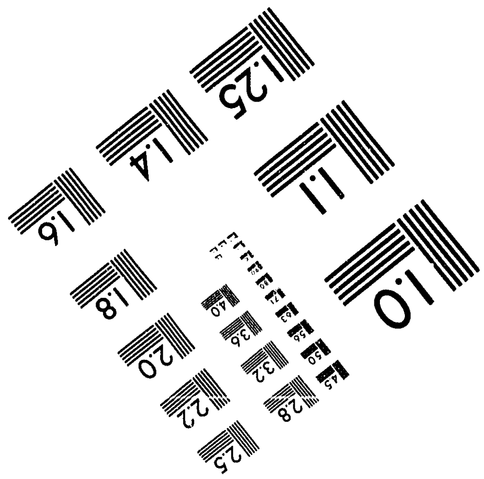
301/587-8202











MANUFACTURED TO AIIM STANDARDS
BY APPLIED IMAGE, INC.

Conf-9308128--1

UCRL-JC-113517
PREPRINT

New Techniques in 3D Scalar and Vector Field Visualization

Nelson Max
Roger Crawfis
Barry Becker

This paper was prepared for submittal to
Pacific Graphics '93
Seoul, South Korea
Aug. 30- Sept. 2, 1993

May 5, 1993

New Techniques in 3D Scalar and Vector Field Visualization

Nelson Max
Roger Crawfis
Barry Becker
Lawrence Livermore National
Laboratory
P.O. Box 808 / L-301
Livermore, California 94551, USA
(max2@llnl.gov)

Abstract

At Lawrence Livermore National Laboratory (LLNL) we have recently developed several techniques for volume visualization of scalar and vector fields, all of which use back-to-front compositing. The first renders volume density clouds by compositing polyhedral volume cells or their faces. The second is a "splatting" scheme which composites textures used to reconstruct the scalar or vector fields. One version calculates the necessary texture values in software, and another takes advantage of hardware texture mapping. The next technique renders contour surface polygons using semi-transparent textures, which adjust appropriately when the surfaces deform in a flow, or change topology. The final one renders the "flow volume" of smoke or dye tracer swept out by a fluid flowing through a small generating polygon. All of these techniques are applied to a climate model data set, to visualize cloud density and wind velocity.

1. Climate Modelling

The Program for Climate Model Diagnosis and Intercomparison at Lawrence Livermore National Laboratory (LLNL) is comparing different climate models which give differing predictions of global warming. Clouds have a significant influence on global temperatures, because they reflect the short wave radiation from the sun, and also absorb and reflect the longer wave radiation re-emitted by the earth. It is not certain which of these effects is larger, hence the influence of clouds is an open research problem.

In this paper we report on a number of techniques we have developed for visualizing climate variables, particularly clouds and wind velocities. These include volume rendering, textured contour surfaces, vector field rendering, and flow volume rendering.

The global climate simulation for 10 days in January was run on a Cray 2 at the National Energy Research Supercomputer Center at LLNL. The computational grid was defined by 320 evenly spaced longitudes, 160 unevenly spaced latitudes, and 19 unevenly spaced surfaces of constant "geopotential", nearly a million grid points. The surfaces of constant geopotential were not at constant altitude, but curved over the mountains, so that the bottom level corresponds to the height of the terrain.

The partial differential equations for weather propagation were solved in the spatial frequency domain, and the resulting spherical harmonic expansions of the climate variables were sampled at the computational grid. The solver only output the requested data variables at one hour intervals, and we interpolated this output to produce animation frames every fifteen minutes.

2. Polyhedron Compositing

There are two chief ways to visualize a scalar function in a volume: (1) draw contour surfaces, or (2) integrate a continuous volume density along viewing rays. The polyhedron compositing scheme of Max, Hanrahan, and Crawfis[1] combines both of these techniques by subdividing the volume cells at contour surfaces, and compositing the resulting polyhedral pieces and surface polygons in back-to-front order. We first divide any volume cell crossed by a contour surface into tetrahedra, and then slice each tetrahedron crossed by a contour surface into two parts. This allows the color and opacity to change discontinuously across a surface, whether or not the surface itself is to be rendered. Also, the sorting allows multiple transparent contour surfaces to be composited in the correct order.

The basic compositing algorithm takes an object (a volume cell or a surface polygon) with color I and transparency T, and combines it with a background frame-buffer color F_{old} to give an updated color F_{new} :

$$F_{new} = T \cdot F_{old} + I .$$

For the volume cells, we used the *density-emitter* model of Sabella[2]. Assume a volume is filled with a variable density $\rho(x)$ of particles, which absorb a fraction $\tau\rho dx$ of the light along a differential length dx, and emits an amount $c\rho dx$ of colored light of its own. Then it can be shown[1] by integration that the transparency along a segment of length L is

$$T = \exp\left(-\tau \int_0^L \rho(x)\, dx\right).$$

The extra colored intensity added by the glowing particles, but attenuated by the opacity of the closer particles, is

$$I = (c/\tau)\,(1-T) .$$

If ρ_{avg} is the average value of ρ along the segment, then

$$T = \exp(-\tau L \rho_{avg}) .$$

These equations assume that c is constant in each volume cell. More complex formulas for I and T are given by Williams and Max[1], for the case that c and ρ both vary linearly along a ray. However, in our current implementation, we just take c to be the average color along the ray segment, which gives a fairly good approximation to the accurate formula.

We input the quantities $C = c/\tau$ and $a = \rho\tau$ at each vertex of the volume cell. (Note that in the above formulas, c, C, I, B, and F are all vectors, with red, green, and blue components.) We use bilinear interpolation across the polygons bounding the cell, as in Gouraud shading, to scan convert C, a, and the depth z into temporary buffers. There are two sets of buffers for each quantity, one for the front facing polygons, and one for the back facing ones. For each pixel (*i, j*) in the projection of the cell, the following computations achieve the results of the formulas above:

```
L = zback(i,j) - zfront(i,j)
a = (aback(i,j) + afront(i,j))/2.
C = (Cback(i,j) + Cfront(i,j))/2.
T = exp(-L*a)
I = C*(1. - T)
F(i,j) = T*F(i,j) + I
```

It appears that the six temporary buffers must be the size of the whole image, to allow for large cells, but in fact we scan convert all the faces of the cell together, one scan line at a time, so only one line's worth of temporary buffers are required. As in the usual implementation of Gouraud shading, we use incremental methods across the scan line, and also from one scan line to the next. In addition, the closed-form analytic integration along the ray segments eliminates the need to sample along the ray. Thus we take advantage of the area coherence across each face, and the volume coherence in the cell.

The above formula for *L* assumes orthogonal projection. For perspective projection, the quantity to interpolate should be[4] -1./*z*, so

```
L = (1./zfront(i,j) - 1./zback(i,j))*sqrt(1. + x*x + y*y)
```

where *x* and *y* give the coordinates where the ray from the eye through the pixel (*i, j*) intersects the plane *z* = 1, and are easily computed from *i* and *j*.

This method requires that each ray intersect the volume cell in a single segment, so that there is a single front and a single back entry for each interpolated variable at each pixel. If the cell is convex, this is always true. We can also guarantee that it will be true for certain non-convex cells which arise in our atmosphere subdivision.

The first alternative, described above, requires a face between two cells to be scan converted twice: once as the front face of the rear cell, and once as the rear face of the front cell. A second alternative method scan converts most faces only once. Faces which have a volume cell to their rear are marked as front faces, to indicate that the cell is to be rendered when they are scan-converted. If there is to be a discontinuity in opacity or color across a face, two copies must appear, one marked and one unmarked, each with their own *a* and *C* values. Instead of sorting the cells, we sort the faces, taking care to put the duplicate copies of any faces in the order corresponding to the cells they bound. Then, for all polygons in back to front order, the second algorithm is

```
For each pixel (i,j) in polygon{
  Interpolate anew, Cnew, and znew
   at (i,j)
  If polygon is a front surface{
    L = ztemp(i,j) - znew
    a = (atemp(i,j) + anew)/2.
    C = (Ctemp(i,j) + Cnew)/2.
    T = exp(-L*a)
    I = C*(1.-T)
    F(i,j) = T*F(i,j) + I
    }
  ztemp(i,j) = znew
  Ctemp(i,j) = Cnew
  dtemp(i,j) = dnew
  }
```

This requires only one set of temporary arrays, but now they must be the size of the whole image. In addition, this version can deal with cases where a viewing ray intersects a cell in two or more segments. However it requires a valid sort on the polygons, which is not always possible. Lucas[5], who has published this alternative, for constant a and C, proposed sorting the faces by the z value of their centroids. He remarked that sorting errors result in incorrect negative values of L, which are mostly compensated by an equal incorrect increase in an adjacent positive value of L.

In our climate work, we used the first alternative, and sorted the cells. Figure 1, showing clouds color coded by altitude, was rendered in this way.

Figure 1. Volume rendered clouds wrapped around a globe with a large vertical exaggeration. The terrain and the clouds are color coded by altitude, using two different scales. Reprinted with permission from "Focus on Scientific Visualization", H. Hagen, H. Müller, G. Nielson editors, Springer Verlag, Berlin

3. Splatting

Westover[6] introduced an alternative method of volume rendering, called "splatting", which approximately interpolates the volume densities by using a rotationally symmetric reconstruction kernel (a "splat") for each data point. The kernels are multiplied by the color and opacity values for the data point, and the results are composited in back to front order. Crawfis and Max[7] made two changes to this method. First, instead of a Gaussian kernel, we used a piecewise quadratic kernel[8], which optimally reconstructs a flat field of constant data values in 2D. Second, we added a representation of vector fields. We drew anti-aliased line segments, whose direction represented the projected vector direction at a sample point, and whose intensity and opacity represented the vector's magnitude. In order to remove any blocky visual patterns from regular sampling, the segments are made long enough to interleave, and the sample positions at the segment centers are jittered. The result of many such vectors is a fine texture as in figure 2, which indicates wind direction. Even though the segments do not progress, the texture appears in animation to move and swirl with the wind.

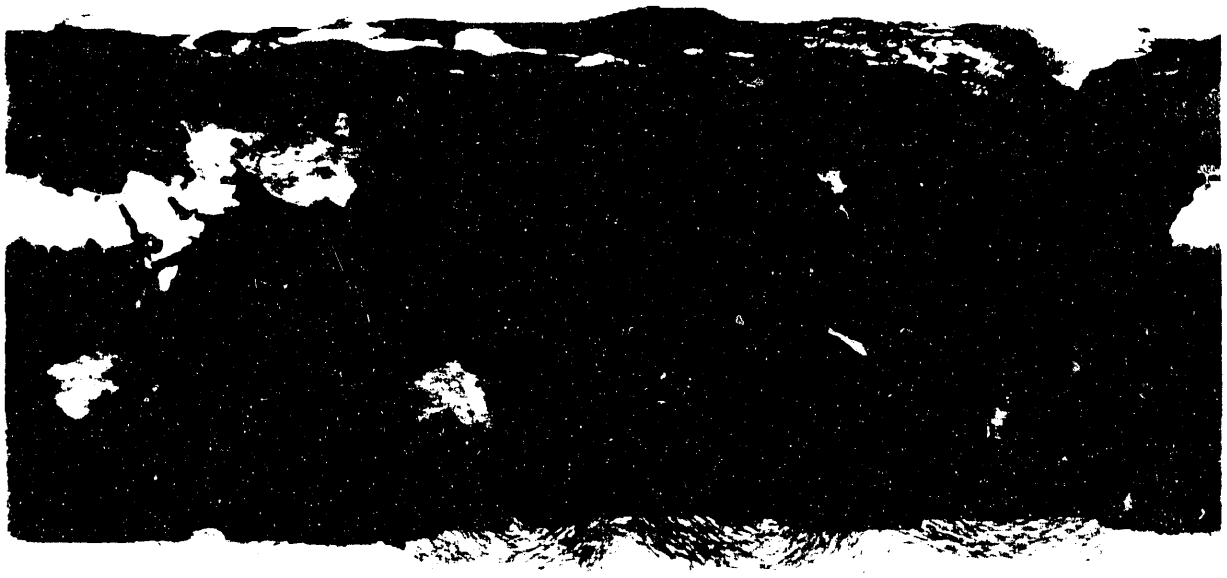


Figure 2. Global winds, drawn with a vector texture color coded by altitude. Reprinted by permission of the Association for Computing Machinery from reference 7.

Figure 3 shows a combination of scalar splats, representing cloud density at an atmospheric layer, and vector line segments, representing wind velocity. The colors of the vectors represent altitude. The compositing alternated layers of scalar splats with layers of vector splats, but the two can also be combined into a single layer[7], so that they partially hide each other.

At first, we used software to evaluate the color and opacity for the piecewise quadratic reconstruction kernel and the anti-alised vector segment, and to do the compositing. Figure 2 took 30 seconds at HDTV resolution (1920 x 1035) on an SGI Personal Iris 4D/35, and figure 3 took one minute.

Recently, we have improved this technique to take advantage of the hardware compositing and texture mapping in the SGI VGX graphics rendering system. We stored an intensity/opacity texture map of the reconstruction kernel, and of a windowed directional vector texture. Then for each sample data point, we composited in hardware a polygon which accessed this texture. We arranged for the polygon to always face the viewer, so that the texture would not be foreshortened.

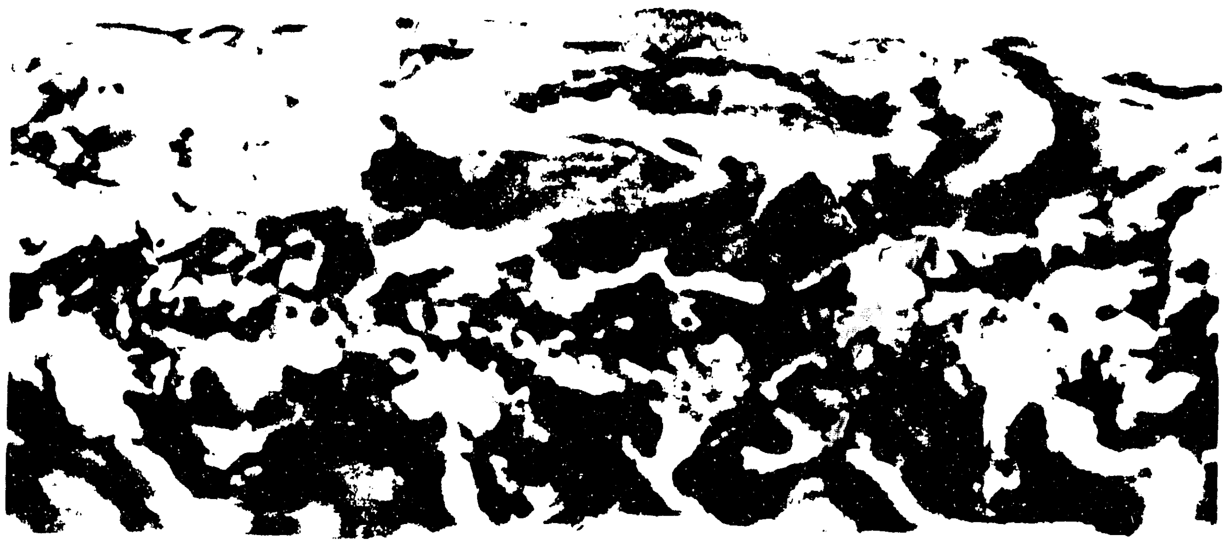


Figure 3. Global winds and clouds, composited together in the same image. Reprinted by permission of the Association for Computing Machinery from reference 7.

When used in this way, the texture should be the 2-D projection of a 3-D reconstruction kernel. For the kernel, we found[9] a C^1 piecewise cubic function of 3-D radius, which optimally reconstructs a flat field from constant data values at the 3-D integer lattice. The resulting 2-D splat is then the line integral of this function along rays perpendicular to the projection plane. These can be computed in closed form. Figure 4 shows the cloud data rendered with this technique.

Figure 4. Clouds over North America, rendered by texture mapped splats. Reprinted from reference 9.

To render vector fields, we use an anisotropic texture which indicates the vector direction, as shown in figure 5. In order to rotate the texture direction, it is only necessary to change the orientation of the polygon accessing the texture map. The texture could also be moved during animation by translating the texture coordinates along the direction of the vectors. However, we needed to multiply the texture by a window function similar to the scalar reconstruction filter, so that adjacent texture samples would blend together when composited. Our hardware could not multiply two separate textures before compositing, so we used the windowed version in the texture map. To achieve animation, we stored windowed versions of the texture in sixteen different translations, and cycled between them.

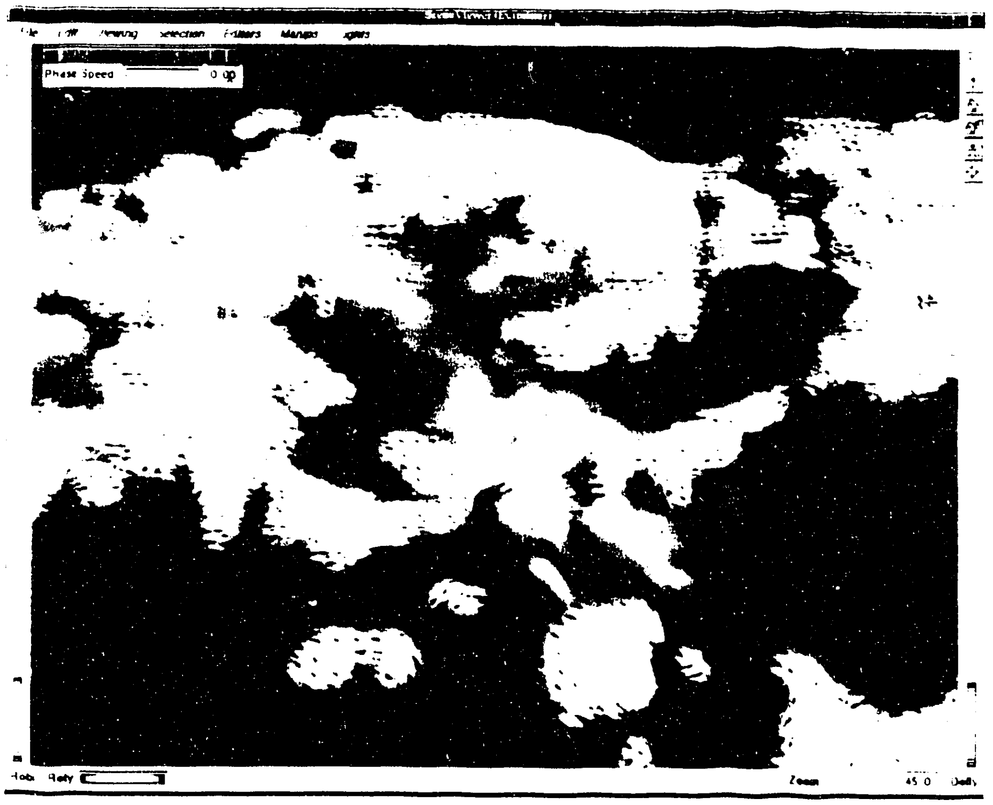


Figure 5. Clouds and wind field over North America, rendered using textured splats. Reprinted from reference 9.

Laur and Hanrahan[10] have implemented an octree hierarchy of splats, by grouping together eight volume cells if their data values are within an error tolerance of their average. They use a piecewise linear approximation to a gaussian splat, which is implemented as a net of polygons. The hardware rendering pipeline interpolates and composites the colors and opacities specified at the polygon vertices. We obtained the source to an SGI Explorer™ module implementing this technique, and replaced the piecewise linear splats by our smooth texture mapped splats, to give a smoother reconstruction. In the case of many small splats, where the per-polygon overhead outweighs the texturing cost, it is also faster.

4. Cloud Texturing

The grid cells in our climate model are 1.125 degrees on a side, or about 100 kilometers square at the equator, so the cloud density variable does not represent the density within individual clouds, but rather the percent cloud cover at a particular altitude level. The cloud rendering method of Gardner[11] creates transparency where the texture function is below a threshold, and can thus create partial cloud cover detail within a single grid element. It therefore gives a more faithful representation of the meaning of the cloud density variable in the simulation than does the direct volume rendering in figure 1.

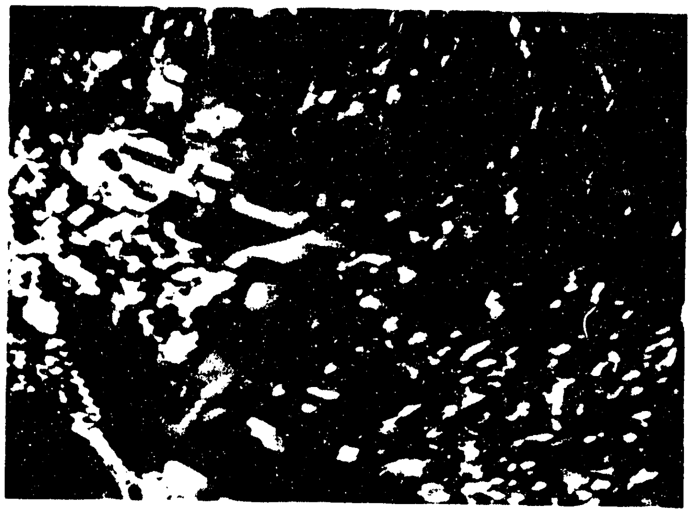


Figure 6. Contour surface of 15% cloudiness from a climate model, rendered with haze. Reprinted with permission from reference 12.

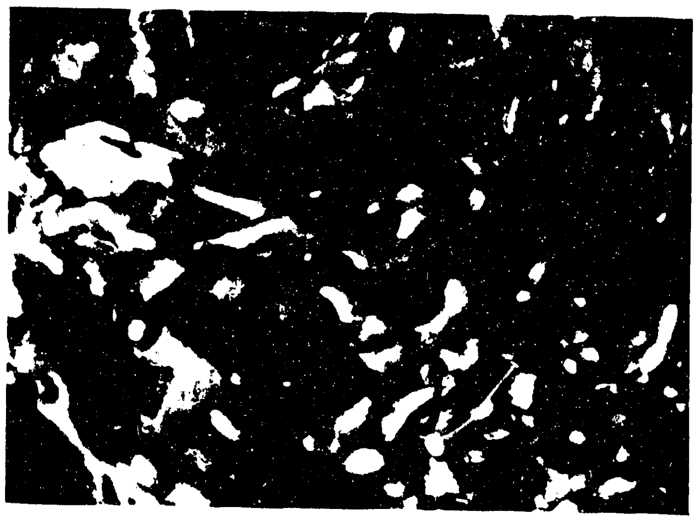


Figure 7. Contour surface after the cloudiness function has been smoothed by a 3x3x3 filter. Reprinted with permission from reference 12.

We found that when the original cloud density from the simulation was contoured, the surface was too rough and bumpy, as in figure 6. Therefore we applied a 3x3x3 smoothing kernel to the density before contouring, giving figure 7. The corresponding textured clouds are shown in figure 8. Figures 6 through 8 all show the contour surface of 15% cloud cover.

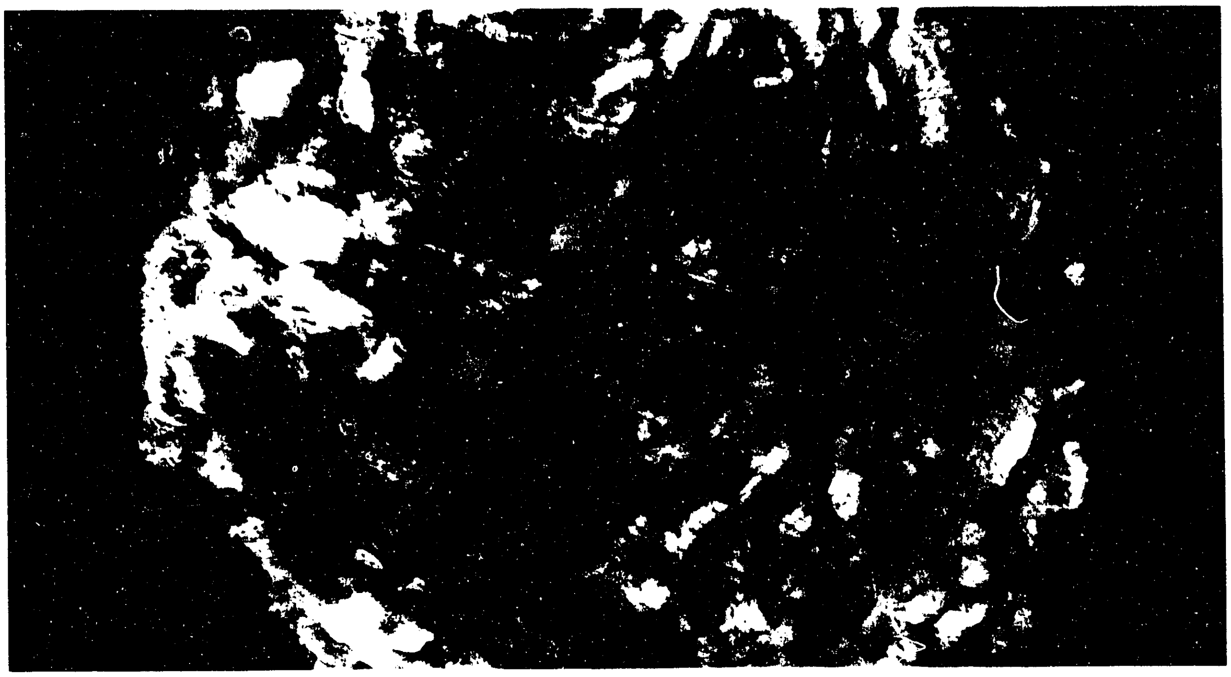


Figure 8. Smoothed contour surface of 15% cloudiness, rendered with a 3D transparency texture. Reprinted with permission from reference 12.

Since the contour surfaces have complicated shape and topology, which change for every frame of animation, it is difficult to assign 2D texture coordinates to the vertices. Therefore, like Gardner, we used a 3D volume texture function, evaluated at surface points, to determine both the color and opacity of the clouds. When this function is below a threshold, the clouds become completely transparent and colorless. The 3-D texture coordinates (u,v,w) range from 0 to 1, and are initialized proportional to longitude, latitude, and altitude, respectively. The longitude wraps around at the Greenwich meridian, which corresponds to both $u = 0$ and $u = 1$. The next section explains how these texture coordinates are advected by the wind to produce, for each frame of the animation, new (u,v,w) coordinates at each of the grid vertices. Once these are known, they can be interpolated across grid edges to the vertices of a contour polygon, and then across the polygon during scan conversion, texturing, and shading. We used a variant of Gardner's "poor man's fractal texture" with a sum (instead of a product) of a small number of 3-D plane sine waves. As suggested by Gardner, we perturbed the phase of each of these waves by another sine wave of longer wavelength in a perpendicular direction, to break up the regularity in the texture pattern. The texture function is thus of the form

$$f(u, v, w) = \sum_i a_i \cos(b_i u + c_i v + d_i w + q_i)$$

where the phase perturbation is

$$q_i = 2 \cdot \sin(0.5 b_i u - 0.5 a_i v) .$$

We sorted the volume cells from back to front, and then rendered in order the front-facing cloud contour surfaces they contained, using Phong shading, with the Gardner color and transparency texture. In order to make the clouds appear more opaque where they are thicker, and more transparent and tenuous near their edges, we added a term proportional to the cloud thickness to the texture function before comparing it to the threshold. When an opaque terrain polygon or a back-facing contour polygon is scan converted, the z-buffer is set to the polygon's depth at each pixel. Then when a front-facing contour polygon is rendered, its depth can be subtracted from the depth in the z-buffer to get the length of the viewing ray inside the volume bounded by the contour surface.

This cloud thickness effect assumes that the cloud density is constant inside the cloud contour surface. We can take into account the variable density by scan converting the volume cells using one of the methods described in section 2, and accumulating their integrated densities *aL* in a special density buffer. When a front-facing contour polygon is scan converted, the accumulated density is used to modify the transparency threshold, and the buffer is reset to zero. Unfortunately, this technique did not seem to add to the realism or the information content of our cloud images.

One thing that does add to the realism is haze, which also serves as a depth cue to distinguish the cloud layers. We used a haze density in figures 7, 8, and 9 which decreases exponentially with altitude. It could be integrated analytically along any ray segment to give an expression[12] involving the normal error function, which was stored in a table.

5. Advection of texture coordinates

The best way we discovered to make the texture move appropriately as our clouds deformed in the wind, was to advect the cloud texture coordinates by the wind. When the cloud surface is being pushed by the wind, the texture and geometry will move coherently, while in a standing wave at a mountain range, the texture will move across the cloud, indicating the wind velocity. However when a cloud grows quickly as in a thunderstorm, or in the daily rain cycle over the Amazon rainforest, it will spread through the 3-D texture space, causing a "boiling" appearance.

Initially, in frame 1 of an animation, the three texture coordinates at a grid vertex are proportional to the longitude, latitude, and altitude indices, and vary between 0 and 1. The Euler method for integrating ordinary differential equations can be used to move the grid vertices forward along the streamlines of the wind flow field. In frames after the first, grid vertices move into the interiors of grid cells, so the velocity must be interpolated from the eight surrounding grid vertices.

Since the advected grid becomes distorted, the texture coordinates must be resampled into the standard fixed grid at which the other simulation data is defined. Unless the inverse of the advection mapping is known, it is hard to decide which distorted cell contains a given grid vertex. In fact, the grid cells could become so distorted that their surfaces self-intersected, making the decision even more difficult. We therefore computed the inverse advection mapping for each grid vertex, instead of the forward mapping. For each grid vertex P in frame n, we found the point Q in frame 1 which would end up at P, by integrating backwards along the streamline through P. The texture coordinates at P are then proportional to the longitude, latitude, and altitude at Q.

The streamlines through the grid points for frame n do not pass through the grid points for frame n-1, so a separate backwards integration is needed for each frame. This makes the number of steps in the Euler integration quadratic in the number of frames. It is also necessary to have the velocities for all the time steps available to compute the backwards integration for the last frame. This was impractical for us, since the data output for each 24-time-step day takes 380 megabytes. In addition, the wind will eventually twist and stretch the cloud texture until it contains only high spatial frequencies, which will spoil the intended visual effect.

For all these reasons, we used each set of texture coordinates for only 24 hours of simulation time. To keep a continuous appearance, we rendered the clouds using a linear combination of 3-D textures computed from two sets of coordinates. We increased the weight of each set from zero to one in 12 hours, and then back to zero in another 12, so that a new set replaced an old one every 12 hours. For hour n of a 12 hour cycle, one set of coordinates is obtained by integrating backwards for n-1 steps along the streamlines, and the other by integrating along the same streamlines for a further 12 backward steps. These integrations took an hour of CPU time, but were done only every four frames; we interpolated the coordinates in between. We have produced a HDTV animation of 960 frames from our 10 day climate simulation, and figure 9 is a sample frame. The average frame took 45 minutes, including texture advection integration, rendering, and data I/O.

If we had used linear interpolation between two shaded cloud images, the overlap of cloud patterns would have been obvious. Instead, we combined the two 3-D textures before applying the non-linear transparency threshold, so only one cloud pattern will be visible.

6. Flow Volumes

Engineers often visualize flows by releasing visible smoke into a gas flow, or dye into a liquid. We developed a computer simulation of this effect by volume rendering a "flow volume", the volume swept by the flow through a small generating polygon P which releases the colored tracer. At present, we are using only steady flows, where the velocity is independent of time, so that we can achieve near real time interaction. Rapid imaging is possible because we need only render the small cells inside the flow volume, instead of accessing the whole data volume, and because we use the hardware pipeline for shading and compositing.

At any time step t, the flow volume is bounded by the generating polygon P, the position P_t to which its points are carried by the flow by time t, and the stream ribbons[13] generated by the sides of P. Thus flow volumes are a volume generalization of stream ribbon surfaces, and involve the same sort of subdivision considerations as described by Hultquist[13]. Max Becker and Crawfis[14] give the details of the adaptive 3-D subdivision into tetrahedra. Without too much extra work, compared to stream ribbons, stream volumes give a visual impression which is similar to the familiar engineering physical experiments.

The polyhedron compositing algorithms described in section 2 required software scan conversion, in order to compute the transparency $T = \exp(-L^*a)$, involving an exponential at each pixel. However, by using the tetrahedron projection method of Shirley and Tuchman[15], together with a hardware texture map for the exponential, we are now able to achieve hardware compositing. The method of Shirley and Tuchman divides the projection of each tetrahedron into triangles. The two non-degenerate cases, shown in figure 9, result in three or four triangles. In both these cases, the quantity L^*a is non-zero only at the "thick" interior vertex marked A, since L is zero at the other profile vertices. (In the degenerate cases, the thick vertex may also lie on the profile.)

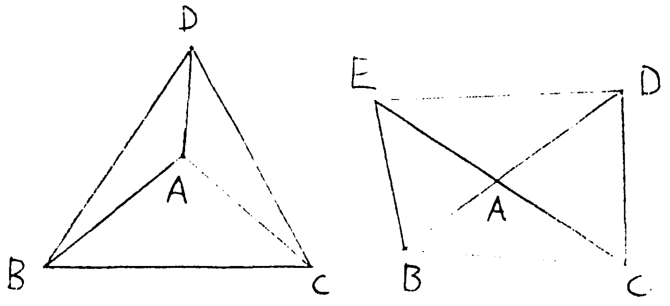


Figure 9. Two non-degenerate tetrahedal projections divided into triangles. Reprinted from reference 14.

Shirley and Tuchman computed $T = \exp(- L^*a)$ at the thick vertex A, and the corresponding color $I = C^*(1 - T)$. They then used the shading and transparency features in the hardware to linearly interpolate T and I across the triangles and do the compositing. This produces artifacts, since the exponential function is not actually linear. In order to remove these artifacts, while still taking advantage of hardware rendering, we put the function $1 - \exp(-u)$ in a 1-D texture table, and used the quantity L^*a as the texture coordinate u at each vertex. The hardware then linearly interpolates this texture coordinate across each triangle, and accesses the texture table to get the "alpha" value $1 - T$ used in the hardware compositing.

As discussed above, volume rendering by polyhedron compositing usually requires that the volume cells be sorted in back to front order, so that they can hide each other appropriately when composited. Such sorting for a twisted flow volume could slow down the interaction. However we can prove[14] that if the smoke has a constant color C, which is the case for our flow volumes, the resulting image is independent of the compositing order, so no sorting is necessary. Using the flexible z-buffer options on the SGI hardware, it is still possible to have opaque objects hide the flow volume cells. First all opaque surface polygons are rendered. Next the triangles in the projections of the flow volume tetrahedra are composited. Their z is compared at each pixel with the z-buffer to determine whether to composite, but the z-buffer is not updated. Figure 10 is an application of our flow volumes to climate data, showing the effect of a typhoon over the Pacific.

The user interface for the flow volume was built using the SGI Inventor™ tools and C++ classes. In addition to specifying the viewing parameters, the user can interactively position the generating polygon P, which always orients itself normal to the flow velocity. The color and opacity of the smoke can also be specified, using Inventor sliders and color design tools. Puffs of smoke can be animated moving along the flow volume, by making the opacity vary with the cell's time step index. Finally, compressible flow can be visualized by making the opacity vary inversely with the volume of the tetrahedral cell.

Figure 10. Flow volume for the wind from the climate simulation, showing the effects of a typhoon in the Indian Ocean. Reprinted from reference 14.

7. Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-ENG-48, with specific support from an internal LDRD grant and also from the High Performance Computing and Communications Program directed to Visualization For Global Climate Modeling. The terrain was rendered using subroutines of Michael Allison and Chris Anderson, and data from Brian Cabral. The simulation output came from Bob Mobly and Chris Anderson. We thank Dean Williams and Gerald Potter for useful conversations.

References

1. N. Max, P. Hanrahan and R. Crawfis (1990) "Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions", Computer Graphics Vol. 24 No. 5, pp 27-33

2. P. Sabella (1988) "A Rendering Algorithm for Visualizing 3D Scalar Fields", Computer Graphics Vol. 22 No. 4 (Siggraph '88 Proceedings) pp 51 - 55

3 P. Williams and N. Max (1992) "A Volume Density Optical Model", Proceedings of the 1992 Workshop on Volume Visualization, ACM, New York, pp 61 - 68

4. W. Newman and R. Sproull (1979) "Principles of Interactive Computer Graphics", McGraw-Hill, New York

5. B. Lucas (1992) "A Scientific Visualization Renderer", Proceedings of Visualization '92, IEEE Computer Society Press, Los Alamitos CA, pp 227 - 234

6. L. Westover (1989) "Interactive Volume Rendering", Proceedings of the Chapel Hill Workshop on Volume Rendering, ACM, New York, pp 9 - 16

7. R. Crawfis and Max N. (1992) "Direct Volume Visualization of Three-Dimensional Vector Fields", Proceedings of the 1992 Workshop on Volume Visualization, Kaufman and Lorensen (eds), ACM SIGGRAPH, NY

8. N. Max (1991) "An Optimal Filter for Image Reconstruction", in "Graphic Gems II", James Arvo (ed), Academic Press, Boston, pp 101 - 104

9. R. Crawfis and N. Max (1993) "Texture Splats for 3D Vector and Scalar Field Visualization", UCRL JC 113~~25~~622, Lawrence Livermore National Laboratory, Livermore CA-submitted to Visualization '93

10. D. Laur and P. Hanrahan (1991) "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rrendering", Computer Graphics Vol. 25 No. 2 (Siggraph '91 Proceedings) pp 285 - 288

11. G. Gardner (1985) "Visual Simulation of Clouds", Computer Graphics Vol. 9 No. 3 (Siggraph '85 Proceedings) pp 297-303

12. N. Max, R. Crawfis and D. Williams (1992) "Visualizing Wind Velocities by Advecting Cloud Textures" Proceedings of Visualization '92, IEEE Computer Society Press, Los Alamitos CA, pp 179 - 184

13. J. Hultquist (1992) "Constructing Stream Surfaces in Steady 3D Vector Fields", Proceedings of Visualization '92, IEEE Computer Society Press, Los Alamitos, CA pp 171 - 178

14. N. Max, B. Becker and R. Crawfis (1993) "Flow Volumes for Interactive Vector Field Visualization", UCRL JC 113~~25~~626, Lawrence Livermore National Laboratory, Livermore CA submitted to Visualization '93

15. P. Shirley and A. Tuchman (1990) "A Polygonal Approach to Direct Volume Rendering", Computer Graphics Vol. 24 No. 5 pp 63 - 70