Advances in Scientific Visualization

Nelson Max and Roger Crawfis
University of California, Davis, and
Lawrence Livermore National Laboratory

Abstract

This paper discusses scientific visualization of scalar and vector fields, particularly relating to clouds and climate modeling. One cloud rendering method applies a 3-D texture to cloudiness contour surfaces, to simulate a view from outer space. The texture is advected by the wind flow, so that it follows the cloud motion. Another technique simulates multiple scattering of incident light from the sun and sky. This paper also presents a simulation of the microscopic cross-bridge motion which powers muscle contraction. It was rendered by ray-tracing contour surfaces of summed gaussian ellipsoids approximating the actin and myosin protein shapes.

Key Words: scientific visualization, scalar fields, vector fields, flow visualization, muscle, cross-bridge, contour surface.

Climate Visualization

Climate simulations calculate several 3D scalar variables like temperature, humidity, and cloudiness at millions of spatial/temporal sample points. In order to make sense of this massive amount of data, it is useful to produce time-varying computer animated renderings. At Lawrence Livermore National Laboratory, we have concentrated on visualizing cloudiness, a scalar quantity, in conjunction with wind velocity, a vector quantity [1]. As we discuss the rendering techniques for scalar fields below, we will write in terms of cloudiness, but they are equally applicable to any other 3D data. However, our flow visualization techniques are more appropriate to velocity fields than to other vector fields, since they use motion along the flow direction.

We used global climate data output by the Cray 2 for every hour in a 10 day simulation for the beginning of January, using an algorithm from the European Center for Medium Range Weather Forecasting. The data is given on a curvilinear 320x160x19 longitude/latitude/altitude grid. This grid is not in simple spherical coordinates, because the lowest altitude level bends to follow the height of the terrain, and the higher levels also bend to accommodate.

The grid spacing is larger than one degree on a side, so the model cannot represent individual clouds. Instead, the cloudiness variable represents the fraction of the sky that is covered by clouds at the level of the altitude coordinate. Cloudiness is a scalar field, so standard volume rendering techniques can be used. These include contour surfaces, as in figure 1, and direct volume rendering, as in figure 2.

Contour surfaces are built from polygons, and can take advantage of polygon rendering hardware. For figure 1, we divided the hexahedral volume cells defined by 8 sample vertices up into 5 tetrahedra, and interpolated the cloudiness function linearly across each tetrahedron. If a contour surface of the linearly interpolated function intersects a tetrahedron, it does so in a planar triangle or quadrilateral.

To produce figure 2, we used a different interpolation scheme across the cell bounded by the 8 sample points. We interpolated the cloudiness linearly along the projections of the edges, and then linearly across horizontal scan lines to get the value at each pixel center on the projection of each boundary face. Each viewing ray intersects the cell in a segment between two boundary face pixel center positions, so we can linearly interpolate these values across the segment. Such a scheme defines a piecewise trilinear interpolation over any convex polyhedral cell [3]. The shapes of the pieces depend on the cell orientation and projection direction, so the interpolation is not invariant under rotation. (A similar problem occurs with the piecewise bilinear interpolation used with Gouraud shading.) However, this method allows the faces to be scan converted coherently, which increases efficiency, as does treating the cell as a whole instead of dividing it up into tetrahedra.

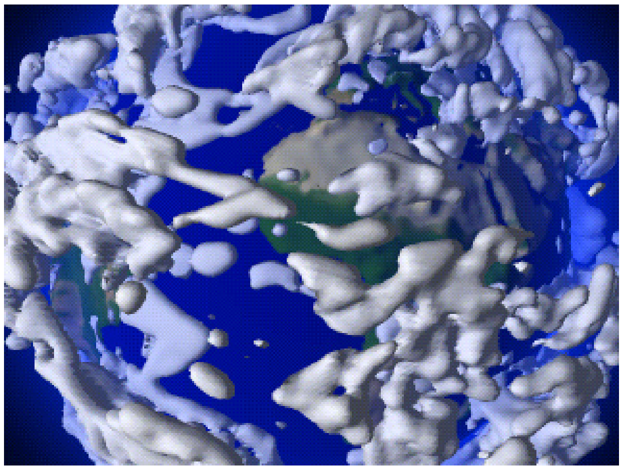

Fig. 1. Countour surfaces of 15% cloudiness.

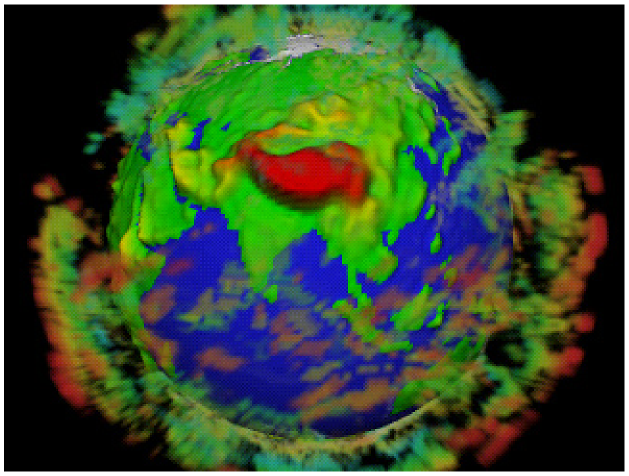

Fig. 2. Volume rendered clouds.

The interpolated cloudiness is interpreted as opacity, which makes the cloud hide objects behind it. The opacity is determined from the cloudiness variable by a “transfer function” which assigns an opacity to each value of cloudiness. We used a piecewise linear function, simplifying the opacity integration along the ray segment in a volume cell. In figure 2, we set the opacity to zero until the cloudiness reached a threshold, leaving a large transparent volume which need not be processed. A separate transfer function can be used to specify the cloud color as a function of another scalar variable like altitude or temperature.

The volume cells are sorted and then composited in back to front order on top of a rendering of the terrain. At each pixel, the current color is multiplied by the transparency of the ray segment, and the color contribution of the segment is added. Williams and Max [2] show how to calculate the color contribution in the piecewise linear case, and Max *et al*. [3] give details of the scan conversion and compositing algorithm.

This algorithm requires a back-to-front sort, which is not always possible. Max [4] describes a sorting method which works for the special geometry of our cloud models. Williams [5] and Stein *et al*. [6] give more general sorting techniques.

Another method of volume rendering, called “splatting” and intoduced by Westover [7], considers the contribution of each data sample separately, instead of interpolating over cells between them. A spherically or ellipsoidally symmetric weighting function is used, instead of the piecewise linear or piecewise trilinear weighting which would result from the interpolation methods discussed above. This means that the weights for the samples contributing to an interpolated value may not sum to 1, but, as shown in Crawfis and Max [8], one can adjust a spherically symmetric weighting function to make them sum to within 0.5% of 1. The 2D *X*-*Y* “footprint” for the projection of this weighting function can then be found by integrating along the *Z* direction, and saved in a texture table. For rendering, a software [7] or hardware [8] texture table reference can provide the necessary opacity at each affected pixel. Figure 3 was produced with hardware texture mapping, using a separate anisotropic texture component to indicate the wind velocity [8]. By cycling through a sequence of different texture maps between frames, the anisotropic texture is made to flow in the direction of the wind velocity.

Another way to indicate wind velocity is to show it advecting texture on the clouds. Inspired by Gardner [9], we developed an analytic “poor-man’s-fractal” texture function, based on sums of perturbed planar sine waves in the longitude, latitude, and altitude coordinates. When evaluated on the contour surface polygons and used to control transparency, this texture approximates the appearance of a satellite photo. Figure 4 shows the front-facing contour polygons in figure 1, rendered with this transparency texture. Blue atmospheric haze, decreasing exponentially with altitude, has been added to increase the realism.

Because of the partial transparency, this image cannot be produced using only a Z-buffer to determine visibility. Instead, we used the sorting algorithm in Max [4] to specify a back-to-front order for the cells containing the contour polygons, and composited the transparency-textured polygons in that order.

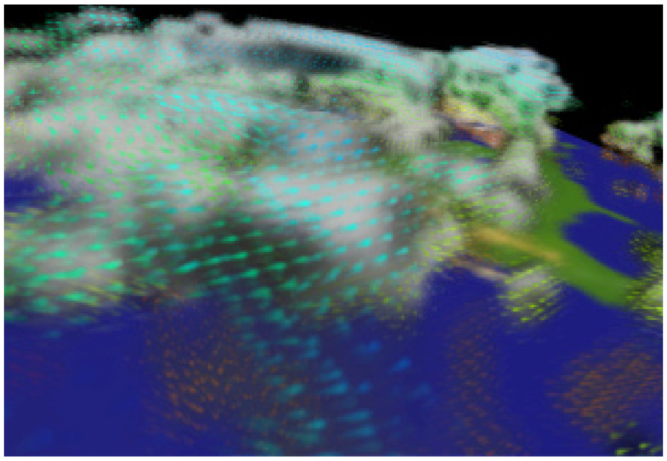
Fig. 3. Clouds and wind velocity by hardware texture mapping.

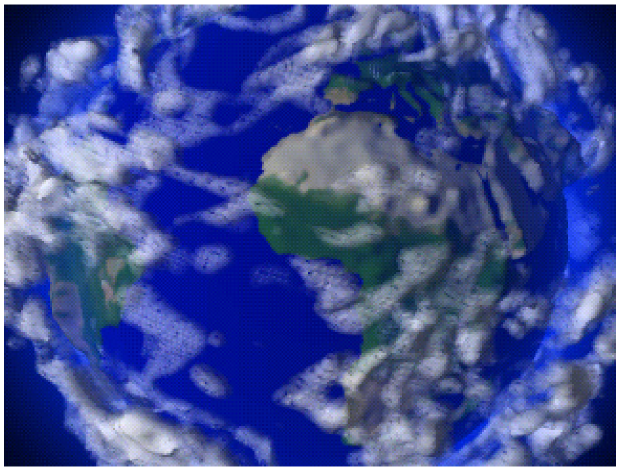
Fig. 4. Clouds of fig. 1 with transparency texture.

In order to make the texture move properly in the wind field, we evaluated the texture function not at the point *A* being rendered, but at the point *B* which, starting at time 0, would reach *A* at the time *t* corresponding to the current frame. To find the texture at *B*, every sample vertex at time *t* was integrated backwards along the time-varying velocity field, to determine where it started at time 0. The starting longitude, latitude, and longitude for the 8 vertices of the cell containing *A* were then interpolated to find the texture coordinates at *B*. (See Max and Crawfis [10] for details.) Figure 4 is a frame from an animation, in which the cloud texture and cloud shapes move in the wind in a consistent fashion.

We can also visualize the wind by simulating smoke introduced into the air flow, a technique used in wind tunnels to visualize flow past airfoils. We start with an initial polygon as the smoke generator, which is always kept perpendicular to the flow field as the user adjusts its center position and size. We integrate the velocity field to find the air volume which has flowed through this polygon in a given amount of time, and divide it up into tetrahedra. The tetrahedra are composited onto the image with a constant smoke color. See Max *et al*. [11] for details of the hardware compositing and for a proof that for constant color smoke, back-to-front sorting is not necessary. With hardware rendering and no sorting, this method can render the smoke in real time as the viewing parameters changed interactively. Figure 5 was rendered in this way.

Cloud rendering

None of the above visualization techniques attempts to simulate the precise interaction of incoming illumination with the water droplets in the cloud. In order to render more realistic clouds, we have simulated the multiple scattering of sunlight and skylight within a cloud. We defined a droplet density on a 24x24x18 grid, using Perlin’s 3D noise texture [12] superimposed on a sum of three ellipsoidal densities. We used a Henyey-Greenstein phase function [13] to specify the directional scattering probability as a function of the angle between the incoming and scattered rays. This function has an adjustable parameter specifying the concentration of the probability in the forward or backward directions, which we set to emphasize forward scattering in order to approximate the phase function of small water droplets.

To simulate multiple scattering, we use a technique which is the volume equivalent to progressive radiosity for surfaces [14]. At each voxel, we store the `total` flux propagating in each of 96 direction bins. A direction cube is centered at the voxel, with each face divided into a 4X4 grid. Each direction bin contains the rays passing through one of the grid squares. We keep a similar array of `unshot` scattered flux which had not yet been propagated. It is initialized using the incoming sun and sky illumination.

In each shooting pass, the beam of `unshot` flux in each direction bin (outer loop) at each voxel (inner loop) is propagated to the other voxels it can reach, one layer at a time. At each voxel, a fraction of the flux, determined by the droplet density data, is intercepted by droplets, and accumulated as `received` flux. This attenuates the beam as it progresses from layer to layer. The flux passing through each voxel is added to the `total` flux array there. We have found a way to amortize the arithmetic for propagation over all voxels shooting flux in one direction bin, so as to compute O(N^2) interactions in time O(N log N), where N is the total number of voxels. (For details, see Max [15].)

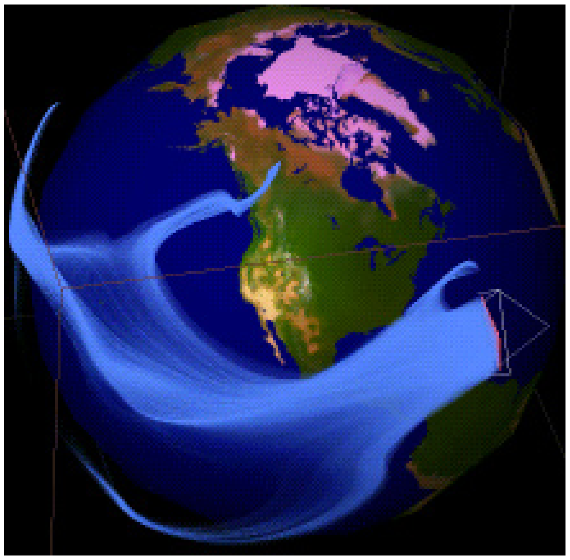
Fig. 5. Simulated smoke in global air circulation.

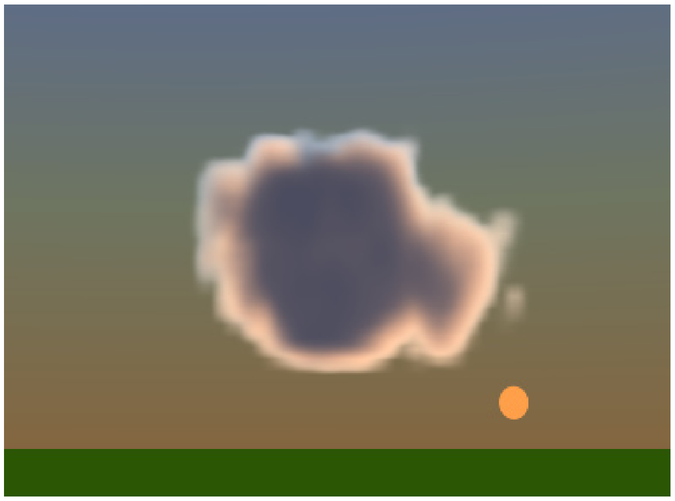
Fig. 6. Multiple scattering of light inside a cloud.

Once the flux for all voxels is shot for a direction bin, the `received` flux is scattered according to the phase function, becoming `unshot` flux again. The fraction of the intercepted flux that is scattered instead of being absorbed is called the albedo. If the albedo is less than 1, the amount of remaining `unshot` flux decreases exponentially with the number of shooting passes, so the shooting iterations converge. Figure 6 used an albedo of .99, and converged after only 15 iterations, because much of the flux left through the edge of the cloud. Each iteration took 15 minutes on an SGI 4D/35 with a Mips 3000 CPU.

Once the `total` flux array has been determined, an image can be produced quickly from any viewpoint by direct volume rendering. We used the ray tracing method of Garrity [16], which took only 2 minutes a frame at 300x255 resolution. Figure 6 is part of an animation where the viewpoint loops around the cloud.

Muscle motion

In the verbal presentation of this paper, we wanted to show the most interesting scientific visualization animation we have helped produce. Therefore we now switch to a completely different topic: muscle motion.

Muscles consist of interleaved fibers of actin, anchored in one structure, and myosin, anchored in another. The “heads” of the myosin molecules form cross bridges, which produce muscle contraction by attaching to actin, forcibly bending at hinge regions so as to pull on the actin fiber, and then releasing it. The energy to drive this cross bridge motion cycle is provided by the splitting of ATP. For a current survey of the experimental evidence for this cycle, see Spudich [17].

Together with a large team of programmers and graphic designers, we produced a computer simulation and animation of this process for the Fujitsu pavilion at Expo ’90 in Osaka, Japan. The simulation involved random Brownian impulse forces, elastic bounces from collisions, and fake animation forces to drive the contraction cycle. The simulation and ray-traced rendering were vectorized to run on a Fujitsu VP200 supercomputer. The ray tracing was also parallelized to run on an experimental system of 32 transputers. For details of the simulation and rendering, see Max *et al*. [18].

The shapes of the actin molecules and myosin heads were represented as contour surfaces of sums of ellipsoidal volume densities. Such shapes are called “blobbies” by Blinn [19] and “metaballs” by Nishimura [20]. The ellipsoids were adjusted to match the shapes reconstructed from transmission electron micrographs of actin fibers with and without attached myosin heads [21]. The “tails” of the myosin molecules were represented by cylinders.

Figure 7 was produced on the transputer system by ray tracing a collection of actin and myosin molecules in a spatially periodic structure. The unit cell contained 6 actin and 3 myosin fibers, so the periodicity is not noticeable in the animation, though it made the simulation easier. It also permitted all the ellipsoid and cylinder descriptions from the simulation output to fit entirely in the memory of each transputer, allowing independent “trivially parallel” ray tracing.

Figure 7 took the 32 transputers 17.5 minutes, at 2048x1500 resolution. It is counter-distorted for Omnimax projection through a fisheye lens onto a tilted dome surrounding a 300 seat theater. At the Fujitsu pavilion, we used Omnimax stereo, with two Omnimax projectors mashed into one. Each projector showed its own horizontally oriented strip of 70mm film, at 48 flashes a second. The 96 flashes a second alternated between the two projectors. The audience wore wireless liquid crystal shutter glasses, synchronized to the alternating flashes by an infrared signal bounced off the dome, so that each eye saw only its appropriate strip of film. This stereo effect put the audience right in the middle of the churning cross bridges, for a thrilling climax to our animation.

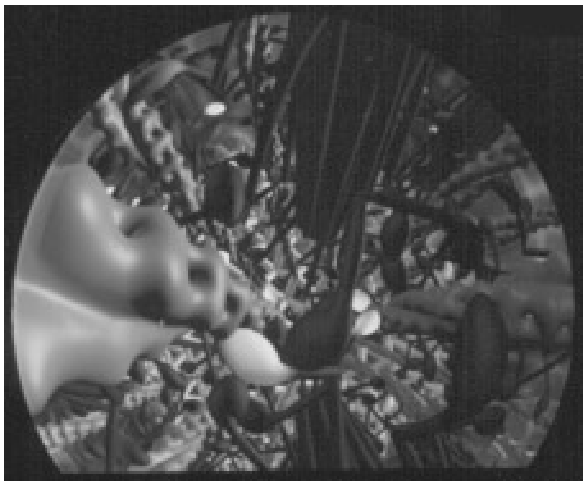

Fig. 7. Muscle cross-bridges in motion during contraction.

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 at Fujitsu Ltd. in Kawasaki, Japan. Barry Becker helped with the smoke rendering in figure 5. Roman Kroitor and Nobuhiko Hayashi helped in the design of the muscle scene.

References

[1] Nelson Max, Roger Crawfis, and Dean Williams, “Visualization for Climate Modeling,” IEEE CG&A Vol. 13 No. 4 (July 1993) pp. 34 - 40.

[2] Peter Williams and Nelson Max, “A Volume Density Optical Model,” Proceedings - 1992 Workshop on Volume Visualization, Boston, October 1992, ACM Order No. 429922.

[3] Nelson Max, Pat Hanrahan, and Roger Crawfis, “Area and volume coherence for efficient visualization of 3D scalar functions,” Computer Graphics Vol. 24 No. 5 (November 1990) pp. 27 - 33.

[4] Nelson Max, “Sorting for Polyhedron Compositing,” in “Focus on Scientific Visualization,” H. Hagen, H. Müller, and G. Nielson, eds., Springer-Verlag, Berlin (1993) pp. 259 - 268.

[5] Peter Williams, “Visibility Ordering Meshed Polyhedra,” ACM Trans. on Graphics Vol. 1 No. 2 (1992) pp. 103 - 126.

[6] Clifford Stein, Barry Becker, and Nelson Max, “Sorting and Hardware Assisted Rendering for Volume Visualization,” Proceedings, 1994 Symposium on Volume Visualization, ACM Siggraph (1994) pp. 83 - 89.

[7] Lee Westover, “Footprint Evaluation for Volume Rendering,” Computer Graphics Vol. 24 No. 4 (August 1990) pp. 367 - 376.

[8] Roger Crawfis, Nelson Max, and Barry Becker, “Vector Field Visualization,” IEEE CG&A Vol. 14 No. 5 (Sept. 1994) pp. 50 - 56.

[9] Geoffrey Gardner, “Visual Simulation of Clouds,” Computer Graphics Vol. 9 No. 3 (Proc. Siggraph 1985) pp. 297 - 303.

[10] Nelson Max and Roger Crawfis, “Visualizing Wind Velocities by Advecting Cloud Textures,” Proc. Visualization ’92, IEEE CS press, Los Alamitos, CA (1992) pp. 179 - 184.

[11] Nelson Max, Barry Becker, and Roger Crawfis, “Flow Volumes for Interactive Vector Field Visualization, Proc. Visualization ’93, IEEE CS press, Los Alamitos, CA (1993) pp. 261 - 265.

[12] Ken Perlin, “An Image Synthesizer”, Computer Graphics Vol. 19 No. 3 (July 1985) pp. 287 - 296.

[13] G. L. Henyey and J. L. Greenstein, “Diffuse Radiation in the Galaxy”, Astrophysical Journal Vol. 88 (1940) pp. 70 - 73.

[14] Michael Cohen, Shenchang Eric Chen, John Wallace, and Donald Greenberg, “A Progressive Refinement Approach to Fast Radiosity Image Generation,” Computer Graphics Vol. 22 No. 4 (August 1988) pp. 75 - 84.

[15] Nelson Max, “Efficient Light Propagation for Multiple Anisotropic Volume Scattering,” in Proc. Fifth Eurographics Workshop on Rendering, June, 1994 (Haas, Müller, Sakas, and Shirley, eds.), pp. 87 - 104.

[16] Michael Garrity, “Ray Tracing Irregular Volume Data,” Computer Graphics vol. 24 No. 5 (November 1990) pp. 35 - 47.

[17] James Spudich, “How Molecular Motors Work,” Nature Vol. 372 (December 8, 1994) pp. 515-518.

[18] Nelson Max, Nobuhiko Hayashi, and Takeyuki Wakabayashi, “Computer Simulation and Animation of Muscle Cross-Bridge Motion”, Visualization and Computer Animation Vol. 1 No. 1 (1990) pp. 9 - 14.

[19] James Blinn, “A Generalization of Algebraic Surface Drawing,” ACM Trans. on Graphics Vol. 1 No. 3 (1982) pp. 235 - 256.

[20] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura, “Object Modelling by Distribution Function and a method of image generation,” (in Japanese) Proc. Electronics Communications Conf. J68-D(4) (1985).

[21] Chikashi Toyoshima and Takeyuki Wakabayashi, “Three Dimensional Image Analysis of the Complex of Thin Filaments and Myosin Molecules from Skeletal Muscles. V. Assignment of Actin in the Actin-Tropomyosin-Myosin Subfragment-1 Complex,” Journal of Biochemistry Vol. 97 (1985) pp. 245 - 263.