Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions

Nelson Max*†
Pat Hanrahan§
Roger Crawfis*

*Lawrence Livermore National Laboratory
Livermore, CA
†University of California
Davis, CA
§Princeton University
Princeton, NJ

Abstract

We present an algorithm for compositing a combination of density clouds and contour surfaces used to represent a scalar function on a 3-D volume subdivided into convex polyhedra. The scalar function is interpolated between values defined at the vertices, and the polyhedra are sorted in depth before compositing. For n tetrahedra comprising a Delaunay triangulation, this sorting can always be done in $O(n)$ time. Since a Delaunay triangulation can be efficiently computed for scattered data points, this provides a method for visualizing such data sets. The integrals for opacity and visible intensity along a ray through a convex polyhedron are computed analytically, and this computation is coherent across the polyhedron's projected area.

CR Categories: I 1.4, I 3.3, I 3.4, I 3.5, I 3.7

Additional Keywords: scalar function, scalar field, volume density, opacity, compositing, volume rendering, visibility sorting, density emitter.

1. Introduction

Three dimensional scalar functions can be visualized either as contour surfaces or as semi-transparent volume density clouds. The function values are traditionally available at vertices on a 3-D cubic or rectangular lattice of voxels.

Contour surfaces can be polygonalized, by cutting a cube with the contour surface [1–4]. The surface normals can be interpolated from estimates of the gradient of the function, and the polygons can then be shaded in a standard graphics pipeline. If several transparent contours are desired, these polygons must be sorted [5], or ray tracing must be used [6].

Ray tracing can also be used to render semi-transparent density clouds, by sampling the scalar function at equally spaced points along the ray (usually by trilinear interpolation of the vertex data), and compositing the color and opacity [6–9]. Alternatively, planes of the lattice can be transformed, filtered, sampled at the image raster, and composited in order into the final image [10]. Contour surface effects may be generated by density classification and gradient shading methods applied directly to the lattice data [8–10], without first generating contour polygons.

There are also many problems where data is not available directly on a cubic lattice. In finite element analysis, more general rectangular, prismatic, or tetrahedral elements may arise. The geometry of a rectilinear lattice or finite element mesh may become distorted during a simulation of elastic or plastic deformation. A rectilinear lattice may be mapped by a non-linear transformation to a curvilinear lattice in order to represent a curved object. Finally, data values may be obtained at scattered 3-D points which have not yet been connected by a mesh of edges. Our algorithm is more general than the algorithms based on lattices, and can deal with these types of irregular data sets. Ray tracing can also be used to render densities in irregular volumes, but can be expensive, and does not take advantage of coherence within the volumes.

Our goal was a method of rendering contour surfaces and density clouds in the same image, which works on arbitrary arrangements of data vertices, and which exploits coherency when coarse data are interpolated. We achieved this by sorting the polyhedra in depth, and then compositing them in depth order, either back to front, or front to back. We have written a subroutine to scan convert and composite the density cloud inside a convex polyhedron, using analytic integration for the color and opacity along the ray projecting to each pixel. We have also written a surface shading, highlight, transparency, and compositing algorithm for polygons. Finally, we have written a procedure to slice a convex polyhedron into other convex polyhedra by the planes defining a contour surface. These are then in turn passed in the correct depth sorted order to the volume compositer. If surface effects are desired at the contour levels, the surface compositer can be called between succesive calls to the volume compositor. By explicitly computing precise contour polygons, we are following a very different philosophy from that in [8–10] where surface effects are computed directly from nearby values of the given function.

Goodsel, et al [6] and Levoy [11] have combined polygonal data and lattice volume data together in one ray traced image. Since our method involves subdividing the

volumes by polygonal surfaces, it can also produce such images.

The scan conversion of a polygon or polyhedron takes advantage of area coherence across its projection. In addition, the analytic integration uses the coherence along the ray segment, so we capitalize on volume coherence throughout each polyhedron. This is much more efficient than tracing each ray separately from one polyhedron to the next, and the analytic integration is more accurate as well as faster than interpolating and compositing multiple samples along a ray. Upson and Keeler [9] discuss a compositing method using coherence across homogeneous segments along the 1-D projection of the 2-D polygon in which a scan plane intersects a lattice cube. However, they still sample along a ray rather than doing analytic integration, so they take advantage of only one of the three dimensions of coherence. Our volume coherence is particularly efficient where the polyhedra involved project to a significant screen area. This occurs, when the viewpoint is close to the object or when the volume subdivision is coarse. However, for regular lattice data in which each voxel projects to only a few pixels, the resampling and compositing method of [10] is faster.

Our technique requires that the input polyhedra be sorted in depth, so that whenever polyhedron A obscures B, B comes later in the sort. Such a sort is trivial for a rectilinear lattice. However, it is possible to fill a volume with a collection of convex polyhedra including three, A, B, and C, such that A obscures B, B obscures C, and C obscures A. If such cycles occur, one of the polyhedra must be subdivided. In the current system, we have not yet written a subdivision scheme, and hope that cycles will not occur in practical data. We will describe a simple algorithm which can detect such cycles if they occur, and sort the polyhedra in time O (n) if they do not. We note that the sort is always possible on the Delaunay triangulation of a set of vertices. Therefore, on scattered data with no other preferred triangulation, we can choose the Delaunay triangulation and be sure that cycles will not occur.

In the next section we describe our model for light propagation through the density clouds, and in the following section, our method of computing analytically or with tables the integrals resulting from the model. In section 4 we describe the algorithm for scan converting a density cloud polyhedron, in section 5, the subdivision at contour surfaces, and in section 6, the rendering of semi-transparent phong-shaded contour polygons. Section 7 describes the depth sorting, and section 8 discusses the resulting images. Two papers by Shirley , et al [12, 13] present similar ideas.

2. Cloud Illumination Models

There have been several computer graphics papers on the scattering, transmission, and shadowing of light propagating through clouds of particles. Kajiya and Von Herzen [14], Rushmeier and Torrence [15], Blinn [16], and Max [17,18] 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. 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 modelling glowing particles, as in the *density emitter* model of Sabella [7]. 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. If D is the total optical density along a ray through an absorbing medium, then the amount of light transmitted is exp(- D) (see Blinn [16]).

Suppose P(t) is a ray from the eye, parameterized by length t, which enters a cloud volume V at P(a), and leaves it at P(b). Then the total optical density of the cloud along the ray from P(a) to P(t) is $\int_a^t \tau\rho(P(u))du$, so light starting from P(t) is attenuated by a factor $\exp\left(- \int_a^t \tau\rho (P(u))du\right)$. The length dt of the ray glows with energy $C\rho(P(t))dt$, so the total glow energy from V reaching the eye is

$$I = \int_a^b C\rho(P(t)) \exp\left(- \int_a^t \tau\rho(P(u))du\right)dt \qquad (1)$$

Also, the light entering the volume at P(b) is attenuated by the factor

$$T = \exp(- D) = \exp\left(- \int_a^b \tau\rho(P(u))du\right).$$

The opacity α, as in [19], is 1-T.

Our algorithm composites the energy I for the polyhedra V back to front by the formula

$$\text{Color} = I + (1-\alpha)\text{Color}$$

and front to back by the pair of formulas

$$\text{Color} = \text{Color} + (1-\text{Alpha})I$$

$$\text{Alpha} = \text{Alpha} + (1-\text{Alpha})\alpha$$

where Color and Alpha are accumulated values of color and opacity. The front to back method requires additional raster memory for the accumulated Alpha. As pointed out by Sabella [7], the integrations give results equivalent to the limit of the compositing schemes of [6], [8–10], when the voxel size or sample spacing approaches zero.

The values of C and τ, and therefore I and α, may be wavelength dependent. In practice, we take C and τ to be vectors with three components, red, green, and blue. In a common model of opaque scattering or emitting particles, τ is independent of wavelength. We treat this case separately, since we can then save on the computation of exponentials.

The modelled density ρ may be set equal to the scalar function f. For greater flexibility in visualization however, we allow ρ to be any non-negative integrable function σ of the scalar function f. For example, if σ is a step function we can render contoured bands of cloud density. Sabella [7] estimated the integral (1) by sampling along a ray, but in fact, its calculation can be reduced to that of the integral of σ, as described in the next section.

3. Calculation of cloud intensity

Suppose we can tabulate or compute analytically the indefinite integral $S(t) = \int_0^t \sigma(u)du$. Suppose also that f varies linearly along the ray from A=P(a) to B=P(b) in the volume V. This is the case in our implementation since we interpolate f trilinearly across regions of the polyhedron V, in a 3-D analogue of the bilinear interpolation used for Gouraud shading. Under these assumptions we may write

$$f(P(u)) = f(A) + (f(B) - f(A))\,(u-a)/(b-a)$$
$$= gu+h$$

where $g = (f(B)-f(A))/(b-a)$ and $h = f(A) - ga$. Then the integral for optical density is

$$D = \int_a^b \tau\rho(P(u))du = \tau\int_a^b \sigma(f(P(u))du = \tau\int_a^b \sigma(gu+h)du$$
$$= \frac{\tau}{g}\int_{ga+h}^{gb+h} \sigma(v)\,dv = \frac{\tau}{g}\,(S(gb+h) - S(ga+h))$$
$$= \frac{\tau}{g}\,(S(f(B)) - S(f(A)))$$

Also, by the chain rule,

$$\frac{d}{dt}\exp(-\int_a^t \tau\rho(P(u))du)$$
$$= \frac{d}{dt}(-\int_a^t \tau\rho(P(u))du)\exp(-\int_a^t \tau\rho(P(u))du)$$
$$= -\tau\rho(P(t))\exp(-\int_a^t \tau\rho(P(u))du)$$

so the integral of equation (1) simplifies as

$$I = \int_a^b C\rho(P(t))\exp(-\int_a^t \tau\rho(P(u))du)dt$$
$$= -\frac{C}{\tau}\int_a^b(-\tau\rho(P(t))\exp(-\int_a^t \tau\rho(P(u)))du)dt$$
$$= -\frac{C}{\tau}\int_a^b \frac{d}{dt}\exp(-\int_a^t \tau\rho(P(u))du)dt$$
$$= \frac{C}{\tau}\,(1 - \exp(-\int_a^b \tau\rho(P(u))du))$$
$$= \frac{C}{\tau}\,(1 - \exp(-\frac{\tau}{g}(S(f(B)) - S(f(A))))).$$

Thus,

$$\alpha = 1 - T = 1 - \exp(-D)$$
$$= 1 - \exp(-\frac{\tau}{g}(S(f(B)) - S(f(A)))) \qquad (2)$$

and

$$I = \frac{C\alpha}{\tau}. \qquad (3)$$

Note that the exponential function needs to be evaluated only once to obtain I and α, when the color C is wavelength dependant but the opacity τ is not.

In order for the integrals to take this simple form, C and τ must be constants, and cannot be interpolated across the polyhedra. If we wish different C and τ values at different ranges of f values, we must subdivide the polyhedra at the contours of f dividing these ranges. This is in contrast to [10], where a voxel may contain a mixture of materials, and thus have a weighted average of material properties like C and τ. To allow such mixtures, we have an option which interpolates C and τ, and approximates the correct integral by using the average values of C and τ along a ray in formulas (2) and (3).

In our first implementation, we defined σ to be linear over succesive intervals of f values, allowing discontinuities between intervals. We subdivided the polyhedra at the breaks between the f intervals. This is not really necessary since formulas (2) and (3) work for general σ, but it simplifies the calculation of $\int_a^b \sigma(f(P(u)))du$ to the trapezoid area formula $.5(b-a)(\sigma(f(B)) + \sigma(f(A)))$. In practice, contour surfaces, contours of changing color C, contours of changing τ, and contours between stretches where σ is linear, are all taken to coincide, so the subdivision of polyhedra does not become excessive.

4. Scan converting and shading polyhedra

Consider a convex polyhedron V in screen coordinates, projected orthogonally onto a convex region R in the (x,y) plane. The region R is covered twice by faces of V: once by those facing the viewer, and once by those facing away. If we scan convert R twice, linearly interpolating z and the scalar function f for the front and back surfaces of V, we have the data necessary to compute I and α as in the previous section. This is accomplished by a standard scan line algorithm, using y buckets for the edges, and two independent x sorts of the scan line intersections of the edges, from the front and back polygonal networks. The algorithm is further optimized to take advantage of the simplicity in the projection of a convex polyhedron.

For each polygon, all the edges are entered into y buckets at the first scan line they intersect, and these are used to update both x-sorted lists of edges from one scan line to the next. Profile edges must be entered into both lists. Since each network covers R exactly once, the edges of a given network never cross each other. Therefore if the edges are entered correctly from the y buckets into the x-sorted lists, the sort need never be checked thereafter.

The x, z, and f values are maintained incrementally for each edge from one scan line to the next. Then, separately for the front and back networks, z and f are interpolated across the intervals between edge intersections, and saved in separate pairs of scan line buffers. The integrals (2) and (3) for I and α are then computed from these buffers, giving coherence over the whole projection of the region. The relation between the front and back surfaces only comes into play in this last integration step. There is never any need to subdivide into smaller homogeneous spans corresponding to the same front- and back-facing polygons. Once I and α are determined, the

scan line of the polyhedron is composited onto the previous image.

A less accurate but much faster alternative is to interpolate I and α directly across the homogeneous regions onto which the same front- and back-facing polygons project. Since I and α are zero along the profile, they need only be computed at the intersections of edge projections in the interior of R. This scheme is particularly advantageous on systems like the IRIS 220 GTX, which have hardware assist for linearly interpolating and compositing "r g b α" images across polygons. However, it less accurate, because I and α are non-linear functions of the cloud volume thickness along a ray. Perhaps quadratic or cubic interpolation across Bezier triangles [20] would provide superior results.

For perspective projections, the vertices are first transformed into *screen* coordinates from which they can be projected orthogonally. This transformation maps planes into planes [21], so the z values can be linearly interpolated. To get the world coordinate length inside a volume region, the screen values are transformed back into world coordinates.

The trilinear interpolation of f described above is ad-hoc, and depends on the orientation of V and its projection onto the picture plane. For a general convex polyhedron, there is no obvious way to interpolate values of f given at the vertices. However, if V is a tetrahedron, there is a unique linear function, g, independent of orientation, which matches the values of f at the vertices. To see this, write $g(x,y,z) = ax+by+cz+d$. The specified data values at the four vertices give four linear equations of the four unknowns; a, b, c and d. Since the four vertices of a non-degenerate tetrahedra do not lie on the same plane, these four equations have a unique solution.

If W is any subvolume of V, obtained by slicing with contour planes and defining new vertex values by interpolation across sliced edges, then the trilinear interpolation on W described above for orthogonal projection will agree with g restricted to W. But the transformations for perspective are non-linear, and destroy this property. Therefore, we allow the user to specify a more accurate perspective option, which interpolates f in world coordinates, instead of in screen coordinates.

5. Subdivision of Polyhedra About Isocontours

To enhance the visualization, sharp changes in the properties of the clouds are desired at user specified thresholds. Thus, we allow the specification of contour values about which the polyhedra can be divided and given differing optical properties. For generality and topological consistency (i.e., no surface holes between adjacent polyhedra), we have chosen to subdivide any polyhedron known to contain a surface corresponding to an isocontour into tetrahedra. We have seen that a linear function on a tetrahedron is uniquely defined by function values on its vertices. Thus any contour surface through a tetrahedron is a planar surface, and does not need to be subdivided into triangles. Since multiple contour planes crossing a tetrahedron are all parallel, it is simple to determine the order in which the various volume and surface slices should be passed to the rendering routines.

We can use this technique directly on triangulations. For more general polyhedral subdivisions, we divide the cells containing contours into tetrahedra. For each polyhedron, a quick test is done to determine whether any of the contour surfaces pass through it. If not, the polyhedron is simply passed on to the renderer. If a polyhedron contains a contour surface, the polyhedron is split into tetrahedra and the tetrahedra are passed to the contouring algorithm for tetrahedra in back to front order.

For rectilinear and curvilinear meshes, we use an odd/even scheme for splitting a cube into tetrahedra. In this scheme, cubes are divided into five tetrahedra. There is a central regular tetrahedron spanning the four cube vertices where the sum of the lattice indices is even, and four other tetrahedra joining a face of the central one to the other four vertices with odd index sums. Each cube face is subdivided by a diagonal joining its two even vertices. This produces two differently oriented subdivisions on the odd and even cubes and assures consistency between adjacent cubes. Thus contour surfaces defined on two adjacent subdivided cubes will meet continuously without holes.

Other methods of splitting the polyhedra are possible. One such method would be to use a marching cubes algorithm [1] to determine a triangulated contour surface that passes through a cube. This triangulated surface may split the polyhedron into non-convex pieces, but our sorting and compositing scheme requires that the polyhedra be convex. Therefore, the volumes must be split successively at the plane of each contour triangle into smaller convex pieces. This algorithm has the advantage of generating a small number of polyhedra for the most common cases. For multiple contour surfaces, it is possible for the slice planes defined for one contour to intersect those defined for another. This may produce many small volumes and contour surface pieces which must be processed further to determine the appropriate optical properties.

6. Shading contour polygons

The contour polygons to be shaded are scan converted with the standard sort of scan-line algorithm as described above, with the unit normals at the vertices also bilinearly interpolated across the polygon and renormalized for shading. For rectangular lattices, the normal for a contour vertex can be interpolated from the gradients of f at lattice vertices. These gradients can in turn be estimated from finite differences of the values of f at adjacent vertices. For subdivisions into tetrahedra, we can estimate the gradient at a vertex by averaging the gradients for the tetrahedra meeting at the vertex. In section 4, we showed how to define a unique linear function $g=ax+by+cz+d$, with gradient (a,b,c), matching the values of f at a tetrahedron's vertices.

We bilinearly interpolate the normal vectors given at the vertices and then renormalize to a unit vector, but have two options for the calculation of the highlights. Let N be the normal vector at a pixel, and H be the unit vector halfway between the viewing direction V and the lighting direction L. Then the first option computes the highlight intensity from the dot product of H and N. This method is particularly fast when H is a constant, which is the case when the viewpoint and light are both at infinity. We always assume the light is at infinity, but even for a perspective view with a finite

viewpoint, we offer a fast shading option assuming V is constant. This assumption would result in constant highlights across flat polygons. For more accurate highlights we have a second option, which computes the vector V separately for each pixel. In this case, if there are multiple light source directions, L_i, it is faster to compute a single vector R in the direction at which a flat mirror normal to N would reflect the ray along V, and find the highlight for each light source from the dot products of R and L_i.

For semi-transparent glass appearance on the contour surface, we use the same transparency formulation as for the cloud density. The length that a ray of unit direction V spends inside a glass surface with normal N and thickness t is $t/(V \cdot N)$. If the glass has optical density τ per unit length, the transparency is $\exp(- \tau t/(V \cdot N))$. This becomes zero when V and N are perpendicular, at the surface profile. Since we do not subdivide to get an accurate profile, the transparency could instead become zero at a line in the interior of a polygon when some of the vertex normals point towards the eye, and some point away. Therefore, we do not use the interpolated normals in this computation, but instead interpolate the transparency computed at the vertices. This also saves computing an exponential, dot product, and divide at every pixel.

7. Depth-Sorting Polyhedral Subdivisions

List-priority algorithms have been used for hidden surface removal since the '60's (Schumacher [22], Newell et al. [5]). The basic idea is to draw filled polygons from back to front into a framebuffer so that the front ones overwrite the ones behind them. Newell's algorithm determined the priority order on a frame by frame basis by comparing the overlap of polygons from a given point of view. Schumacher's algorithm precomputed a priority graph. This graph was then topologically sorted and every polygon given a number. The sort was such that once back-facing polygons were removed, the remaining polygons were always ordered from back to front based on the preassigned numbers. Schumacher's algorithm also used separating planes to partition convex clusters. This is a forerunner of binary space partition (BSP) trees [23]. All these methods allow a collection of polygons to be quickly sorted from back to front given any viewpoint.

Most modern graphics hardware performs hidden surface elimination using a z-buffer so these algorithms are not commonly used for that task. However, if polygons are to be drawn with transparency, the list priority algorithms are still the best known way to draw the scene because transparency calculations can be reduced to compositing operations, if the surfaces are drawn in the proper order.

We are interested in sorting a cell subdivision in three dimensions in back to front order or front to back order for compositing. For simplicity, we describe below an algorithm giving a front to back order. Each cell is assumed to be a convex polyhedron with planar faces. We will assume that the face adjacencies are given. That is, given a polyhedron and one of its faces, we can find the polyhedron opposite the face in constant time. Assume also that the plane equation of the face is available. For simplicity, we will also assume the subdivision is space-filling so there are no holes. This means that if one cell hides another, there is a sequence of pairwise adjacent cells connecting them, so that each cell is in front of the next across a common face.

To sort the cells in front to back order we first determine relative priorities of adjacent polyhedra. This is done by comparing the viewpoint to the plane of the face dividing a pair of adjacent polyhedra. Since the polyhedra are convex, this plane separates the polyhedra — that is, they must lie entirely on opposite sides of the plane. The frontmost polyhedron is the polyhedron that is on the same side of the plane as the viewpoint. Relative priorities can be constructed in time proportional to the total number of faces. These relative priorities are stored in a directed graph, with a directed edge from A to B if A and B share a common face and A is in front of B. Note that the priority graph changes whenever the viewpoint changes.

The second step topologically sorts the priority graph into a single priority list ordered from front to back. The topological sort will always succeed as long as the directed graph does not contain cycles. Such cycles would correspond to a cycle of polyhedra, each of which is in front of and adjacent to the next. Furthermore, the sort can be performed in time proportional to the number of edges k in the graph (which equals the number of faces in the subdivision). As described in Knuth [24], topological sorting can be done in linear time by keeping track of the number of active incoming edges at every vertex in the graph. All the vertices with no incoming edges are placed in a queue. Vertices are successively removed from this queue and placed on the priority sorted list. When they are removed, all their neighbors along outgoing edges are examined. For each neighbor the count of active incoming edges is decremented, and if the count goes to zero, then all the neighbors which have priority over it have been output, so it is placed on the queue to be output. If the queue becomes empty before all polygons are listed, there must be a cycle in the graph. Otherwise, the algorithm will determine a priority sort in O(k) time.

A special case of the general space subdivision is a triangulation. In 3D, a triangulation is a decomposition of space into a set of tetrahedra. Tetrahedral subdivisions are nice because the number of face adjacencies is always 4 and, as mentioned previously, they allow well-defined interpolation formula.

A special triangulation is the Delaunay triangulation. This is the dual of a decomposition of space into Voronoi polyhedra. The Voronoi diagram is defined given a set of points in space. Each point is surrounded by a polyhedron defined as the locus of points which are nearer that point than any other point in the set. The union of all these polyhedra are the Voronoi diagram [25,26]. The vertices of the dual – the Delaunay triangulation – are the original set of points, and a tetrahedron joining four points in the set belongs to the triangulation only if the sphere circumscribed about the four points contains no other point in the set. Delaunay triangulations can be computed in $O(n \log n + k)$ time [26], where n is the number of vertices, and k is the number of faces of the triangulation. The tetrahedra tend to be well-behaved and not long and skinny. Thus, Delaunay triangulations are conveniently generated from point sets and these triangulations are useful grids for finite element calculations, or when scattered data points are given without any other preferred grid.

Delaunay triangulations also have another remarkable property recently proved by Edelsbrunner [27]: "Given a viewpoint, the tetrahedra comprising a Delaunay triangulation can always be sorted in back to front order." This property makes these triangulations a useful representation for volume rendering, since the topological sort described above will never find cycles.

Edelsbrunner and Barry Joe [personal communication] have recently simplified the proof of this fact, using a function g(V,A) of a 3-D point V and a tetrahedron A in the Delaunay triangulation, defined as follows. Let the sphere S(A) circumscribed about the tetrahedron A have center C(A) and radius r, let d be the distance from V to C(A), and let $g(V,A) = d^2 - r^2$. If V is outside of S(A), g(V,A) is the square of the length of the tangent from V to S(A).

We show below that if A and B are two adjacent tetrahedra, and V is on the same side of the plane of their common face as A, then g(V,A) < g(V,B). If we take V to be the viewpoint, this ordering relation can be used to build the directed graph described above, and also to prove there are no cycles. For if there were a cycle of adjacent tetrahedra, A, B, … Z, each in front of and adjacent to the following one, then g(A,V) < g(B,V) < … < g(Z,V) < g(A,V), which is a contradiction.

To prove this ordering property, consider the difference h(V,A,B) = g(V,A) - g(V,B). Since g(V,A) and g(V,B) have the same quadratic terms, this is a linear function in the coordinates of V. Also, if V lies on the plane P of the common face, h(V,A,B) = 0, since a tangent from V in this plane to the circle circumscribed about the common face is also tangent to both spheres S(A) and S(B).

To show that h(V,A,B) is positive when V is on the "B side" of plane P, consider the vertex D of B which is not on plane P. Since D is on sphere S(B), g(V,B) = 0. By the defining property of the Delaunay tetrahedron A, however, D must be outside the sphere S(A). Therefore g(D,A) > 0, so h(D,A,B) > 0.

8. Results

Figure 1 shows the effect of the scan converting on a test problem defined as a decreasing function radiating from a circle in the center of the data set. A torus-like cloud can be seen, when this function is mapped to density. Figure 2 shows the same test data with a sharp contour introduced. Values above the contour value are less opaque than the values below the contour value, and different colors are given to the clouds above and below the contour surface. With Figure 3, we add a specular-reflective surface to the image. The surface is partially opaque and the decreasing scalar field can be seen behind it. All of these images were generated from a small data set of 4096 cubes using one CPU of a Stellar 2000 at a resolution of 500 by 500 pixels. The computation and compositing of the color and opacity were vectorized over scan lines. The approximate times to generate the images are respectively, 24, 42, and 48 CPU seconds. As can be seen, the cost of contouring is substantial due to the number of additional polyhedra generated and the loss of coherency.

Figure 4 shows an image of the test data with two contour surfaces, one partially opaque and one completely opaque, against a checker board background. Figure 5 shows two isocontours of equal density for a super-deformed state in the nucleus of Th^{238}. The data for this image was generated on the Cray XMP supercomputer using a Hartree-Fack calculation by Cecill Eggens and Mort Weiss. This image took two minutes to generate. The data sets were generated on a 17x17x27 regular grid, requiring several hours of Cray X/MP time.

Figures 7 shows a Hipip (high potential iron protein) molecule from last years data set tape. The data is defined on a 64x64x64 grid courtesy of Louis Noodleman and David Case at the Scripps Clinic. Here we have zoomed in on two clusters of iron chains. Figures 8 and 9 show two views of the electron density of superoxide dismutase. One uses a color and opacity mapping to change the colors without doing contouring, while Figure 9 adds explicit contouring and light surfaces. The data is courtesy of Duncan McRee of the Scripps Clinic.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. Support for this research came from the Institution Research and Development funds at Livermore, and from NFS grant No. DMS–861562–01 at the University of Minnesota, Geometry Supercomputer Project .

The authors would like to thank Michael Laszlo, John Sullivan, and Allan Wilks for their software to construct Delaunay triangulations in 3D and for Anna Farr for her work in getting that software to work in our environment. Thanks to David Dobkin and Bernard Chazelle for pointing out Edelsbrunner's paper on sorting triangulations; and to Herbert Edelsbrunner for discussions about his result. We would also like to thank Jeff Kallman, Jane Wilhelms, and Alan Van Gelder for useful conversations, Charles Grant and Anna Farr for proofreading, Gerri Braswell for typing, and Robert Shectman and LCC for last minute help in getting final images developed.

References

[1] Lorenson, William E., and Cline, Harvey E., *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. Computer Graphics Vol. 21 No. 4 (July 1987, Siggraph '87 Proceedings) pp. 163-169

[2] Wyvill, Geoff, McPheeters, C. and Wyvill, Brian, *Data Structures for Soft Objects.* The Visual Computer Vol. 2 No. 4 (1986) pp. 227-234

[3] Bloomenthal, Jules, *Polygonization of implicit surfaces.* Computer Aided Geometic Design Vol. 5 (1988) pp. 341-355

[4] Bloomenthal, Jules and Wyvill, Brian. *Interactive Techniques for Implicit Modeling.* Computer Graphics Vol. 24 No. 2 (March 1990, Proceedings 1990 Symposium on Interactive 3D Graphics) pp. 109-116

[5] Newell, M.E., Newell, R.G., and Sancha, T.L., *A New Approach to the Shaded Picture Problem.* Proceedings of the ACM National Conference (1972) pp. 443-450

[6] Goodsell, David S., Mian, I. Saira, and Olson, Arthur J. *Rendering volumetric data in molecular systems*. Journal of Molecular Graphics Vol. 7 No. 1 (March 1989) pp. 41-47

[7] Sabella, Paolo, *A Rendering Algorithm for Visualizing 3D Scalar Fields.* Computer Graphics Vol. 22 No. 4 (August 1988, Siggraph '88 Proceedings) pp. 51-55

[8] Levoy, Marc, *Display of Surfaces from Volume Data.* IEEE Computer Gaphics and Applications Vol. 8 No. 3 (May 1988) pp. 29-37

[9] Upson, Craig and Keeler, Michael, *V - BUFFER: Visible Volume Rendering.* Computer Graphics Vol. 22 No. 4 (August 1988, Siggraph '88 Proceedings) pp. 59-64

[10] Dreben, Robert A., Carpenter, Loren, and Hanrahan, Pat, *Volume Rendering.* Computer Graphics Vol. 22 No. 4 (August 1988, Siggraph '88 Proceedings) pp. 65-74

[11] Levoy, Marc, *A Hybrid Ray Tracer for Rendering Polygon and Volume Data.* IEEE CG&A, Vol 10 No. 2 (March 1990) pp. 33-40

[12] Shirley, Peter and Tuchman, Alan, *A Polygonal Approximation to Direct Scalar Volume Rendering*, (in this issue)

[13] Williams, Peter L. and Shirley, Peter, *An A Priori Depth Ordering Algorithm for Meshed Polyhedra*, CSRD Technical Report #1018, University of Illinios, Champaign-Urbana (1990)

[14] Kajiya, James T. and Von Herzen, Brian P., *Ray Tracing Volume Densities*. Computer Graphics Vol. 18 No. 3 (July 1984, Siggraph '84 Proceedings) pp. 165-174

[15] Rushmeier, Holly E. and Torrance, Kenneth E., *The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium.* Computer Graphics Vol. 21 No. 4 (July 1987, Siggraph '87 Proceedings) pp. 293-302

[16] Blinn, James, *Light Reflection Functions for Simulation of Clouds and Dusty Surfaces.* Computer Graphics Vol. 16 No. 3 (July 1982, Siggraph '82 Proceedings) pp. 21-29

[17] Max, Nelson, *Atmospheric Illumination and Shadows.* Computer Graphics Vol. 20 No. 4 (August 1986, Siggraph '86 Proceedings) pp. 117-124

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

[19] Porter, Thomas and Duff, Tom, *Compositing Digital Images*. Computer Graphics Vol. 18 No. 3 (July 1984, Siggraph '84 Proceedings) pp. 253-259

[20] Max, Nelson, *Smooth Appearance for Polygonal Surfaces.* The Visual Computer Vol. 5 No. 3 (June 1989) pp. 160-173

[21] Newmann, William M., and Sproull, Robert F., *Principles of Interactive Computer Graphics, Second Edition*. McGraw Hill, New York (1979) pp. 361-363

[22] Schumacher, K. A., Brand, R., Gilliland, A.M., and Sharp, A.W., *Study for Applying Computer Generated Images for Visual Simulation*. U.S. Air Force Human Resource Laboratory Technical Report AFHRL - TR - 69-14 (1969)

[23] Fuchs, Henry, Kedem, Zvi M., and Naylor, Bruce F., *On Visible Surface Generation by A - priori Tree Structures.* Computer Graphics Vol. 14 No. 3 (July 1980, Siggraph '80 Proceedings) pp. 124 - 133

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

[25] Preparata, Franco P. and Shamos, Michael I, *Computational Geometry: An Introduction.* Springer Verlag, New York (1985)

[26] Edelsbrunner, Herbert, *Algorithms in Computational Geometry*. Springer-Verlag, Heidelburg (1987)

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

Figure 1. Four views of sample torus test data: no contouring, contouring, light shading, and culling.

Figure 2. Test data with two contours and three differrent cloud properties.

Figure 3. Proton density of Th^{238}. Data courtesy of Cecill Eggens and Mort Weiss, LLNL.

Figure 4. HIPIP electron orbital calculation. Data courtesy of Louis Noodleman and David Case, Scripps Clinic.

Figure 5. X-ray crystallography of SOD enzyme. Data courtesy of Duncan McRee, Scripps Clinic.

Figure 6. SOD enzyme with light shaded isocontours.