Volume Probes: Interactive Data Exploration on Arbitrary Grids

Don Speray
The University of Texas at Austin, Dept. of Computer Sciences
The Computational Mechanics Company, Inc.

Steve Kennon
The Computational Mechanics Company, Inc.

Abstract

A taxonomy of computational grids used in scientific and engineering practice is presented and a technique for cutting them by, and displaying data on, 2D surfaces is developed. When sliced by a surface, these grids give rise to a graph G(C,F) where C, the nodes, are the intersected cells and F, the arcs, are their connectivity across faces. Starting from any cell known to be intersected by the surface (a seed), G is traversed breadth-first and is constructed locally on the fly, that is, only the spreading "front" explicitly exists at any time. Only sliced cells are visited, shared computed values such as edge intersections are passed to neighbors, and most of the geometric work is done via table lookup. A seed cell is found by fence-hopping from any cell to a distinguished point on the surface.

This means of slicing grids is then utilized in an effective visualization tool. Concentrating on planar surfaces, local coordinate systems are defined for constructing clipping windows and linear transformations within the planes which further reduces display time and allows effects such as zooming within the windows. Several of these planar windows are then organized into various objects, called probes, that can exploit the mind's "retinal memory" when repeatedly swept through amorphous data.

CR Categories and Subject Descriptors: **I.3.3** [**Computer Graphics**]: Picture/Image Generation – *Display algorithms*; **I.3.5** [**Computer Graphics**]: Computational Geometry and Object Modeling – *Geometric algorithms*; **I.3.6** [**Computer Graphics**]: Methodology and Techniques – *Interaction Techniques*

Additional Keywords: slicing, post–processing, probes, retinal memory

Introduction

Slicing will remain one of the fastest volume display methods for scalar fields for the same reason that it complements other common techniques. It shows a limited (one might say infinitesimal) subset of the volume per slice and displays the full range of data existing there. Contrast this with contouring or ray casting where a limited range of data is shown throughout the full volume. These techniques benefit from *a priori* knowledge of the data for selecting contour or opacity values, say, to arrive at the desired image. Since slices can be generated quickly, data exploration with slicing would be a useful first step in many analysis regimens.

In many applications, however, these other methods are inadequate or inappropriate. Choosing a set of distinguished values to display may be undesirable and it doesn't take many nested colored transparent surfaces before confusion ensues. They also suffer when noise or high frequency behavior is present in the data near values of interest, showing up as "fragmentation" (floating gravel), confusing surface classification algorithms, or simply taking longer to render. Acquired data is rarely smooth and computational results are seldom as smooth as one wishes (especially in modeling turbulence [7] !). Slicing is less susceptible, but not immune, to these problems. It is hard to imagine comprehensible 3D versions of the gas jet displays of Norman [10] and Winkler [9] that attempt to convey an equivalent level of information.

Another issue is the time it takes to ray-cast a volume, or trace out a complete contour and photorealistically render the result. There is a place in a researcher's armamentarium for simple and fast display, especially in light of the fact that these methods may be slower in the context of the grids to be discussed.

Finally, *planar* slices open up volume data analysis to researchers without late-model workstations. The results may be displayed in 2D with line countours or filled polygons, perhaps accompanied with a low bandwidth 3D display to orient the slice in the volume.

We further believe that slicing can be effective in showing total data behavior when several slices sweep the volume in concert and quickly enough to build and reinforce a mental image. We begin by discussing the grids in use today, follow with an efficient algorithm for slicing them, and end with an application of the algorithm that satisfies most of the issues raised.

Computational Grids for Graphicists

The following taxonomy of grids will suffice for our discussion; see, for example, [1, 11, 14] for more information. They are presented in order of increasing generality (and complexity). With each type is the required indexing to find a point's world coordinates. Neighboring points delineate subvolumes known as cells or elements.

cartesian (i, j, k)

This is typically a 3D matrix with no intended world coordinates, so subscripts map identically to space. If the cells are small and numerous (as to be *almost* atomic in practice, like 2D pixels), then it is known as a **voxel** grid; however, the term is often loosely applied.

regular $(i * dx, j * dy, k * dz)$

Cells are identical rectangular prisms (bricks) aligned with the axes.

rectilinear $(x[i], y[j], z[k])$

Distances between points along an axis are arbitrary. Cells are still rectangular prisms and axis-aligned. See figure 1 for a simple 2D example.

structured $(x[i, j, k], y[i, j, k], z[i, j, k])$

This type, also known as **curvilinear**, allows non-boxy volumes to be gridded. Logically, it is a cartesian grid which is subjected to non-linear transformations so as to fill a volume or wrap around an object. Cells are hexahedra (warped bricks). These grids are commonly used in computational fluid dynamics (CFD). See figure 2.

block structured $(x_b[i, j, k], y_b[i, j, k], z_b[i, j, k])$

Recognizing the convenience of structured grids, but the limited range of topologies that they handle, researchers may choose to use several structured grids (blocks) and sew them together to fill the volume of interest.

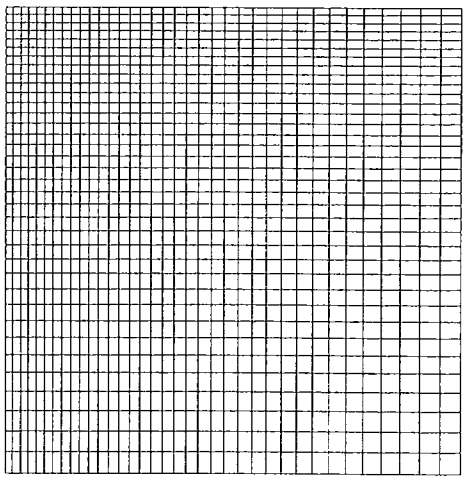

Figure 1: Rectilinear

unstructured $(x[i], y[i], z[i])$

Unlike the previous types, where connectivity is implicit, there is no geometric information implied by this list of points and edge/face/cell connectivity must be supplied in some form. Cells may be tetrahedra, hexahedra, prisms, pyramids, etc., and they may be linear (straight edges, planar faces) or higher-order (eg. cubic edges, with two interior points on each edge). Tetrahedral grids are particularly useful because they allow better boundary fitting, can be built automatically, and are often simpler to work with, graphically. Unstructured grids are standard in finite-element (FEA) and finite-volume analysis (FVA) and are becoming common in CFD. See figure 3.

hybrid

It may occasionally be desirable to use structured and unstructured grids together, putting each where their fitting and computational strengths are most beneficial. See figure 4, but note that the structured layer is exaggerated. At this scale, it would appear as a thick black line.

In addition to the three coordinates, there may be several physical quantities computed at each point. The size of grids can range from a few thousand elements for simple FEA problems to a several million for complex CFD studies and the trend in CFD is to use grids that challenge each new generation of supercomputer.

Computational grids are designed to minimize numerical error and this usually means many small cells are located where "interesting things" happen in the volume. Unfortunately, a fixed grid must anticipate where these occur throughout the duration of a time-varying phenomenon and the result is a dense grid requiring more

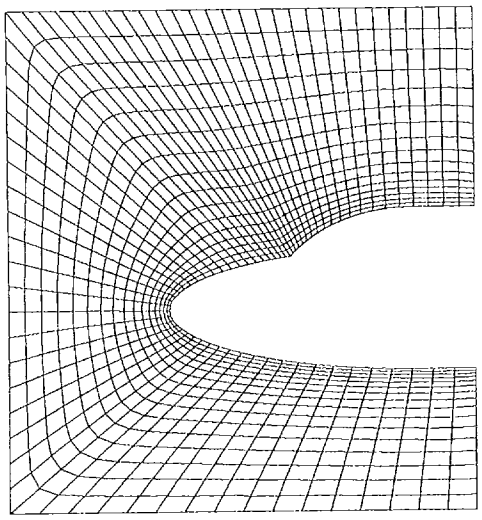

Figure 2: Structured

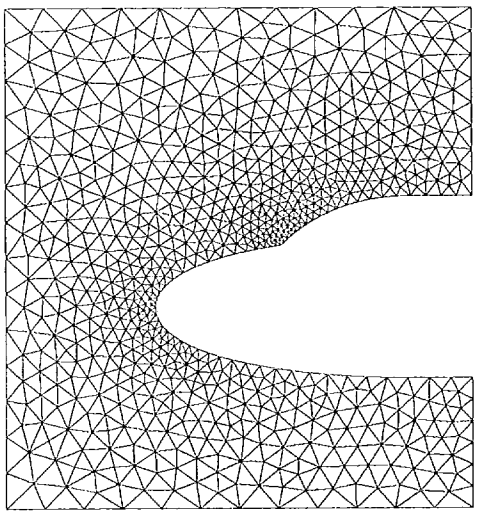

Figure 3: Unstructured

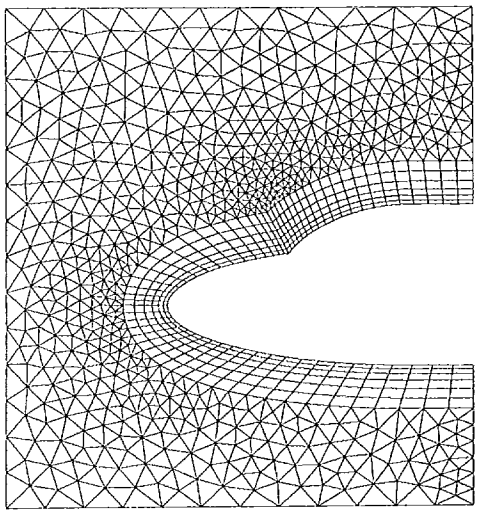

Figure 4: Hybrid

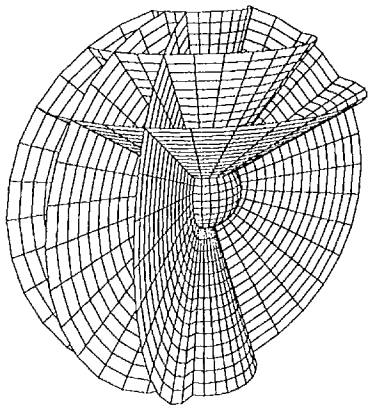

Figure 5: Simple 3D structured grid

computation than might be necessary at any given time step. Consequently, adaptive refinement of grids is becoming more popular.

Another means to reduce error in unstructured grids is to use higher-order isoparametric elements. These shape functions and cell geometries also permit larger, hence fewer cells. An analysis code using them would supply the functions for performing interior interpolation, which involves solving several simultaneous nonlinear equations. It is reasonable to place the burden of such interpolation on the analysis code, which could refine an element into a number of smaller linear elements for display, or provide a simple mechanism to do so, and we will assume this. Incidentally, iterative techniques are used for this, even for the trilinear case. For rectilinear grids trilinear interpolation is equivalent to seven linear interpolations, whereas this is no longer true for general hexahedra, and iterative methods leverage the work to compute one point for quick convergence at nearby points.

In CFD, grids extend beyond the region of study solely to eliminate boundary-induced numerical artifacts in the results, so much of the volume may be of no interest at all, though general features are grasped by seeing its entirety. On the other hand, quite dramatic behavior can occur in small, tightly meshed regions. This grid-driven changing scale of focus is characteristic of interactive CFD graphics [12].

The 3D grid in figure 5 is used for later illustration of various aspects of the slicing algorithm. This is a structured spherical coordinate grid over a hemisphere with latitudes near both poles missing. It contains hexahedral cells and has a non-convex boundary consisting of the inner sphere, the polar cones, two "wings" of constant longitude, and an outer sphere which is not drawn. A few inner grid surfaces of constant longitude and constant latitude are shown.

It is the intention of this section to raise the literacy of the graphics community about what is encountered in scientific and engineering practice. Only the final four grid types handle arbitrary volumes and are the most in need of interactive display techniques. It is hoped that more graphics researchers will find computational fluid dynamics and finite-element analysis to be sources of interesting challenges. A source of ideas in CFD might be further generalizations and idealizations, as are particle traces and flow ribbons, of what is actually done in windtunnels [8].

Slicing

Slicing amounts to identifying intersected cells and displaying data within the region of intersection of each with the surface. Our goal is to slice any grid with user-defined surfaces at any position and orientation in the volume.

The orderliness of a grid and the shape of the surface, of course, have an impact on finding intersected cells. This is important since relatively few cells are actually cut. For example, a voxel grid sliced by the plane $ax + by + cz + d = 0$ has sliced cells with the following points as vertices:

$$(i, j, \lfloor -(ai + bj + d)/c \rfloor),\ (i, j, \lceil -(ai + bj + d)/c \rceil)$$

This is an over-simplified characterization of sliced cells, but it can be seen that speed is achieved by directly computing cell locations and by-passing a search.

With a structured grid it is common to fix one subscript and display the resulting surface, allowing for intermediate positions by using a non-integral "subscript" and interpolating. This is fast because it slices the 3D array (in IJK space) with a plane parallel to one of the subscript planes. Of course, the shape and position of the slice follow the grid in world coordinates, as with the inner surfaces in figure 5.

Others have sliced unstructured grids but have used exhaustive search of the grid for intersected cells [4, 5, 13]. The present technique avoids a search by utilizing cell connectivities and the continuity of the surface to follow the surface through intersected cells. The emphasis in this paper is on the issues of determining the intersections and not on how one might display the results.

A cutting surface may be either open or closed, dividing space into two parts. Let the surface be defined by the characteristic function

$$S(x, y, z) = \begin{cases} 1 & \text{if } (x, y, z) \text{ is in the positive half space} \\ 0 & \text{otherwise} \end{cases}$$

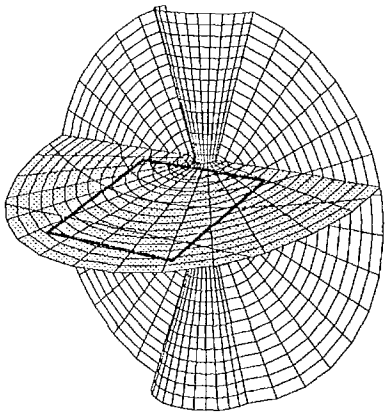

Figure 6: Surface polygons

Surfaces are assumed to be approximately planar within cells for the purposes of determining edge and face intersections, that is, faces may be cut only once. This assumption also places a requirement on cell faces: ideally, faces would be planar, but are often only approximately so. In practice, nearly-planar faces have not been a problem; further cell refinement is possible, perhaps at the cost of interior interpolation.

As in marching cubes [6], each vertex of a cell is assigned a bit position within an index and $S(x, y, z)$ provides the bit value. This index identifies, in a precomputed table, the intersected edges and faces in polygon order. In our implementations, we deal only with tetrahedral and hexahedral cells. The table of 14 cases for tetrahedra was built by hand. The hexahedral table was built by software that inspected each of the 254 candidate cases by walking across faces, following edge intersections, until an impossible or disallowed situation was encountered. There are 63 unique valid configurations and each appears twice in the table because flipping bits makes no difference. The table was verified by using the Ho-Kashyap procedure [2] to linearly separate the vertices of a cube and by checking the twelve extra cases, due to twisting, by hand. These tables could be used to verify the integrity of a grid by sweeping it with a plane to cut cells and seek impossible vertex configurations.

Given a sliced cell, its table entry lists the faces to move through to follow the surface. Except for the first "seed" cell, then, the search is table-driven. The following section will deal with finding this seed.

Intuitively, a cutting surface is covered by a network of polygons resulting from being cut by cell faces. An example is figure 6, which shows a nearly horizontal plane cutting the sample grid. (Ignore the heavy rectangle, for now.) The dual graph of the network is G(C,F) where C, the nodes, is the set of polygons (intersected cells, in 3D) and F, the arcs, are their connectivities across edges (cell faces, in 3D). A breadth-first traversal of G, start-

ing from the seed, is appealing for three reasons. First, only the nodes on the traversal front need to be known so G may be built locally and on the fly. Second, it allows maximum sharing of computed results across faces since a cell's "more distant" neighbors are all added to the front at the same time. Finally, rules to terminate the traversal, such as clipping on the cutting surface or reaching a threshold value in the data, are easy to implement.

The traversal is performed using a queue, initialized with the seed. In each cell entry are the S values for each vertex, interpolated values along each intersected edge, pointers to entries of neighboring cells (which also serve as face markers), bitmaps to tell what info is currently known, and a unique id. The queue head (current cell) is processed as follows:

1. compute missing S values and do the lookup
2. compute missing edge intersections, for both spatial and data values
3. for each sliced unmarked face with an unqueued neighbor, append the neighbor to the queue
4. for all neighbors in the queue, pass shared information and mark their adjoining face pointer so as to ignore the current cell when they queue *their* neighbors.
5. output display info

The potentially time-consuming part is searching the queue for neighbors in steps 3 and 4. A cell may have several "closer" neighbors, any one of which will process first and queue it. In general, the others have no way of knowing this has happened.

The simplest solution, when memory is available, is to mark each queued cell in an auxilliary list with the current slice number and its entry's address. This list has a slot for each cell in the grid and uses whatever cell addressing is natural to the grid. Searching is eliminated by comparing a cell's recorded slice number with the current slice. The list need be reset only when the slice number wraps around, relative to the word-length allocated for them. With this list, face marking in step 4 is not required since this list serves as a global memory of processing.

Alternatively, the queue may be partitioned into eight smaller lists by noting that the seed cell divides a volume into octants. Each cell is located by either the world coordinates of a distinguished vertex or, for structured grids, its subscripts. These locate it relative to the seed and determine its octant list. This partitioning only affects searches through the single queue. Since

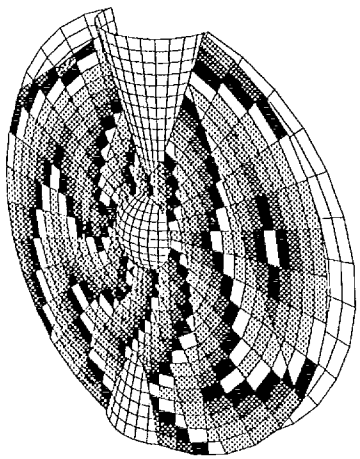

Figure 7: Traversal generations

the traversal front surrounds the seed, at least four of these lists become populated and they assist in reducing queue searching.

The search time may be reduced further by exploiting the structure of the queue (and each octant list). Define generation G_{k+1} to be the set of newly-encountered neighbors of all cells in G_k, where G_0 is the seed cell. Unfortunately, in general, these sets are not disjoint. At any moment the queue consists of a shrinking population of G_k at the front and a growing population of G_{k+1} at the rear. If a neighbor is already on the queue, it is more likely to be in G_{k+1} so the search begins there. It is guaranteed to be there for a cartesian grid and a planar surface, and G_k is the set of sliced cells with Manhattan distance k from the seed. Figure 7 is a representation of the traversal process. A cutting plane is nearly parallel to the rear boundary "wings" and the seed cell is at the lone white polygon. Each generation is marked by a different shade of gray, modulo a small number of shades. The nature of the spreading front shows clearly.

Informally, with a grid of N cells, the queue length is $O(N^{1/3})$, since queued cells form the circumference of an area on the surface.

Clipping

Limiting the cutting surface to a neighborhood of the hot spot can focus the user's attention by eliminating irrelevant parts of the volume and, by terminating the graph traversal, their computation time. Further, if the shape of the clip region can be controlled, then several such clipped surfaces may be combined into objects that convey more information than a single surface. A later

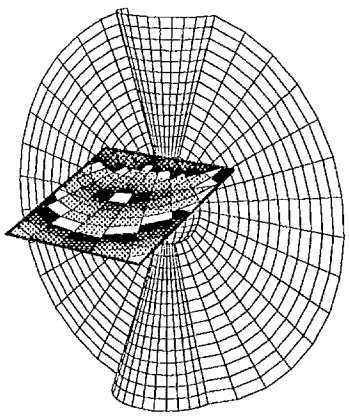

Figure 8: Clipped window

section describes such objects that assume rectangular clip windows on planar surfaces.

When a surface cuts a cell's face and enters a neighbor cell, the intersection becomes an edge on the surface which separates two polygons. If the edge is *completely* clipped away, the neighbor is not queued. If the edge is *partially* clipped then the window-edge endpoint of the edge lies on the face of a cell and linear interpolation between the original endpoints provides its value. This is Gouraud, not bilinear, interpolation and is quite sufficient. The four corner points of a rectangular clip window are interior points of cells and the analysis code can be called on to compute their values. Depending on the grid resolution and needs, the four corner polygons could be ignored, eliminating significant computational machinary for four points, out of perhaps many thousands. Figure 8 shows clipping applied to figure 6 and the curtailment of the traversal.

Seeding the Slice

A target point (hot spot) on the surface serves to find the seed cell and any solution to the point-containment problem will work. We don't accelerate the search by preprocessing because in the context of interactive slicing, our method is not a bottleneck. It amounts to line-of-sight fence-hopping (face-hopping) from a known cell to the point and works well when the surface and its hot spot move smoothly through the grid.

Starting at a point in any cell (the previous seed is an ideal choice), a line is constructed with parameter t to the target point such that $t = 0$ is the start point and $t = 1$ is the target. Next, intersections of this line with each face of the start cell are computed in terms of t. Faces are treated as having infinite extent. The line exits the cell on the face with the least positive value of t. Unlike ray tracing, we don't care *if* a face is actually intersected – we know that one of them is.

The line enters the adjoing cell and the process repeats. Eventually, a cell is reached where all candidate intersections have $t > 1$ and this is the seed. For the second and subsequent cells, the new start point may be repositioned away from the edges of the entry face to avoid looping around edges or vertices. This strategy also adds robustness in case of nearly-planar faces.

As a cutting surface is driven interactively by a user, the hot spot moves incrementally through the grid and little work is required to update the seed cell. When the hot spot is driven "out of sight" (the line of sight exits, then re-enters, a non-convex grid domain) of the starting point or out of the grid, we apply the heuristic of starting over from another region of the grid. For example, in a structured grid the center cell on each of the six grid boundaries usually provide good vantage points. If it still isn't located, it is reasonable to assume it is out of the grid. One could define alternate hot spots, refuse to move it out of the grid, or simply ask the user to return it.

Probes

We now turn to an application of the slicing algorithm which uses plane surfaces. A *probe* is a collection of planar cutting surfaces, refered to as *sheets* to imply they may have limited extent. It has a local coordinate system defined by the linear transformation **to_probe** which maps world coordinates to the probe's. Likewise, each of its sheets, s, has a local coordinate system defined by $\mathbf{to\_sheet_S}$ which maps from the probe to the sheet. A sheet's cutting surface is its $z = 0$ plane which conveniently defines the two halfspaces for the characteristic function S. Its origin is a simple choice for hot spot and the world coordinates are the bottom row (assuming transforms post-multiply points) of

$$(\mathbf{to\_probe} * \mathbf{to\_sheet_S})^{-1}$$

Each sheet also has a special-effects transform $\mathbf{fx_S}$ and a map to the unit cube $\mathbf{to\_cube_S}$. $\mathbf{fx_S}$ provides zooming and panning (parallel or normal to the xy plane). $\mathbf{to\_cube_S}$ maps a selected region of the sheet containing the hot spot into the unit cube for clipping, although for this application clipping applies to the unit square. The complete transformation from world coordinates to each sheet is

$$\mathbf{to\_probe} * \mathbf{to\_sheet_S} * \mathbf{fx_S} * \mathbf{to\_cube_S}$$

The inverse mapping is

$$(\mathbf{to\_probe} * \mathbf{to\_sheet_S} * \mathbf{to\_cube_S})^{-1}$$

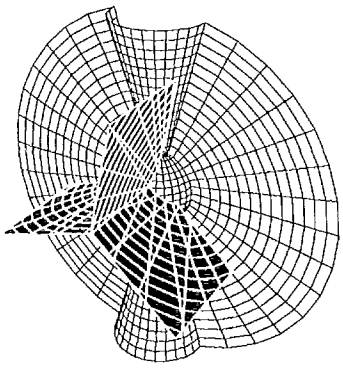

Figure 9: Paddlewheel probe

and is just a modeling transform to the host graphics system, that is, there is no execution overhead. $\mathbf{fx_S}$ does not participate in the inverse so that its effects remain for display.

The entire probe is driven by manipulating only **to_probe**. Probes are designed by specifying each $\mathbf{to\_sheets_S}$. The simplest probe has a single sheet and, with its varying clip window and zooming, resembles a magnifying glass roaming through the data. Two other designs are

trihedral: This probe uses three orthogonal sheets and provides good three-dimensional information near their intersection. Such "corner" views are common with voxel-based tools, but this probe is easily rotated to any orientation.

paddlewheel: Several sheets touch along an axis and are separated by at least 60°, less if transparency is used to avoid obscured sheets. When rotated about the axis, the paddlewheel sweeps out a region. A complete revolution may be faked by film looping the sweep between consecutive vanes. Figure 9 shows a 3-vane paddlewheel.

The potential for probes lies in the speed in which a volume may be swept by multiple surfaces. When they are swept rapidly, smoothly, and repeatedly through the volume, using nothing more than color-encoding of the data and Gouraud polygon fill, the effect can be like a continuum of contour shells. Experience with multiple IJK slices of structured grids, as described earlier, on 16mm film showed that "retinal memory" is a powerful mechanism for grasping the whole of a volume.† Probes provide additional structure to the slices and suggest ways of sweeping, as in the paddlewheel. With today's top-end workstations, and whatever the future holds, this effect is within reach of interactive use. Of course, it could also work for other volume display methods, though slicing holds the earliest promise for the first reason mentioned in the introduction.

† Work done by Robert Smith, Eric Everton, and one of the authors (DS) at NASA Langley Research Center, starting in 1982, and described briefly in [3].

Extensions

We have two planned extensions for probes. The first will incorporate contouring within the neighborhood of a sheet. By moving a (2D) pointer along the surface of a sheet, the user will be selecting the data value at that point. Its contour will then be built within the sheet's clipping box. By varying the probe's position and size of the box, users will see contours in any region of interest without an exhaustive search through the entire grid.

The second extension is for vector fields. By adding to the scalar display, on sheets, lattices of points from which vectors or particles may launch, probes become devices that vary continuously from "tuft screens" [8] to particle rakes. Tufts are pieces of yarn which blow in the direction of flow and are idealized in graphics by variable length vectors (and are not to be confused with "hedge-hogs," which display surface normals). A vector is essentially a single time-step particle trace. This device gives the user control over its size, lattice density, and the number of time steps (and their direction in time) to trace particles.

Conclusion

We have presented an algorithm for slicing the computational grids commonly used today. It maximizes the use of local knowledge, at each sliced cell, of the behavior of the slice in order to eliminate global search and redundant calculations. Using it for planar surfaces, we next constructed probes of several planes for use in scalar fields. Probes may be used to show total data behavior in a volume by exploiting the mind's ability to accumulate an image by repeated exposure to pieces of it.

Two simple extensions are described for merging slicing with contouring and for examining vector fields. The characterization of all the tools discussed is that they have user-defined extent and six-degrees-of-freedom mobility, emphasizing their role in exploring data.

Acknowledgements

We appreciate the helpful discussions with Chris Berry,

Olivier Hardy, and C. Y. Huang of The Computational Mechanics Company, K. R. Subramanian and Chris Buckalew of UT's Computer Sciences Department, and Lee Metrick of Schlumberger's Austin System Center. Keith Waters of Schlumberger's Lab for Computer Sciences provided help in making a video at a crucial point and C. Y. Huang provided the images of 2D grid types.

References

[1] *Computational Fluid Dynamics*, Office of the Chief Scientist, NASA Langley Research Center, undated.

[2] Duda, R., Hart, P., *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[3] Gregory, T., Carmichael, R., "Interactive Computer Graphics: Why's, Wherefore's, and Examples," *Astronautics & Aeronautics*, April 1983.

[4] Ho, S. H., "Visualization of 3D Solid Finite Element Meshes by the Method of Sectioning," *Computers & Structures*, Vol. 35, No. 1, pp. 63–68, 1990.

[5] Löhner, R., Parikh, P., Gumbert, C., "Some Algorithmic Problems of Plotting Codes for Unstructured Grids," *Proceedings of the AIAA* 9^{th} *Computational Fluid Dynamics Conference*, June 1989, AIAA-89-1981.

[6] Lorensen, W., Cline, H., "Marching Cubes: A High Resolution 3D surface Construction Algorithm," *Computer Graphics*, Vol. 21, No. 4, 1987.

[7] Mandelbrot, B., *The Fractal Geometry of Nature*, W. H. Freeman & Co., 1983.

[8] Merzkirch, W., *Flow Visualization*, 2^{nd} edition, Academic Press, Inc., 1987.

[9] Neal, M., "What's Going On Up There ?", Application Briefs, *Computer Graphics and Applications*, July 1987, p. 8.; also in "Visualization in Scientific Computing," SIGGRAPH Video Review 28.

[10] Norman, M., in "Visualization in Scientific Computing," SIGGRAPH Video Review 28.

[11] Sengupta, S., Hauser, J., Eiseman, P., Thompson, J. (eds.), *Numerical Grid Generation in Computational Fluid Mechanics '88*, Pineridge Press Ltd., 1988.

[12] Watson, V., Buning, P., Choi, D., Bancroft, G., Merritt, F., Rogers, S., "Use of Computer Graphics for Visualization of Flow Fields", *State of the Art in Data Visualization*, SIGGRAPH '89 Course #28 Notes, July 1989; also in AIAA Aerospace Engineering Conference, Feb 1987.

[13] Winget, J., "Advanced Graphics Hardware for Finite Element Results Display," *Advanced Topics in Finite Element Analysis*, ASME Pressure Vessels and Piping Conference, 1988, PVP Vol. 143.

[14] Zienkiewicz, O., Taylor, R., *The Finite Element Method*, 4^{th} ed., vol. 1, McGraw-Hill, 1989.