Rapid Exploration of Curvilinear Grids Using Direct Volume Rendering (Extended Abstract)

Allen Van Gelder and Jane Wilhelms
Computer and Information Sciences
University of California, Santa Cruz CA 95064
avg@cs.ucsc.edu wilhelms@cs.ucsc.edu

Abstract

Fast techniques for direct volume rendering over curvilinear grids (common to computational fluid dynamics and finite element analysis) are developed. Three new projection methods that use polygon-rendering hardware for speed are presented and compared with each other and with previous methods for tetrahedral grids and rectilinear grids. A simplified algorithm for visibility ordering, based on a combination of breadth-first and depth-first searches, is described. A new *multi-pass blending* method is described that reduces visual artifacts that are introduced by linear interpolation in hardware where exponential interpolation is needed. Visualization tools that permit rapid data banding and cycling through transfer functions, as well as region restriction, are described.

1 Introduction

Direct volume rendering is attractive because of its extreme flexibility, being able to map data values to color and opacity in any fashion and providing information about the whole volume in a single image. Information in such a semi-transparent image could be clarified by interactive manipulation, but such animation is hampered by computational cost. Fast methods generally require powerful parallel processors, or are subject to visual artifacts. Problems of speed and artifacts are exacerbated when volume-rendering non-rectilinear grids. However, when using direct volume rendering for a general perusal of volume information, improvements in speed may be worth even relatively significant artifacts.

Direct volume rendering is a visualization method for scalar sample data volumes where values are mapped to color and opacity and directly rendered by accumulation to the screen pixels. It can be done by ray-casting [DCH88, Lev88, UK88, Kru90], or by projecting volume sample regions or cells to the screen [UK88, Wes90, LH91, MHC90, ST90, WVG91]. Projection must be in front-to-back or back-to-front order for correct compositing with fractional opacity values. If interpolation between sample points and integration in depth are not done accurately visual artifacts may occur [WVG91]. (By cell depth, we always mean the thickness in the direction orthogonal to the screen.) Further, because no geometric primitives such as polygonal isosurfaces are extracted, most or all of the work of direct volume rendering must be repeated if the viewpoint changes. Significant speed-ups can be achieved by the use of coherence, by simplifying interpolation and integration, and by making use of graphics hardware [UK88, MHC90, ST90, LH91, WVG91]. Our goal in this investigation was to achieve some of these gains on curvilinear grids.

2 Overview of This Research

A curvilinear grid is a 3-dimensional rectilinear grid in computational space that is "warped" in physical space around regions of interest (e.g., aircraft wings in computation fluid dynamics applications). The grids present problems for direct volume rendering because cells may vary greatly in size (e.g., neighbor distances in a commonly used grid [HB85] vary by a factor of 10,000), and have irregular shapes and degeneracies (e.g., multiple sample points in computational space may map to the same physical space location). All methods described herein are designed for curvilinear grids that may not be convex as a volume, but whose cells are 6-sided convex polyhedra (hexahedra), possibly with some degeneracies in that some edges of the cell have zero length, and some faces have zero area.

Initial explorations convinced us that using ray-casting to directly render these volumes was unacceptably slow [RW92]. The main thrust of this paper is to explore more rapid, projection-based, methods. We have implemented three projection methods for curvilinear grids. All of our methods have some things in common:

1. they convert cell projections to Gouraud-shaded

polygons and use hardware for rendering;

2. they use hardware compositing; and

3. they preprocess cell information for speed, at some cost in space.

The main ideas are summarized here; a more complete presentation is offered in a technical report [VGW93].

It is hard to compare these projection methods with ray-casting exactly, because the ray-caster is more sensitive to image size and orientation. For images taking up half or most of a 500x500 pixel window, Table 1 shows that, compared to the ray-caster:

1. For rendering from a new viewpoint, the speedup is a factor of 4 to 94, depending on the projection method.

2. For re-rendering without rotation, the speedup is a factor of 50 to 150, depending on the projection method.

Although not shown in the table, as image size increased by a factor of two due to zooming, the cost of ray-casting tripled, while the cost of hardware-assisted projection was negligibly affected. Also, the ray-caster required two sizable precomputed files to achieve its performance.

The faster projection methods may show significant artifacts in certain cases; however, the most careful of the three methods produces images that are usually close in quality to the ray-caster. Our new projection methods are described in Section 3.

To take into account opacity, a front-to-back ordering must be established, because cells (defined by eight corner sample points) in front may partially or totally obscure those behind. Because curvilinear grids can wrap around calculating this visibility ordering is nontrivial [MHC90, Wil92b]. Further, accumulating color and opacity values correctly in depth involves an exponential function [Kru90, MHC90, WVG91]. To approximate this quickly by a quadratic, we have developed a multi-pass blending method. These issues are discussed in Section 4 and Section 5.

In Section 6 we discuss some new methods of rapidly designing and changing transfer functions for volume exploration. In Section 7, we discuss region restriction.

3 Rapid Rendering Approaches

Although most previous work on direct volume rendering is limited to regular grids, irregular grids are receiving increasing attention. Ray-casting on irregular grids has been reported [Gar90, Use91, RW92]. A few researchers have explored projection methods [MHC90, Wil92b, Wil92a]. Challinger [WCA+90, Cha90], Lucas [Luc92], and Giertsen [Gie92] use methods that could be considered hybrids of both types. Most of these methods were found to be significantly slower than what we needed; an exception being Williams' approach (hardware Gouraud-shading of tetrahedra). However, when the grid is naturally hexahedral, the latter method involves translating each cell into five tetrahedral cells; it also seemed to have problems with artifacts.

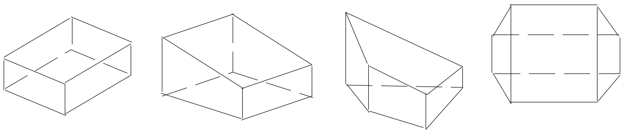

Figure 1: Typical projection of rectilinear cell and a few irregular hexahedron projections.

We have implemented three projection methods for curvilinear grids. Certain hardware features are necessary for these methods to be successful: hardware Gouraud-shading in color and opacity; hardware compositing; and, minimally, eight bits of color and opacity per pixel to reduce precision problems and aliasing. No accumulation buffer is used.

Projection of irregular hexahedra is significantly more complex than that of rectilinear cells when cell depth is considered. For a parallel projection of a rectilinear cell there are 3 nondegenerate and 11 degenerate projection topologies [WVG91]. Also, for a given viewpoint, all cells in the volume fall into the same case. For irregular hexahedra, the number of nondegenerate projection topologies is significantly higher (see examples in Figure 1), and the number of degenerate ones higher still. Also, various cells fall into different cases.

The first of our three methods avoids the issue entirely and only draws faces, without consideration of cell depth. Our second is a rough approximation to correct projection. Our third accurately analyses the projection of each cell. (See Fig. 4.)

3.1 Depthless Cell Face Projection

Our first method is a very simple but admirably fast one: each data value is mapped to a color and opacity and the faces of each curvilinear grid cell are drawn as Gouraud-shaded polygons whose vertices have these mapped values. Data structures for this method record information for three adjoining faces of the cell, so each face is only drawn once. Usually this method is used with zero-opacity for maximum speed. By "zero-opacity", we mean that color values are added to each pixel without any reduction due to opacity. For such a case, no visibility ordering is needed.

This method is extremely fast and trivial to implement. Small cells contribute the same intensity as large cells, depth not being considered. On our grids, cell size is generally inversely proportional to interest, because volumes are finely gridded in areas of most interest. Some scientists may prefer this automatic weighting. Further, there is a problem with using hardware-compositing on these grids (or any grids with many tiny cells), because the typical intensity/opacity resolution is only eight bits per channel. Small cells may contribute well under 1/255 of the maximum possible intensity and, thus, never appear in the image at all. Using the depthless method, data is not ignored in this way, just improperly weighted.

A more serious problem is that noticeable visual artifacts appear from some angles because the distance between cell faces is not taken into account. These artifacts tend to delineate cell boundaries and probably would not be misinterpreted as data information.

3.2 Cell Face Projection with Depth

Our next method takes some account of cell depth. For each cell vertex, a data value and depth is calculated. The depth is the distance between the front and back faces of the cell that project to the location of that particular vertex (usually zero on the convex hull). The data value is the average of the data value of the vertex and the interpolated data value of the location on the other cell face that projects to the same location.

First, cell vertices are mapped to screen space. For each cell, vertices that lie on the convex hull of the projection of the cell are identified, by creating an edge list for the cell and traversing it around the exterior starting at the lowest vertex. The remaining cell vertices are interior to the convex hull. Convex hull vertices with non-zero depth are recognized either because: two vertices project to the same location (in which case depth and estimated data value is stored with one vertex, the other being treated as depthless); or three consecutive convex hull vertices project onto a line.

For any interior (not on the convex hull) cell vertex, the appropriate opposite cell face is found by examining a cell edge list generated for convex hull detection. The two cell edges lying immediately left and right of the cell vertex in question (but not emanating from that vertex), define the opposite cell face. Interpolation across that face is used to find a data value and z-depth, which are used to calculated average data value and cell depth at that location.

Cell vertex information need only be recalculated and stored when the volume is rotated. The mapping to color and opacity occurs during drawing, so changes in mapping do not require recalculation of the projection. For rendering, each face is drawn once for each cell as a Gouraud-shaded polygon.

This method gives a more "realistic" rendering (assuming some physical, colored medium being imaged), but it only an approximation of the correct projection. For some angles, it produces noticeable artifacts. While slower for new orientations than the depthless method, it is much faster than the more correct method presented next.

3.3 Incoherent Projection

The method we have dubbed "incoherent projection" is the most careful of the projection methods presented, and the most expensive. It builds on the "coherent projection" technique for rectilinear grids [WVG91], extending it to general convex hexahedra. Unlike coherent projection of rectilinear volumes, in curvilinear volumes different cells do not all fall into the same case, so a table-based case analysis technique was not attractive to implement. Therefore, each cell is analyzed individually in screen space. A pleasant side-effect of this approach is that perspective projections are no more difficult than parallel ones.

The main idea is to render the 2-dimensional screen space projection of each cell as an arrangement of polygons, each polygon being a region that has the same front and back face in the cell projection. This allows simplifying assumptions to be made concerning these polygonal regions.

The algorithm determines the polygonal regions using a sweep method described below. Some of the vertices of the polygonal regions are actual cell corners; others are not and must be calculated. As described in the previous section, an average data value (mapped later to color and opacity) and cell depth for vertices of the polygonal regions must be calculated, taking into account values at the cell exterior that projects to these vertices. Finally, hardware Gouraud shading is used to color each of these polygonal projections using the calculated color and opacity for vertex values.

As shown in Figure 2, some vertices (A and B) corresponding to the polygonal decomposition of the projection of a hexahedron do not correspond to vertices of the hexahedron, but are produced by intersection of edge projections. We call these *intersection vertices*. The first technical issue is the location of these intersection vertices. We used a sweep line algorithm which we now outline, first assuming no degeneracies. The algorithm simultaneously finds the intersection

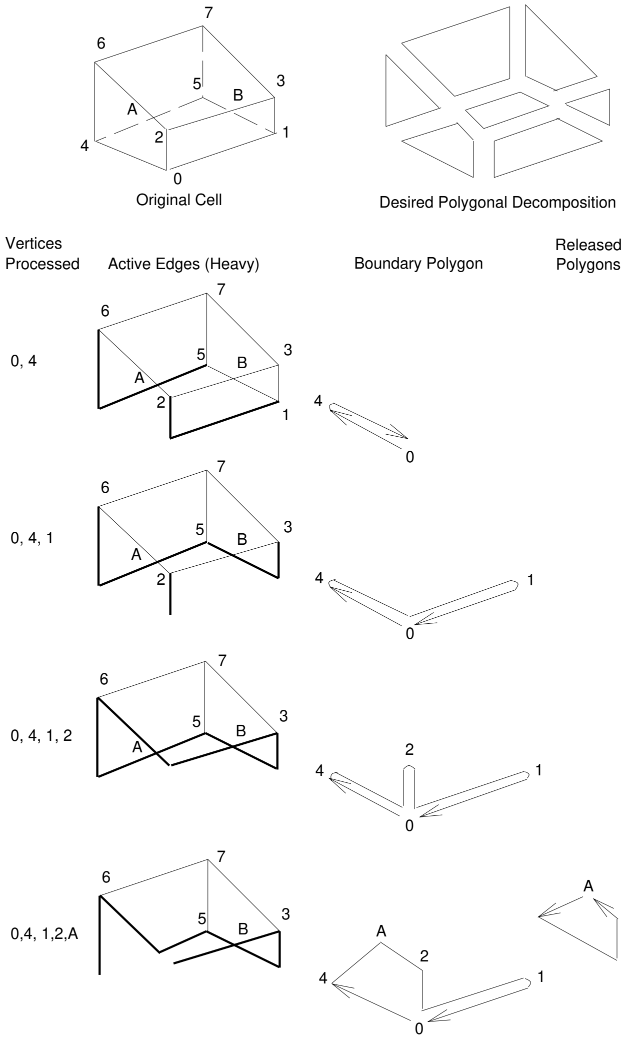

Figure 2: Illustration of Sweep Algorithm. When A connects to boundary in two places, a closed counterclockwise polygon is formed, and spliced out.

vertices and the polygons that comprise the arrangement of the projection. (It also finds the convex hull as a by-product.) The hexahedron to be analyzed is given in screen space (x, y, z), so the objective is to find its projection on the x-y plane.

The algorithm maintains three data structures:

1. A priority queue of vertex *events*. The "minimum" of this event queue is the unprocessed vertex with minimum y-value.

2. An x-sorted list of *active edges*, where an edge is active if it goes from a processed vertex to an unprocessed vertex.

3. A *current boundary polygon* in the form of an edge list, surrounding the processed vertices and edges.

Initially, the event queue contains the original hexahedron vertices sorted by y-value, and the active-edge list and current boundary are empty.

This algorithm follows the standard pattern of sweep-line algorithms. Updating the *current boundary* proceeds as suggested in Figure 2. Insert the first removed edge and its reversal into the current boundary edge list such that the (nonconvex, nonsimple) polygon formed is planar (no edges cross); this forms a sort of needle. If there is a second removed edge, do likewise, but this completes a counterclockwise polygon, which is spliced out. (The last vertex of the cell has a third removed edge, which creates a second polygon to be spliced out.)

The above outline omits the details of handling "degeneracies". Projection degeneracies occur when any two projection vertices have the same x-value or y value. These can be removed by assuming a slight rotation of the screen space that does not change any nondegenerate topology. A more difficult, less standard degeneracy occurs when the original hexahedron is itself degenerate: if two points coincide in 3-space, no spatial transformation will separate them. Our solution was based on certain assumptions about what degeneracies could occur: we assume that no two adjacent edges have zero length; that no two adjacent faces have zero area; and that faces are planar. These assumptions leave a lot of flexibility: tetrahedra and pentahedra can be represented as degenerate hexahedra.

Suppose an edge of zero length is encountered. We want to perturb the hexahedron to give the edge some length while maintaining convexity. This requires finding a direction in which the two coinciding vertices can be "pulled apart". If the edge connects two faces with positive area, their planes intersect in a well-defined line that determines the required direction. The more difficult case is when the edge is adjacent to one face of zero area. Then we take advantage of the fact that, under the above assumptions, the diagonally opposite edge "in the same direction" cannot be degenerate. Form a triangle with this opposite edge as base and the coinciding vertices as apex, then slightly "pull apart" the coinciding vertices in the planes of the triangle and the face of positive area.

4 Visibility Ordering

A visibility ordering, which is an ordering on the cells such that no earlier cell occludes a later cell in screen space, is necessary to render cells taking into account opacity. Visibility ordering issues for tetrahedra were thoroughly explored by Williams

[Wil92b]. This section outlines a considerably simpler implementation for curvilinear grids that is robust in practice. The main ideas are also applicable to tetrahedral grids. Two issues concerning visibility ordering are: does one exist, and if so, how to find one. Although the theory is murky in the general case, in practice our method has never failed to find a visibility ordering. Williams reports similar practical experience.

The main idea that is well known for efficient visibility ordering is that of linear-time topological sorting [MHC90, Wil92b]. A topological sort of a directed acyclic graph is a numbering of its vertices such that there is no path from a smaller vertex to a larger one. This can be accomplished in linear time by a depth-first search and post-order numbering. For the visibility application the graph's vertices are cells and its directed edges given by the *immediately occludes* relation: cell A *immediately occludes* cell B if they share a face and A occludes B in screen space. For convex volumes, a topological sort finds a visibility order if one exists and discovers a cycle otherwise [MHC90, Wil92b]. Though there is no definite theory know for nonconvex volumes, they are common in practice. Williams describes an heuristic for nonconvex volumes. We present an alternative that is considerably simpler, for connected, possibly nonconvex, volumes.

Our algorithm builds a directed graph from the undirected graph defined by the adjacency of cell, by taking into account the orientation of the shared face in screen space. (The z component in screen space of the shared face normal determines which cell occludes the other.) We combine an undirected breadth-first search with directed depth-first searches. For curvilinear grids, edges need not be represented explicitly, as they can be determined by arithmetic on cell indices.

The breadth-first search is implemented with a FIFO queue of cells, which initially contains one cell that has a vertex that is farthest from the viewpoint, and all cells are unmarked.

```
nextNum = 0;
while (FIFOqueue not empty)
  nextCell = front(FIFOqueue);
  if (nextCell not marked)
    newNum=depthFirstsearch(nextCell,nextNum);
    nextNum = newNum;
 dequeue(nextCell);
```

The depth-first search also needs to test edge directionality, and post "uphill" neighbors to the FIFO queue; otherwise it is quite standard.

```
depthFirstSearch(cell, nextNum)
  mark cell;
  for (neighbor adjacent to cell)
    if (neighbor not marked)
      if (neighbor immediately occludes cell)
        enqueue(neighbor, FIFOqueue);
      else
        newNum=depthFirstSearch(neighbor,nextNum)
        nextNum = newNum;
    else if (neighbor marked & has no visNumber)
      Error - Cycle in vis order,
      (has never happened in practice);
    else
      {neighbor has visNumber; do not visit}
  visNumber[cell] = nextNum;
  return  nextNum + 1;
```

Williams reports that about 60,000 tetrahedra per second can be ordered (SGI 4D/VGX). We found that a comparable number of hexahedra per second were ordered by our algorithm. Converting to tetrahedra would increase ordering cost by a factor of 5.

5 Multi-pass Blending

Linear interpolation is not always what is desired, but it is what the hardware offers. However, SGI workstations have a blend function setting that permits the "source" color to be multiplied by "one minus source alpha", and added to the background (cells already rendered). This permits some quadratic functions to be used for color interpolation, by multiplying two linear functions: color and alpha. (First, in separate passes, the background needs to be attenuated according to the new cell's opacity, with a different blend function; this explains the name "multi-pass".)

Assume a cell is filled with a semi-transparent light-emitting medium. When cell faces are planes, the depth δ of the cell varies linearly along any line, but the effective transmission of color varies as $(1-e^{-\alpha\delta})$. This can be approximated between two vertices in the projection by a quadratic function of δ that is zero at vertices of 0 depth and gives the correct color value at the "thickest" vertices of the cell projection. The remaining parameter of the quadratic was chosen to minimize the squared error between the quadratic and the exponential function that it is approximating, over the range of thicknesses that occur in the cell. Somewhat amazingly, this can be solved in closed form; details appear in a technical report [VGW93].

The drawing phase for multi-pass blending takes about four times as long as for single-pass (see Table 1) when applied to all cells. Therefore, a sensible optimization would be to use it only on cells that project onto several hundred pixels.

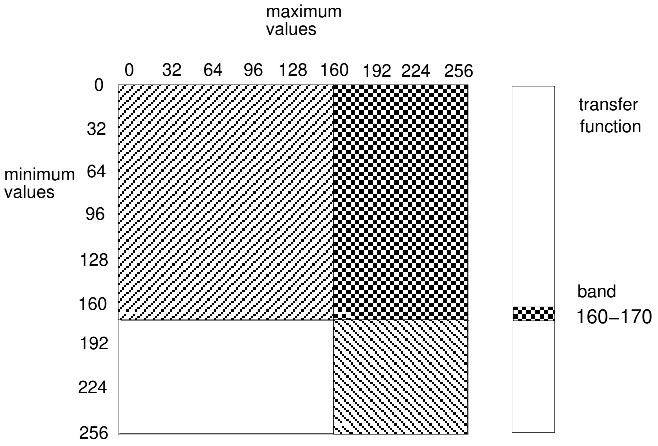

Figure 3: Example of Data Structure for Banding: Only cells associated with upper right region need be drawn.

6 Transfer Function Manipulation

Our visualization software provides an an interactive transfer function editor to map data values to color and opacity, but it can still be frustrating and slow to find useful mappings. We developed a *band* option where the user defines one or more bands of data values that are visible, and only cells with values within the bands need be drawn.

Rendering with a single-band, in particular, is extremely fast. A supplemental data structure consisting of a two-dimensional array of size 256 by 256 quickly finds cells in the range. Each array location is a linked list of pointers into the data. (We assume 8-bit channels for color and opacity.) The minimum data value of a cell (scaled into the range 0–255) determines the row of the array and the maximum data determines the column of the array with which a cell is associated. In drawing an image using the banded function, only those rows less than or equal to the maximum value of the band need be drawn; and within those rows, only columns greater than or equal to the minimum value of the band need be drawn (see Figure 3). This method generally takes much less time than rendering the whole volume. The band can easily and quickly be cycled, giving the effect of a moving fuzzy isosurface and helping to indicate the relation of neighboring data regions.

7 Restriction and Inverse Mapping

The final approach in our recent attempts to make direct volume rendering more useful was to implement a method of interactively restricting the parts of the volume rendered. At the suggestion of Arsi Vaziri of NAS/NASA-Ames, we invert the mapping for this restricted region to find the actual locations within the data that are being drawn. Lucas has described another approach to this problem using z-buffering [Luc92].

The restricted area is a simple bounding box defined interactively using sliders. Adding the option to draw cell origins as points helps clarify the relation of the volume rendering to the sample points. If the restricted region is fairly small, it is practical to print the locations and values of the cells that lie within the box to the screen, allowing the user to determine the actual computational space location of features of interest.

8 Experimental Results

We explored these methods on two curvilinear grids. (The software works on a regular grid, but it is not optimized to take advantage of the greater simplicity of these grids.) The curvilinear grids tested were the "blunt fin" [HB85] and the "post" [RKK86], both from NASA-Ames Research Center. The blunt fin is contains 40,960 samples, and the post contains 109,744 samples.

Table 1 shows the rendering times for our three projection methods, and a ray caster for comparison, using these two grids. Times are user and system CPU seconds on a Silicon Graphics R3000-based uniprocessor VGX. Figure 4 shows representative images produced by our projection methods. Coherent projection [WVG91] required about 4 to 7 seconds on a comparably sized *rectilinear* grid.

First we consider our three direct volume rendering methods. Cell face projection without depth is significantly faster than the others and is desirable for rapid scanning of the volume, though it is sometimes subject to artifacts. We ignore the cost of making the data structure for this method because it is done once as the data is read in and never changes. Cell face projection with depth uses the size and shape of cells more carefully, and requires rendering information that changes with orientation. We calculate this information (the "making data structures" cost in Table 1) and store it. Drawing the image from these structures takes about three times as long as the depth-less method ("Rendering" columns in Table 1). Precomputation is desirable because the image can be scaled, translated, transfer functions changed and intensity/opacity scaled, without recomputing the data structures.

Incoherent projection, our most careful and expensive method is more time-consuming but freer from

Data Set	Method	Visibility Sort	Making Data Structures	Single-Pass Rendering	New Viewpt. Total	Multi-pass Rendering
Blunt Fin	Faces Without Depth	0.64	–	1.14	1.8	–
	Faces With Depth	0.64	6.63	2.94	10	11.72
	Incoherent Projection	0.64	37.56	3.52	42	12.05
	Ray-Casting	–	–	–	170	–
Post	Faces Without Depth	1.65	–	3.02	5	–
	Faces With Depth	1.65	21.76	10.99	34	39.76
	Incoherent Projection	1.65	104.48	13.70	120	37.38
	Ray-Casting	–	–	–	280	–

Table 1: Rendering Times of Blunt Fin and Post Data. (Time in CPU seconds.)

artifacts. Again we split the calculation into determining orientation-specific information and then rendering. The cost of re-rendering without orientation changes is not much worse than the method based on cell faces with depth.

The linear-time visibility sort used contributed only minimally to the cost of rendering. Multi-pass blending noticeably increased rendering time, by three or four times. However, the multi-pass method can produce smoother images.

9 Conclusions

We discovered that projection methods do provide reasonable speed for volume rendering medium-sized curvilinear grids, far beyond what we could achieve with ray-tracing approaches. We found the versatility provided by a range of methods of varying speed and image quality very helpful. Use of transfer function banding provided a convenient and fast method of exploring volume contents, particularly when used with band cycling, and region restriction helps to clarify image contents and relate it to the original volume.

Acknowledgements

This research was supported in part by NSF PYI grant CCR-8958590, NSF New Technologies Program grant ASC-9102497, NSF Infrastructure grant CDA-9115268, and a NASA-Ames Research Center Cooperative Agreement Interchange No. NCC2-717. We thank Tom Goodman, Andy John, and Ron MacCracken for their help in programming.

References

[Cha90] Judy Challinger. Object-Oriented rendering of volumetric and geometric primitives. Master's thesis, University of Califoria, Santa Cruz, 1990.

[DCH88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *Computer Graphics*, 22(4):65–74, July 1988.

[Gar90] Michael P. Garrity. Raytracing irregular volume data. *Computer Graphics*, 24(5):35–40, December 1990.

[Gie92] Christopher Giertsen. Volume visualization of sparse irregular meshes. *IEEE Computer Graphics and Applications*, 12(2):40–48, March 1992.

[HB85] Ching-Mao Hung and Pieter G. Buning. Simulation of blunt-fin-induced shock-wave and turbulent boundary-layer interaction. *J. Fluid Mechanics*, 154:163–185, 1985.

[Kru90] Wolfgang Krueger. Volume rendering and data feature enhancement. *Computer Graphics (Proceedings of the San Diego Workshop on Volume Visualization)*, 24(5):21 – 26, 1990.

[LH91] David Laur and Pat Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. *Computer Graphics (ACM Siggraph Proceedings)*, 25(4):285–288, July 1991.

[Lev88] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, March 1988.

[Luc92] Bruce Lucas. A scientific visualization renderer. In *Proceedings of Visualization '92*, pages 227–233. IEEE, October 1992.

[MHC90] Nelson Max, Pat Hanrahan, and Roger Crawfis. Area and volume coherence for efficient visualization of 3d scalar functions. *Computer Graphics (ACM Workshop on Volume Visualization)*, 24(5):27–33, December 1990.

[RW92] Shankar Ramamoorthy and Jane Wilhelms. An analysis of approaches to ray-tracing curvilinear grids. Technical Report UCSC-CRL-92-07, University of California, Santa Cruz, 1992.

[RKK86] S. E. Rogers, D. Kwak, and U. K. Kaul. A numerical study of three-dimensional incompressible flow around multiple posts, 1986. AIAA paper 86-0353, Reno, Nevada.

Figure 4: *Left*: Comparison of Volume Rendering Methods on Blunt Fin. *Right*: Close-up with Incoherent Projection; Transfer Function Display alongside Blunt Fin Grid; Post Image.

[ST90] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, December 1990.

[UK88] Craig Upson and Michael Keeler. The V-buffer: Visible volume rendering. *Computer Graphics*, 22(4):59–64, July 1988.

[Use91] Sam Uselton. Volume rendering for computational flid dynamics: Initial results. Technical Report RNR-91-026, NAS-NASA Ames Research Center, Moffett Field, CA, 1991.

[VGW93] Allen Van Gelder and Jane Wilhelms. Rapid exploration of curvilinear grids using direct volume rendering. Technical Report UCSC-CRL-93-02, 1993.

[Wes90] Lee Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–76, August 1990.

[WCA+90] Jane Wilhelms, Judy Challinger, Naim Alper, Shankar Ramamoorthy, and Arsi Vaziri. Direct volume rendering of curvilinear volumes. *Computer Graphics*, 24(5):41–47, December 1990.

[WVG91] Jane Wilhelms and Allen Van Gelder. A coherent projection approach for direct volume rendering. *Computer Graphics (Proceedings ACM Siggraph)*, 25(4):275–284, 1991.

[Wil92a] Peter Williams. Interactive splatting of nonrectilinear volumes. In *Visualization '92*, pages 37–44. IEEE, October 1992.

[Wil92b] Peter Williams. Visibility ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, April 1992.