Direct Volume Rendering of Curvilinear Volumes

Jane Wilhelms, Judy Challinger,
Naim Alper, and Shankar Ramamoorthy
University of California, Santa Cruz

Arsi Vaziri,
NAS/NASA-Ames Research Center

Abstract

Direct volume rendering can visualize sampled 3D scalar data as a continuous medium or extract features. However, it is generally slow. Furthermore, most algorithms for direct volume rendering have assumed rectilinear gridded data. This paper discusses methods for using direct volume rendering when the original volume is curvilinear, i.e., is divided into six-sided cells which are not necessarily equilateral hexahedra. One approach is to ray-cast such volumes directly. An alternative approach is to interpolate the sample volumes to a rectilinear grid, and use this regular volume for rendering. Advantages and disadvantages of the two approaches in terms of speed and image quality are explored.

1 Introduction

Direct volume rendering creates an image of sampled volume data by mapping from the data values directly to pixel contributions, without the creation of intermediate geometric representations. By using sophisticated mapping schemes, the volume can be visualized as a semi-transparent continuous medium or features such as isosurfaces can be extracted, or both [DCH88, Lev88, Sab88, UK88, Wes90]. This extreme flexibility makes direct volume rendering an attractive technique. However, it is slow, and considerable recalculation is necessary to create images from alternate viewpoints. Methods that create intermediate geometrical representations, such as polygon meshes, have the advantage that algorithms available in hardware on graphics workstations can make rendering such representations interactive.

Another important limitation on most published methods in direct volume rendering is that data is generally assumed to be sampled in a rectilinear grid. In many applications, sample data is not distributed in this regular a fashion. Grids may be warped to fit around objects or consist of highly irregular sample patterns.

In this paper, we explore the use of direct volume rendering for data from computational fluid dynamics. Relatively little work has been done in this area [SN89]. The sample points are assumed to lie in a "curvilinear grid", where space is subdivided into cells defined by eight corner points, but the faces of the cells are not necessarily planar and the grid points may not lie on orthogonal axes [Fle88, plo89]. We study two approaches:

1. Cast rays through the curvilinear volume, extending ray-casting methods for rectilinear volumes [Lev88, Cha90].
2. Interpolate between the sample points to find new sample points arranged in a rectilinear grid, and use traditional ray-casting on the new volume.

The first approach is less prone to accumulation of interpolation errors and easily accommodates irregular sampling densities. The second approach is generally much faster, as many simplifying assumptions can be made in traversing regular grids.

Section 2 provides background in direct volume rendering. Section 3 describes the ray-casting method used. Section 4 describes the interpolation methods used to convert curvilinear volumes to rectilinear ones. Section 5 provides some experimental results. Section 6 describes extensions to the work using hierarchical interpolation. Section 7 describes some encouraging results on using volume visualization to study turbulence. Section 8 draws conclusions.

2 Overview of Volume Rendering

For direct volume rendering, visual contributions from the volume to each pixel must be calculated by mapping the scalar sample values within the volume to color and opacity. Early work using this method (see, e.g. [Wil90b] for references) used quite simple mappings. Recent work has concentrated on flexible and sophisticated mappings which can represent the volume as a continuous semi-transparent medium, extract features, or use some combination of both [DCH88, Lev88, Sab88, UK88, Wes90]. The two main approaches used for volume rendering are *ray-casting* and *projection*.

- In *ray-casting*, rays are cast out from the viewer through the hypothetical screen pixels. The contributions to the ray from points [Lev88] or regions [Sab88, UK88] along it are calculated. This approach is simple to implement, but is liable to aliasing problems and is possibly less amenable to parallel processing [UK88].
- In *projection methods*, each subvolume of the volume is projected onto the hypothetical screen pixels, and its contribution to the pixels affected is calculated [UK88, DCH88, Wes90]. The projection of subvolumes can be more complex to implement, though it has potential for taking advantage of coherence [Wil90a] and is less liable to aliasing.

Another variation in the approaches described is that some methods [DCH88, Sab88, Wes90] treat the samples as *voxels*, i.e. constant data value regions surrounding the grid location, while others [Lev88, UK88, Wil90a] treat the samples as the corners of hexahedral *cells*. A continuous function is generally assumed to pass through the cell as defined by the corner sample values. Treating the volumes as voxels makes fewer assumptions about the data but can lead to rather blocky images. Treating the volumes as cells produces smoother visual images but assumes that the interpolation method is appropriate [WVG90b]. Interpolation can be a significant cost in the algorithm.

3 Ray-Casting Volumes

We have used the ray-casting approach to direct volume rendering, as seeming more amenable for implementation with irregular volumes [Cha90]. The renderer can handle a mixture of volumetric and geometric objects [Lev89] and works with both rectilinear and curvilinear volumes. Implementation was in the object-oriented language C++, taking advantage of the ease with which software can be extended to incorporate new renderable objects [Poh89]. The object-oriented paradigm for design specifies that software objects encapsulate both the data and the functions that process it. Though the potential graphical primitives are diverse and the methods required to process them differ greatly, visualization involves the same three tasks. Each object type provides virtual methods for performing these tasks on its own representation: (1) A list of potentially visible objects on each scanline is updated. (2) For each ray passing though a given pixel, an ordered list (itself an object) of ray-object intersections is calculated by examining the potentially visible objects. (3) The shading contribution of each ray-object intersection to the pixel (also an object, which can composite itself) is calculated.

3.1 Rectilinear Volumetric Primitives

Regular rectangular volumes are handled in a fairly traditional fashion [Lev88]. The volume is positioned and the orientation matrix stored. Intersections of a ray with the volume bounding box are determined by transforming the bounding box by the same transformation that would align the ray with the z axis [Rog85] and then testing each of the six bounding box faces for intersections with the z-axis. The entry and exit intersection points, if any, are added to the z-ordered intersection list. The starting and ending points of the ray-object intersection list are mapped back to grid values in the original array, using the inverse of the transformation matrix. Sample values are acquired at user-specified equidistant steps using one of three user-specified interpolation methods (see Section 4) and used to index into color and opacity transfer function tables [UK88]. The contribution from each of the samples is composited into the pixel [PD84]. This process ends when the exit point is reached, or the pixel becomes opaque. If an isosurface has been requested as well, a check is made between each pair of sample points to see if the isosurface threshold has been crossed [GO89].

3.2 Geometric Primitives

Treatment of geometric objects (at present, triangles or triangle-meshes) is much like that of rectilinear volumes: the triangles are transformed by the orientation matrix that takes the ray to the z-axis, intersections with the z axis are calculated, and the intersections are added to the ordered intersection list. For shading, the barycentric coordinates of the intersection point with respect to each triangle are computed and used to interpolate the vertex normals to obtain the intersection normal. Lambertian ambient and diffuse shading is used [FDFH90]. The shading contribution is composited into the current pixel with full opacity.

3.3 Curvilinear Volumetric Primitives

The curvilinear volumes consist of both the data sample values and a *computational grid* describing the actual physical location of each sample point [plo89]. In its physical-space coordinate system, the faces bounding the element may be degenerate, non-planar, and non-orthogonal, presenting additional problems. Adding curvilinear volume primitives simply involved designing and implementing structures and methods needed to volume render the new primitive. For purposes of intersection, an element (a six-sided cell) is considered to be bounded by twelve triangles (to ensure planarity). Assuming convexity, two intersections are generated if the ray intersects the element. The intersections are computed using the same technique described for the triangle primitive, and the shading method for an element is like that for a rectilinear volume. Because the number of elements in a curvilinear volume can be quite large, data structures and methods are supplied to support a scan line list and active element list to reduce the number of intersection calculations that must be performed for each ray.

4 Interpolation of Curvilinear Volumes

The curvilinear input grid must be interpolated to produce a rectilinear output grid. (We use the term "interpolation", rather than "resampling" to clearly indicate that the values in the new output grid do not have the same validity as the points in the original sample grid, assuming the underlying

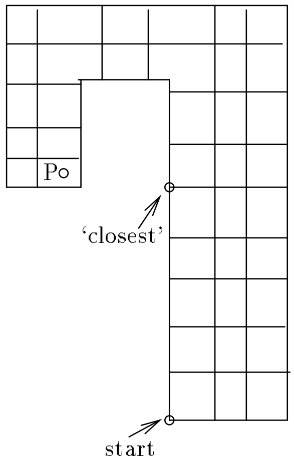

Figure 1: A two-dimensional grid where the greedy grid search algorithm fails

function which was sampled to create the original grid is not used.) The input grid consists of hexahedral cells bounded by eight vertices numbered 0, 1, ..., 7, the coordinates of the i^{th} vertex being (x_i, y_i, z_i), and the associated data value is denoted F_i. The data values are assumed to be the values of an unknown 'smooth' (at least locally) trivariate function $f(x, y, z)$, which we call the *underlying function*. An estimation of f at a point P is denoted as $F(P)$.

Let $P = (x, y, z)$ be the point at which the value of the underlying function is to be estimated. The interpolation schemes examined here all proceed as follows :

1. Find the cell of the input grid within which P lies.
2. Estimate the function value at P based on the known values at the corners of this cell.

To find the input cell that contains the point P, the input grid edges are traversed until a point is reached such that all adjacent points are further away from P. It is assumed that that grid point is closest to P. This greedy algorithm will sometimes fail to produce the correct result (see Figure 1). [1] Once the nearest grid vertex has been located, it is then easy to find which of the (at most) eight cells incident on this vertex contains P.

The three methods explored are *nearest neighbor*, *trilinear interpolation*, and *inverse weighting*. The nearest neighbor method is simplistic : the sample value at a point is estimated to be the same as the value at the nearest point on the original grid. This gives at best a rough estimate of the nature of the volume. The other two methods are described here in more detail.

These schemes are *local* — the function value at a point is estimated based only on points that are 'close' to P. All three ensure that at the corners of a cell the estimated value matches the known function value at that point; i.e., they are interpolation schemes not approximation schemes. Though they ensure C_0 continuity at vertices of the grid, they cannot guarantee even C_0 continuity along interior faces of the grid. This creates a problem: points lying on interior faces but not on interior edges are contained in the two cells that share that face; points on interior edges are contained in the four cells that share that edge. Depending on which cell is considered to contain the point different estimates are obtained.

[1] The point location problem for two-dimensional grids has been well studied and optimal algorithms are available. The three-dimensional case has not received much attention thus far.

4.1 Trilinear Interpolation

The trilinear interpolation approach assumes that at any point $P = (x, y, z)$ within any cell of the input grid the underlying function can be approximated as

$$F(P) = a + bx + cy + dz + exy + fxz + gyz + hxyz$$

where a, b, ..., h are suitable constants. The function f is required to interpolate the known values at the eight cell corners. Substituting in the known function values and locations for the eight cell corners provides eight equations in the eight unknowns a through h. Solving this set of equations provides the cell's trilinear interpolation function.

If the cell is an orthogonally oriented rectangular parallelopiped with vertex 0 at the origin the constants a, ..., h can be computed directly, but this is usually not the case with curvilinear volumes.

Trilinear interpolation is not a very satisfactory method. The underlying function is assumed to vary linearly along lines parallel to the cartesian axes, as a quadratic through planes parallel to the planes defined by these axes, and as a cubic elsewhere. For a rectilinear grid oriented along the cartesian axes, the function varies linearly along edges, as a quadratic across cell faces, and as a cubic along diagonals through the cell. Rotating the cell relative to the underlying frame produces quite a different estimate of function behavior through the cell. It is possible, even with rectilinear cells, to be unable to find a trilinear function defined by the corner values in certain orientations; for example, consider a cell rotated at 45 degrees so that the diagonals lie parallel to the cartesian axes. With irregular grids, it may not be possible to orient the cell so that its edges lie along cartesian axes, and the cell faces may not be planar. Degenerate cases can occur where a trilinear function cannot be fit. More commonly, the function through the cell face may be trilinear and vary depending upon which cell on either side of the face the point is assumed to belong to. Therefore, C_0 continuity cannot be assumed. (For properly-oriented rectangular grids trilinear interpolation does at least provide C_0 continuity along interior faces.)

4.2 Inverse Distance Weighted Interpolation

For distance weighted interpolation schemes [BS84, BL84, Fra82], the value of the underlying function at a point is 'influenced' by its value at other nearby points, influence decreasing with increasing distance from the point.

Let P be a point at which the underlying function $f(x, y, z)$ must be estimated; further, assume that the function value is known at a set of points $P_1, P_2, \ldots, P_n$ that are close to P. Let F_i denote the (known) value of f at P_i, $1 \leq i \leq n$. The value of f at P is estimated as a weighted sum of the various F_i, i.e.,

$$F(P) = \sum_{i=1}^{n} w_i F_i$$

where the weight w_i associated with the point P_i depends inversely on the distance between P and P_i. This inverse dependence ensures that the influence of the point P_i decreases with distance from P. The weights are usually normalized so they sum to one.

In this implementation, the set of nearby points are taken to be the eight vertices of the cell within which the point $P = (x, y, z)$ lies. The particular weighting scheme used was presented in [BS84]. The weights are computed as follows :

$$w_i = \prod_{\substack{k=0 \\ k \neq i}}^{7} [d_k(P)]^2 / \sum_{j=0}^{7} \prod_{\substack{l=0 \\ l \neq j}}^{7} [d_l(P)]^2$$

Here $d_j(P)$ is the Euclidean distance between the j^{th} vertex of the cell and the point P. Note that if P coincides with the j^{th} vertex of the cell, then $w_j = 1$, and $w_i = 0, i \neq j$; therefore, $F(P) = F_j$.

As presented above, the inverse distance weighting scheme will, like trilinear interpolation, not guarantee C_0 continuity along interior faces of the grid. This problem can be minimized in the following (somewhat *ad hoc*) manner: for points on interior faces we use all the vertices of all the cells that contain that point. In fact, it would be preferable to extend the scheme to consider samples beyond a particular cell. Interpolation may produce samples that are more sparse than the original volume, at least locally. Considering only surrounding cells in determining values for interpolated points in effect ignores the contribution of cells in the original volume that do not happen to include an interpolated point. This will always be a problem with very local methods of interpolation. Furthermore, with arbitrary hexahedral cells, the cell vertices may be much further from the new point than vertices belonging to other neighboring cells.

5 Experimental Results

These methods have been explored on a computational fluid dynamics simulation of a "blunt fin" from NASA-Ames Research Center, using the density scalar field for rendering. The original data was interpolated to create two rectilinear volumes: *Interpolate-1* data with approximately the same number of data points as the original, and *Interpolate-2* data with 8 times as many data points. There are three interpolated volumes of each size, differing by whether the interpolation method was nearest neighbor, inverse distance, or trilinear interpolation. For rendering, the same interpolation method was used as for interpolation; e.g., the volume sampled using nearest neighbor methods was rendered using nearest neighbor methods. Three issues come to light in examining the interpolated data (see Table 1).

1. Extraneous data points must be generated which refer to regions outside the warped space, because the rectilinear volume includes the original. Samples in this region can be made transparent, but it does cost memory usage as well as some irrelevent processing time.

2. More serious, the minimum distance between sample points is very small in the warped grid to give most information in regions of greatest interest. However, the interpolated sample points are equidistant along each coordinate axis and the minimum distance between interpolated data points is much larger. In order to make the minimum distance for the interpolate data the same as in the original file, very large interpolated volumes would have to be created. Otherwise, detail is lost, and (in the present implementation) cells may be missed. Because we believe speed is the only justification for interpolating volumes, we choose to accept this and sample more sparsely than the closest original points. Section 6 describes a recently implemented alternate approach.

3. Interpolation moves the maximum and minimum values of the original file toward the mean, thus extreme values are lost.

Table 2 presents some results on interpolation and rendering. Interpolation times are reasonably close for all three methods. As this cost is only incurred once to create the volume, its expense is less important than rendering costs. It could be brought down by better traversal techniques. Rendering was done on an SGI Iris 4D50GT. Note that rendering a rectilinear grid is significantly faster even when the volume involved is 8 times as large. Trilinear coefficients were precalculated for curvilinear volumes; this precalculation took about 110 seconds. By far the predominant cost with curvilinear volumes is intersection testing to determine the location of the ray within the volume. The use of more sophisticated location-finding and coherence will reduce this, but it seems unlikely that rendering curvilinear volumes will ever be as fast as rectilinear volumes.

Images were rendered using 128x128 rays, zooming in on the middle region of the volume. The transfer function mapped low values to green, medium to blue, and high to red. Opacity increased linearly. Pixels were replicated for final pictures.

Figure 2 indicates that the general appearance of all sample volumes is much the same, though the nearest neighbor method presents a decidedly blocky appearance. In the original volume, the cells are smallest near the leading edge of the blunt fin. It is in this region that the effect of the relatively large cells of the interpolated volumes can be seen. The preponderance of blue, and the less amount of red, in interpolated images is due to the shift toward the mean during interpolation. The green along the leading fin edge is due to the renderer estimating color values for cells partly outside the original volume. This could be avoided by having the renderer ignore such cells.

Curvilinear Volume: Nearest Neighbor, Weighted Average, Trilinear Interpolation

Interpolate-1 Volumes: Nearest Neighbor, Weighted Average, Trilinear Interpolation

Interpolate-2 Volumes: Nearest Neighbor, Weighted Average, Trilinear Interpolation

Figure 2: Comparison of Images

Blunt Fin	Original Data	Interpolate 1	Interpolate 2
Resolution	40x32x32	40x32x32	80x64x64
Number Samples	40,960	40,960	327,680
Smallest Distance	0.019	0.18	0.09
Greatest Distance	3.32	0.57	0.28
Min/Max Values			
Original	0.19/4.98	-	-
Nearest Neighbor	-	0.38/3.48	0.34/4.62
Inverse Distance	-	0.38/3.26	0.36/4.36
Trilinear Interpolation	-	0.40/3.26	0.36/4.31

Table 1: Volume Data Characteristics

Blunt Fin	Nearest Neighbor	Inverse Weights	Trilinear Interp.
Interpolation Time			
–Interpolate 1	114.7	128.2	230.4
–Interpolate 2	936.7	1030.5	1061.5
Rendering Time			
–Original Data	700.0	796.2	807.8
–Interpolate 1	94.1	189.7	64.7
–Interpolate 2	152.6	306.4	97.7

Table 2: Timing Tests (user/system seconds)

6 Hierarchical Extensions

Use of a hierarchical rectilinear grid avoids many of the problems of interpolating from curvilinear volumes with irregular sampling densities. We are exploring the use of an octree data structure for the interpolated volume. To create the octree, sample points in the original curvilinear grid are added one by one to the (originally empty) octree. Whenever a particular partition of the octree contains more than some user-specified number of curvilinear data samples, that node is split into eight subnodes. When all the curvilinear samples are added to the octree, an interpolation step calculates the estimated data value at the eight corners of each octree partition. This volume can then be rendered using a hierarchical volume ray-tracer. The coherent projection approach [Wil90a] is also being extended to render hierarchical volumes. Related work using octrees for isosurface generation is [WVG90a].

Alternatively, to avoid interpolation errors in creating the octree, it would be possible to use the octree together with the original curvilinear volume and render using values from the original sample values.

7 Volume Visualization of Turbulence

Turbulence is the irregular regime of fluid motion which randomly varies with space and time. In spite of more than a century of research, it remains an unsolved problem of modern physics. The study of coherent structures of a turbulent flow field and direct numerical simulation of turbulence [MK86, Spa88] are currently the mainstream approach to turbulent research. Direct numerical simulation of the turbulent boundary layer over a flat plate by Spalart [Spa88] provides flow data at 9.4 million grid points for 104 time steps. The resulting 54 gigabytes of turbulence database contains values for three velocity components, three vorticity components, and pressure at each node. Volume visualization is applied to a 65 x 50 x 55 sub-volume of the data set [RKS89] using an implementation of VBUFFER [UK88] on a Stardent GS-1000.

Direct volume rendering of space filling 3-D data, such as fluid flow data, characteristically results in "cloud-like or fuzzy" images. However, by applying additional gradient shading to the volume-rendered turbulent flow data, crisp and solid-looking images can be obtained. The additional steps are crucial to visualization of current data since the final image should contain a clear depiction of the turbulent flow structures. Figure 3 represents the vertical component of velocity solid-texture-mapped onto the streamwise velocity component. Applying gradient shading has resulted in a crisp image of vortex structures in a turbulent flow where contributions from ambient, diffuse, specular, and glossiness are also added to the image.

7.1 Turbulent Flow Structures: "The hairpin vortex"

Coherent structures in turbulence have been identified by researchers for some time [KRSR67]. Evidence for existence of a basic vortex structure, called the "hairpin or horseshoe

Figure 3: Volume Visualization of Turbulence - Revealing the Hairpin Vortex Structure

vortex" has been found in experiments [HB81] and in numerical solutions [KM86]. The hairpin (or horseshoe) vortices are the primary agents of several turbulent models. Direct visualization of these and other vortical structures has been in general difficult. Applying the volume visualization method outlined above has resulted in a clear depiction of the hairpin vortex and other structures in the turbulent flow over a flat plate. It is possible to study the growth of the hairpin vortex at different streamwise locations since the structure is easily recognizable. Downstream and to the right of the bounding walls is populated by a number of structures; they include a tornado-like horseshoe vortex with the head turned down, also found by [BR71].

The turbulent flow can be viewed as a tangle of vortices undergoing various types of interaction [WH90]. Volume visualization of the coherent structures provides the basic tool to conduct "numerical flow visualization" experiments revealing the embedded details. The combination of texture-mapping and gradient-shading results in an image clearly superior to a traditionally volume-rendered images. Flow structures in the image become evident only after the gradient shading has been applied. This representation, unlike geometrical rendering techniques, preserves full integrity of the original dataset. It is thus more readily adaptable to analysis, and extraction of quantitative information. A study of the temporal development of turbulent structures through animation of time sequences (in preparation) can provide a useful tool to enhance our understanding of vortex formation, evolution, and dynamics.

8 Conclusions

Evidence suggests that reinterpolation to a rectilinear volume provides an acceptable method for faster direct volume rendering of curvilinear volumes. However, evidence also indicates that the possibility of erroneous images is a serious problem, particularly when regular sampling is used. This must be taken into account in interpreting the images. Interpolation to an adaptively subdivided hierarchical volume holds promise of avoiding the worst of these problems. Future work to improve the speed of intersection algorithms may make methods that work directly on the original data more attractive.

Acknowledgements

This work was supported by a NAS NASA-Ames Research Center Consortium Agreement. We appreciate the use of NASA-Ames data sets for this study and wish especially to thank Drs. Thomas Lasinski and Sam Uselton for their help. This work was also supported by a State of California Micro-Electronics Grant. We also wish to thank Silicon Graphics Inc., Digital Equipment Corporation, and Sun for their donations of equipment and Stardent Computers for their loan of a Titan.

References

[BL84] R. E. Barnhill and F. F. Little. Three- and four-dimensional surfaces. *Rocky Mountain Journal of Mathematics*, 14(1):77 – 102, 1984.

[BR71] G.L. Brown and A. Roshko. *AGARD-CP-93*, 23, 1971.

[BS84] R. E. Barnhill and S. E. Stead. Multistage trivariate surfaces. *Rocky Mountain Journal of mathematics*, 14(1):103 – 118, 1984.

[Cha90] Judith Challinger. Object-Oriented Rendering of Volumetric and Geometric Primitives. Master's thesis, University of California, Santa Cruz, 1990.

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

[FDFH90] James D. Foley, Andies Van Dam, Steven Feiner, and John Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, Reading, Mass., 2 edition, 1990.

[Fle88] C. A. J. Fletcher. *Computational Techniques for Fluid Dynamics*. Springer-Verlag, 1988.

[Fra82] R. Franke. Scattered data interpolation : Tests of some methods. *Mathematics of Computation*, 38(157):181 – 199, 1982.

[GO89] David S. Goodsell and Arthur J. Olson. Molecular applications of volume rendering and 3-D texture maps. In *Volume Visualization Workshop*, pages 27–31, Chapel Hill, NC, May 1989. Dept. of Computer Science, University of North Carolina.

[HB81] M. R. Head and P. Bandyopadhyay. *J. of Fluid Mechanics*, 107:297, 1981.

[KM86] J. Kim and P. Moin. *J. Fluid Mechanics*, 30:741, 1986.

[KRSR67] S. J. Kline, W. C. Reynolds, F. A. Schraub, and P. W. Runstadler. *J. Fluid Mechanics*, 30:741, 1967.

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

[Lev89] Marc Levoy. *Display of Surfaces From Volume Data*. PhD thesis, The University of North Carolina at Chapel Hill, 1989.

[MK86] P. Moin and J. Kim. *J. of Fluid Mechanics*, 155:61, 1986.

[PD84] Thomas Porter and Tom Duff. Compositing digital images. *Computer Graphics (ACM Siggraph Proceedings)*, 18(3):253–260, July 1984.

[plo89] PLOT3D User's Manual, 1989.

[Poh89] Ira Pohl. *C++ for C Programmers*. Benjamin/Cummings Publishing, California, 1989.

[RKS89] S. K. Robinson, S. J. Kline, and P. R. Spalart. Technical Report TM-102191, NASA-Ames Research Center, Moffett Field, CA, 1989.

[Rog85] David F. Rogers. *Procedural Elements for Computer Graphics*. McGraw-Hill, Inc., 1985.

[Sab88] Paolo Sabella. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics (ACM Siggraph Proceedings)*, 22(4):51–58, July 1988.

[SN89] Peter Shirley and Henry Neeman. Volume visualization at the center for supercomputing research and development. In *Volume Visualization Workshop*, pages 17–20, Chapel Hill, NC, May 1989. Dept. of Computer Science, University of North Carolina.

[Spa88] P. R. Spalart. *J. of Fluid Mechanics*, 187:61, 1988.

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

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

[WH90] J. M. Wallace and F. Hussain. *Appl. Mech. Rev*, 43:S203, 1990.

[Wil90a] Jane Wilhelms. A coherent projection approach for direct volume rendering. Technical Report UCSC-CRL-90-38, CIS Board, University of California, Santa Cruz, 1990.

[Wil90b] Jane Wilhelms. Visualizing sampled volume data. In Nadia Magnenat-Thalmann and Daniel Thalmann, editors, *Scientific Visualization and Graphics Simulation*. John Wiley and Sons Limited, 1990.

[WVG90a] Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation. Technical Report UCSC-CRL-90-28, CIS Board, University of California, Santa Cruz, 1990. Extended abstract to appear in ACM Volume Visualization Workshop 1990.

[WVG90b] Jane Wilhelms and Allen Van Gelder. Topological ambiguities in isosurface generation. Technical Report UCSC-CRL-90-14, CIS Board, University of California, Santa Cruz, 1990. Extended abstract to appear in ACM Volume Visualization Workshop 1990.