

Short Paper

A Volume Rendering Algorithm Using an Analytic Approach for Curvilinear Volumes

CHYI-CHENG LIN AND YU-TAI CHING*

Technology and Service Department

Institute for Information Industry

Taipei, Taiwan 116, R.O.C.

**Department of Computer and Information Science*

National Chiao Tung University

Hsinchu, Taiwan 300, R.O.C.

A new approach is presented to visualize curvilinear volume data produced using the CFD or FEM Methods. *Ray casting* methods are effective and renders images with good quality. However, these methods suffer from a tremendous number of computations and can not deal with the curvilinear volume efficiently. The marching cubes method is a popular approach when the ambiguity problem reduces the reliability of the rendered image. We propose a new method to solve these problems. Our method is a cell-by-cell process which constructs surfaces using trilinear functions. Several speed up techniques are proposed to reduce the number of computations in our algorithm. In addition to this, error analysis and error minimization are also discussed. Our method is also easy to implement parallelized.

Keywords: computational fluid dynamics(CFD), volume visualization, curvilinear data, isosurface approximation, analytic approach

1. INTRODUCTION

Volume rendering for irregular data using the Computational Fluid Dynamic(CFD) or Finite Element Method(FEM) has become important recently. When dealing with a data volume, some types of data distribute uniformly but some do not. According to the data distribution, three types of data are involved: *rectilinear*, *curvilinear*, and *unstructured* data [10, 27]. *Rectilinear data* consists of a set of box-like hexahedron cells, and if the cells are all identical, we say the data are *regular*. *Curvilinear data* can be thought of as resulting from a wrapping of rectilinear data, and *unstructured* data consist of a set of arbitrarily shaped cells. In this paper, we mainly address *rectilinear* and *curvilinear* data.

Received December 26, 1997; accepted September 8, 1998.
Communicated by Zen Chen.

Generally speaking, rectilinear data is easier to deal with than curvilinear data. The first problem with *curvilinear* data is that the traversing order is not explicit. The traversing order is needed to determine the *sample points* in methods like ray-casting algorithms [11, 12, 15]. One should define the view-dependent traversal order of cells either in a Front-to-Back or Back-to-Front way. The second problem in rendering curvilinear grids is to determine the cell where the sampling points reside. If the cells are all regular(cubes), a floor/ceiling function can easily locate the cell where a sample point belongs. However, if the cell are curvilinear, how to locate the sample point is another nontrivial computational problem.

To visualize curvilinear data, many methods have been proposed in recent years [11, 14, 17, 21-23, 29]. *Isosurface extraction* methods are some of the most famous methods. Some of them connect 2D contours, which are either curve functions or line segments, into 3D graphics primitives to construct isosurfaces. Lorensen *et al* proposed the *marching-cube* method which uses polygon primitives to extract isosurfaces [14, 27]. The marching-cube algorithm is easily implemented, but it has an ambiguity problem which causes details in the volume to not be precisely presented. *Direct volume rendering* renders images from input data without using geometric primitives or functions. *Ray casting* is the method most frequently used in direct volume rendering algorithms. The advantage of direct volume rendering is that everything in the volume contributes to the final rendered image. However, it suffers from a tremendous computational cost.

Most of those algorithms handle rectilinear data in an easier way than when they are applied to curvilinear grids. For example, the sorting and point locating problem is harder to deal with in the ray casting method.

We propose a new method to solve these problems. Firstly, a *trilinear interpolant* is defined for each cell. Whenever a threshold is defined, an *isosurface* is constructed by converting the interpolant into a trilinear function. The *isosurface* is a continuous surface of degree 3 on which every point has equal value. Our construction of isosurfaces prevents the ambiguity problem for some methods by producing polygons to approximate the isosurface. It also resolves the *topology inconsistency* [16] problem when scaling must be applied to regions or features of interest. Based on the definition of isosurfaces, each cell is investigated to compute the intersection points of rays and isosurfaces. Since the solution for the ray/isosurface intersections can be formulated as a function of degree 3, closed form solutions of this equation can be derived easily. Thus, this method guarantees that the intersection will be found (if it exists) even in the case of thin objects [13].

To extract and render isosurfaces in cells, a *Front-to-Back* method is used when cells are traversed. Cells are sorted in topological order and investigated one-by-one to compute the isosurfaces inside. All the isosurfaces are then projected onto a frame buffer, which is kept to prevent redundant projection. An improvement in speed using this algorithm is obtained by using a technique for frame buffer maintenance. In this paper, we use a *union-find* algorithm to reduce the frame buffer maintenance time.

Computation of the projection of isosurface generation will be presented in section 2.1. Traversing the volume and computing the ray/isosurface intersection are discussed in section 2.2. The discontinuity problem is discussed in section 2.3. The frame buffer maintenance technique is presented in section 3. The result for curvilinear grids of Computational Fluid Dynamics and some visualization strategies are presented in section 4.

2. ISOSURFACE GENERATION

2.1 Trilinear Interpolant for Curvilinear Data

2.1.1 Some definitions

Curvilinear grids are defined by a set of ordered pairs (p_k, v_k) , where p_k are positions of nodes and v_k are nodal values. Without loss of generality, the number of nodal points around a *cell* is 8(a hexahedron) or 4(a tetrahedral), and they are connected by edges. According to all the pairs of (p_k, v_k) inside each cell, the density function F is defined. Volume rendering of curvilinear grids extracts the *isosurfaces* of a specific *threshold* and renders them in an image. That is, a volume rendering algorithm searches the isosurface $F = t$ inside volume cells with a value t defined as the threshold.

The quality of a rendered image is related to the definition of the function F . Although an F of higher degree has a smooth distribution, it also requires a tremendous number of computations. On the other hand, linear functions(degree=1) describe an isosurface using piecewise polygons and sometimes ignore details. Recently, the *ray-casting* method has been employed in trilinear interpolation(degree=3) to render images with good-quality. In this paper, we apply the *tri-linear function* to interpolate the values inside one cell. Since a tri-linear function has a closed-form solution, it will be useful when we calculate the ray/surface intersection.

A trilinear function $F(x,y,z)$ varies linearly on the three *interpolation axes* x,y,z . The result is that values inside the space bounded by x,y,z vary tri-linearly. Usually, cell axes x,y,z are orthogonal to each other. However, in this paper, three adjacent edges are selected as the *interpolation axes* and the one adjoining them is the local coordinate origin(see Fig. 1). If the cell is a *hexahedron*, one node p_i is selected as the coordinate origin. Three edges adjoining p_i are selected as the interpolation axes. The interpolation function is derived as follows. Let H be a *regular* hexahedron; then, the density function F_H of this cell can be formulated as

$$F_H = axyz + bxy + cyz + dxz + ex + fy + gz + h,$$

where $a = v_1 + v_2 + v_4 + v_7 - v_0 - v_3 - v_5 - v_6$, $b = v_0 + v_6 - v_2 - v_4$, $c = v_0 + v_3 - v_1 - v_2$, $d = v_0 + v_5 - v_1 - v_4$, $e = v_4 - v_0$, $f = v_2 - v_0$, $g = v_1 - v_0$, and $h = v_0$ (see Fig. 1). However, if H is *curvilinear*, the eight coefficients must be modified. The correct coefficients satisfy the *boundary constraints* of the eight (v_k, p_k) pairs, i.e,

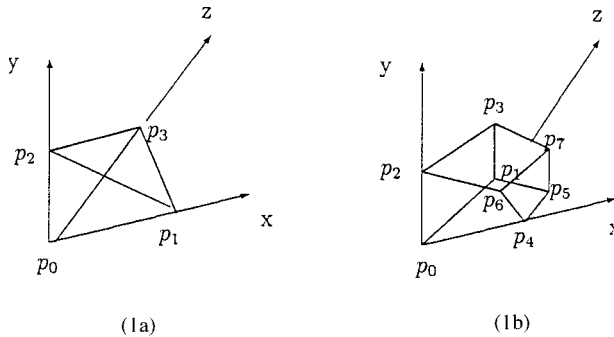


Fig. 1. The interpolation axes of a tetrahedron(1a) and a hexahedron(1b).

$$F_H(p_k) = v_k,$$

where $k = 0 \dots 7$.

2.1.2 Curvilinear grids

Let p_0 be the coordinate origin, and let the interpolation axes be $\overline{p_0 p_1}$, $\overline{p_0 p_2}$ and $\overline{p_0 p_4}$. Suppose all the nodal points are transformed into the local coordinate system and are expressed as $(p_{i,x}, p_{i,y}, p_{i,z})$, $i = 0 \dots 7$.

The target interpolation function F_H is expressed as follows:

$$F_H = J_0xyz + J_1xy + J_2yz + J_3xz + J_4x + J_5y + J_6z + J_7.$$

The coefficients $J_0 \dots J_7$ of function F_H are determined by eight constraints in three steps. Firstly, the three constraints $F_H(p_k) = v_k$, $k = 1, 2, 4$ enable the coefficients J_4 , J_5 , J_6 and J_7 to be derived (note that $\overline{p_0 p_1}$, $\overline{p_0 p_2}$ and $\overline{p_0 p_4}$ are now unit vectors). That is,

$$J_4 = e, J_5 = f, J_6 = g, J_7 = h.$$

In the second step, the constraints 3, 5, and 6 are verified. Each constraint represents a 2-dimensional polygon. That is, each constraint is a degree 2 function which interpolates border faces $\overline{p_0 p_4 p_6 p_2}$, $\overline{p_0 p_2 p_3 p_1}$ or $\overline{p_0 p_4 p_5 p_1}$. Each face is a quadrilateral, and the interpolant is bilinear. For example, the bilinear function to interpolate the quadrilateral $\overline{p_0 p_4 p_6 p_2}$ (the xy plane in Fig. 1b) must be derived by constraint 6:

$$v_6 = J_1 p_{6,x} p_{6,y} + (v_4 - v_0) p_{6,x} + (v_2 - v_0) p_{6,y} + v_0,$$

and J_1 will be expressed as

$$J_1 = \frac{v_6 - v_0 - (v_4 - v_0) p_{6,x} - (v_2 - v_0) p_{6,y}}{p_{6,x} p_{6,y}}.$$

Similarly, the other two constraints must be interpolants for the quadrilateral $p_0 p_4 p_5 p_1$ and $p_0 p_2 p_3 p_1$. Thus J_2 and J_3 are obtained:

$$J_2 = \frac{v_3 - v_0 - (v_2 - v_0) p_{3,y} - (v_1 - v_0) p_{3,z}}{p_{3,y} p_{3,z}},$$

$$J_3 = \frac{v_5 - v_0 - (v_4 - v_0) p_{5,x} - (v_1 - v_0) p_{5,z}}{p_{5,x} p_{5,z}}.$$

Finally, constraint 8 will enable the coefficient J_0 to be obtained. That is,

$$J_0 = - \frac{J_1 p_{7,x} p_{7,y} + J_2 p_{7,y} p_{7,z} + J_3 p_{7,x} p_{7,z} + J_4 p_{7,x} + J_5 p_{7,y} + J_6 p_{7,z} + J_7}{p_{7,x} p_{7,y} p_{7,z}}.$$

It can be easily seen that $F_H(p_k) = v_k$ holds for $i = 0 \dots 7$, and that F_H is a function of degree three.

On the other hand, the cell may be a *tetrahedron*. Let the tetrahedron be denoted as T . In this case, we only need four constraints $F_T(p_k) = v_k$, where $k = 1..4$. If $\overline{p_0 p_1}$, $\overline{p_0 p_2}$, and $\overline{p_0 p_3}$ of t (see Fig. 1a) are selected as interpolation axes, then the interpolant of T can be expressed as

$$F_T = J_4 x + J_5 y + J_6 z + J_7.$$

However, this is just a special case of the interpolant F_H , and the coefficients J_4 , J_5 , J_6 and J_7 can be easily derived.

By the definition of F_H or F_T , cell-by-cell processing for volume rendering is feasible. The cells are traversed from front to back, and the isosurface $F_H = t$ or $F_T = t$ is computed and rendered if it exists (see the section below). Note that F_H or F_T can be computed in the preprocessing step.

2.1.3 Discontinuity

Although the computational cost of the trilinear function is less than that of a higher-order function, then cell-by-cell processing causes a problem of *discontinuity*. *Discontinuity* exists when isosurfaces of neighboring cells are not continuous. By the definition of the trilinear interpolant, this will not happen when the cells are regular. For a regular cell, the value of F_H or F_T varies linearly on the edges. Thus, if an isosurface intersects an edge $\overline{p_i p_j}$, then neighboring cells will have the same linear function on edge $\overline{p_i p_j}$. However, the function value may vary non-linearly on cell edges for a curvilinear cell. Since the non-linearity is related to the definition of F for each cell, the isosurfaces of neighboring cells may not be connected completely. In this case, isosurfaces on edge $\overline{p_i p_j}$ will be discontinuous and produce errors (gaps). Minimization of the error will be discussed in section 2.3.

2.2 Cell Traversing and Computation of Ray/isosurface Intersections

In our method, Front-to-Back projection is used, and a traversal order for the cells must be decided. One way to traverse the cells is to use an *element-node* table and an *element adjacency* table [6]. Each time, the next cell is located from this table, and processing goes on recursively. Another way to define the traversal order is by sorting operations. In recent years, several efficient sorting techniques have been developed, and the sorting time can be reduced so that it is linear [19]. Each time the threshold is changed, a new traversing order must be defined. All isosurfaces in cells are computed and rendered again. The ray/isosurface intersections of a tetrahedron cell can be easily computed since the interpolation function F_T is linear. Suppose the ray is parameterized as $(x_0 + u_0 t, y_0 + u_1 t, z_0 + u_2 t)$, where (u_0, u_1, u_2) is the direction of the ray. After substituting this into equation (1), the resulting equation is

$$F_T = (v_1 - v_0)(x_0 + u_0 k) + (v_2 - v_0)(y_0 + u_1 k) + (v_3 - v_0)(z_0 + u_2 k) = t,$$

and the solution is

$$k = \frac{v_{iso} - (v_1 - v_0)x_0 - (v_2 - v_0)y_0 - (v_3 - v_0)z_0}{(v_1 - v_0)u_0 + (v_2 - v_0)u_1 + (v_3 - v_0)u_2}.$$

However, for a hexhedron cell, we extract the isosurface using the method we proposed in [13]. Substituting the parameter of the ray-vector into equation 1, we have

$$g = M_1 \cdot k^3 + M_2 \cdot k^2 + M_3 \cdot k - \Delta v, \quad (1)$$

where

$$\begin{aligned} M_1 &= J_0 u_0 u_1 u_2, \\ M_2 &= u_0 u_1 z_0 J_0 + u_0 u_2 y_0 J_0 + u_1 u_2 x_0 J_0 + u_0 u_1 J_1 \\ &\quad + u_0 u_2 J_2 + u_1 u_2 J_3, \\ M_3 &= u_0 f_{x_0} + u_1 f_{y_0} + u_2 f_{z_0}, \end{aligned} \quad (2)$$

and

$$\begin{aligned} f_{x_0} &= \frac{\partial V(x_0, y_0, z_0)}{\partial x} \\ &= Ay_0 z_0 + By_0 + Dz_0 + E, \\ f_{y_0} &= \frac{\partial V(x_0, y_0, z_0)}{\partial y} \\ &= Ax_0 z_0 + Bx_0 + Cz_0 + F, \\ f_{z_0} &= \frac{\partial V(x_0, y_0, z_0)}{\partial z} \\ &= Ax_0 y_0 + Cy_0 + Dx_0 + G, \end{aligned} \quad (3)$$

and Δv is the difference between the isovalue and the value of front-face inter-section point.

The function g is cubic polynomial, and there is a closed form solution of k for $g = 0$. Solving k for $g = 0$ gives the point $(x_0 + u_0 \cdot k, y_0 + u_1 \cdot k, z_0 + u_2 \cdot k)$, which is the intersection point of the ray/isosurface.

2.3 Isosurface Continuous Consideration

2.3.1 Some definitions

When a trilinear function is defined for each curvilinear cell, isosurfaces of neighboring cells may not connect completely. We define the *interpolation error* as the discontinuity between neighboring cells. Recall that in selection of the coordinate system, there are eight possible interpolants for F_H and four possible interpolants for F_T . However, the cor-

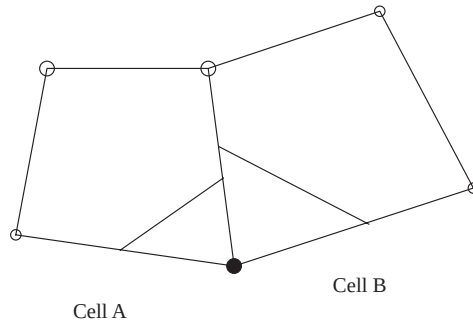


Fig. 2. Isosurfaces of cells A and B are discontinuous on shared edges. (There is an interpolation error.)

rect values for each cell face are unknown, so the interpolation error may be defined on the cell edges. An efficient way to define the interpolation error is to use the value difference between the linear interpolant and trilinear interpolant on that edge (see Fig. 2).

That is, the *interpolation error* on edge $\overline{p_i p_j}$, denoted as E_{ij} , is defined by $F - L$, where $F = F_T$ or F_H , and L is the linear function on edge $\overline{p_i p_j}$. By this definition, we can further define an *edge error*, denoted as EE_{ij} , as the total of the interpolation errors for all points on the edge $\overline{p_i p_j}$. This can be expressed by integration of the interpolation error for all edge points. Finally, we define the *cell error*, denoted by CE , as the addition of edge errors of a cell.

2.3.2 Two propositions

From the above definitions, we have the following two propositions.

Proposition 1: For a tetrahedron, $EE_{ij} = 0$ for all edges and $CE = 0$.

Proof: Suppose a set of edges $\overline{p_0 p_1}$, $\overline{p_0 p_2}$, and $\overline{p_0 p_3}$ of tetrahedral cell $p_0 p_1 p_2 p_3$ are selected as interpolation axes x, y, z . The interpolation function defined on edge $\overline{p_1 p_2}$ can be expressed as the function F_T on the line $\overline{p_1 p_2}$, which is parameterized as $(1-t, t, 0)$. That is,

$$\begin{aligned} F_T(t, 1-t, 0) &= (v_1 - v_0)(1-t) + (v_2 - v_0)t + v_0 \\ &= (v_2 - v_1)t. \end{aligned}$$

F_T is equal to the linear interpolant on edge $p_1 p_2$; thus, $F_T - L = L - L = 0$. Since all EE_{ij} are zero for $i, j = 0 \dots 3$, CE equals zero.

Proposition 2: EE_{ij} of a hexahedron is bounded by

$BE_{ij} = \frac{1}{2}v_i(|P| + |Q| + |R|) + \frac{1}{3}|PQ| + |PR| + |QR|$, where $P = p_{j,x} - p_{i,x}$, $Q = p_{j,y} - p_{i,y}$, $R = p_{j,z} - p_{i,z}$. Note that we restrict $|p_{i,x}|, |p_{i,y}|, |p_{i,z}| \leq 1$.

Proof: In a hexahedron H , assume three axes are selected as *interpolation axes*. In general, assume another axis $\overline{p_i p_j}$ is parameterized by $((p_{j,x} - p_{i,x})t + p_{i,x}, (p_{j,y} - p_{i,y})t + p_{i,y}, (p_{j,z} - p_{i,z})t + p_{i,z}) = (Pt + p_{i,x}, Qt + p_{i,y}, Rt + p_{i,z})$, where $0 \leq t \leq 1$.

$$\begin{aligned} E_{ij} &= F_H - L \\ &= F_H(Pt + p_{i,x}, Qt + p_{i,y}, Rt + p_{i,z}) - ((v_j - v_i) + v_i) \\ &= J_0(PQRt^3 + (PQp_{i,z} + QRp_{i,y} + PQp_{i,x})t^2 + \\ &\quad (Pp_{i,y}p_{i,z} + Qp_{i,x}p_{i,z} + Rp_{i,x}p_{i,y})t + p_{i,x}p_{i,y}p_{i,z}) \\ &\quad + J_1(PQp_{i,x}p_{i,y} + Pp_{i,y} + Qp_{i,x} + p_{i,x}p_{i,y}) \\ &\quad + J_2(QRp_{i,y}p_{i,z} + Qp_{i,z} + Rp_{i,y} + p_{i,y}p_{i,z}) \\ &\quad + J_3(PRp_{i,x}p_{i,x} + Pp_{i,z} + Rp_{i,x} + p_{i,x}p_{i,z}) \\ &\quad + J_4(Pt + p_{i,x}) + J_5(Qt + p_{i,y}) + J_6(Rt + p_{i,z}) + J_7 \\ &\quad - ((v_j - v_i)p_{i,x}t + v_i). \end{aligned} \tag{4}$$

And this can be rearranged as

$$\begin{aligned}
 E_{ij} = & (J_0 PQRt^3 + (J_1 PQ + J_2 QR + J_3 PR)t^2 + (J_4 P + J_5 Q + J_6 R)t + J_7) \\
 & + (J_0 p_{i,x} p_{i,y} p_{i,z} + J_1 p_{i,x} p_{i,y} + J_2 p_{i,y} p_{i,z} + J_3 p_{i,z} p_{i,x} + J_4 p_{i,x} + J_5 p_{i,y} + J_6 p_{i,z} + J_7) \\
 & + P(J_0 p_{i,y} p_{i,z} + J_1 p_{i,y} + J_3 p_{i,z})t \\
 & + Q(J_0 p_{i,x} p_{i,z} + J_1 p_{i,x} + J_2 p_{i,z})t \\
 & + R(J_0 p_{i,x} p_{i,y} + J_2 p_{i,y} + J_3 p_{i,x})t \\
 & + (PQp_{i,z}t^2 + PRp_{i,y}t^2 + QRp_{i,x}t^2) \\
 & - ((v_j - v_i)t + v_i) - J_7.
 \end{aligned} \tag{5}$$

Thus, EE_{ij} can be expressed as

$$EE_{ij} = \int_0^1 E_{ij} dt,$$

where t is the parameter on edge $\overline{p_i p_j}$. We will show that EE_{ij} is bounded if $|p_{i,x}|, |p_{i,y}|, |p_{i,z}| \leq 1$. After integration, term 1 of (5) is less than v_j . Term 2 of (5) is equal to v_i . And terms 3, 4 and 5 of (5) are less than $\frac{1}{2}|P|v_i, \frac{1}{2}|Q|v_i, \frac{1}{2}|R|v_i$, respectively. Term 6 of (5) is less than $\frac{1}{3}(|PQ| + |PR| + |QR|)$:

$$\begin{aligned}
 EE_{ij} & \leq v_j + (v_i - J_7) + \frac{1}{2}|P|v_i + \frac{1}{2}|Q|v_i + \frac{1}{2}|R|v_i + \\
 & \quad \frac{1}{3}|PQ|, \frac{1}{3}|QR|, \frac{1}{2}|PR| - \frac{1}{2}(v_j - v_i) - v_i \\
 & \leq \frac{1}{2}v_i(|P| + |Q| + |R|) + \frac{1}{3}(|PQ| + |PR| + |QR|) \\
 & = BE_{ij}.
 \end{aligned} \tag{6}$$

If the edge $\overline{p_i p_j}$ is one of the interpolation axes, the error will be zero, and we need not approximate the error, i.e., $EE_{01} = EE_{02} = EE_{03} = 0$.

The other nine axes will each have a E_{ij} and a EE_{ij} . According to the definition of the cell error, CE is defined and bounded by

$$CE = \sum_{i,j} EE_{ij} \leq \sum_{i,j} BE_{ij}.$$

Using BE_{ij} as an error approximation, a global optimum problem can be described as a graph optimization problem. Each cell is represented as a vertex in this graph. Each cell edge is represented as an edge in the graph with weight equal to BE_{ij} . The global optimum problem is to find the minimum cost graph. However, it is still an open problem in our research.

Based on the error approximation, a naive way to reduce the interpolation error is just to subdivide the cell into two smaller ones on the edges with greater errors. Another solution is to select one coordinate system which minimizes the interpolation error. For the sake of efficiency, we find the local minimum solution which will minimize the error for a cell. That means we will find one coordinate system to reduce CE as much as possible.

This procedure is as follows. For each vertex, we select the three interpolation edges adjacent to it. Then, the other nine $EE_{i,j}$ are computed and added together. In the final step, we choose one of the nodes as the coordinates origin and adjacent edges as interpolation axes. Note that minimization also can be accomplished in the preprocessing step.

3. FRAME BUFFER MAINTENANCE

The volume cells are sorted Front-to-Back, and this ordering ensures that we process the cells closer to the frame buffer earlier. The cells reported later thus might not be visible from the view point. By maintaining the frame buffer, we can reduce the cost of processing cells which are not visible. First, the scan-conversion algorithm is applied to the areas projected by those cells with isosurfaces inside. Without loss of generality, we assume that the scan-lines are parallel to the X axis. Let (i, j, Y) denote a horizontal line segment from points (i, Y) to (j, Y) in the frame buffer. Our algorithm can efficiently determine whether a scan-line (i, j, Y) is totally invisible or not.

Let the size of the frame buffer be $H \times L$. A point (k, m) in $(0, L, m)$, $0 \leq m \leq H$ is occupied if the point (k, m) in the frame buffer has been rendered already. We say (i, j, m) is a set of *occupied points*, denoted as S_o , if and only if (k, m) , $i < k < j$ are occupied and (i, m) and (j, m) are the left-most and right-most points in S_o . An unoccupied set S is similarly defined.

Assume that there are r occupied sets in $(0, L, m)$. The status of the points on $(0, L, m)$ can be represented as a string $S^1, S_o^1, S^2, S_o^2, \dots, S^r, S_o^r$. Given a scan-line (p, q, m) , we consider the following cases:

1. if both p and q are in the same occupied set, S_o^t , then all points on the line segment (p, q, m) are occupied ((p, q, m) is not visible);
2. some points on (p, q, m) are visible if
 - (a) $p \in S_o^t$ and $q \in S_o^{t'}$, then (p, q, m) crosses at least an empty set,
 - (b) p, q , or both, are in an unoccupied set.

In the second case, computing the intersections between the isosurface and the rays shooting from the points on (p, q, m) is required. In case 2a, if all rays shooting from (p, q, m) hit the isosurface, then S_o^t and $S_o^{t'}$ should become one set.

This is a typical *union-find* operation [1-3]. The *find* operation is used to determine which sets p and q belong to, and the *union* operation is used to unify two sets. If *forest representation* is used, a single *union* operation takes constant time. If both *weighting rules* for *union* and *path compression* for *find* are applied, the *amortized cost* of a single *find* is $O(\alpha(F, H))$, where F is the number of *find* operations and H is the number of points in a row. The function $\alpha(F, H)$ is the inverse of Ackermann's function, which is less than 4 for practical applications. That is, we can determine if a scan-line is visible or not in almost constant time.

This algorithm works for any Front-to-Back projection with a scan conversion method if the programmers have to maintain the frame buffer by themselves. Note that this algorithm needs extra memory space equal to the size of the frame buffer.

4. RESULTS

Our algorithms for rendering the curvilinear data is summerized as follows.

Algorithm curvilinear-rendering();

1. Sort all cells in a Front-to-Back order by topological sort.
2. For a cell C :
 - Compare the eight nodal values v_i with the threshold t ; if nodal values are all greater or all less than t , then break; else the optimum interpolation function is used (it is computed in the preprocessing step).
3. Scan conversion is carried out for the projection area of C .
4. For each scanline, use the Union-Find algorithm to decide if scanline projection is needed. If yes, compute the ray/isosurface intersections.
5. For each intersection point, perform the normal vector estimation process and Phong Shading.

Our method can be easily extended to visualize isosurfaces for several thresholds in a image. Increasing the number of thresholds will increase the perception of data distribution. To represent the intensities, the color pallete used for the diffuse term of Phong Shading is uniformly and automatically selected from the RGB color system to show the differences between those surfaces.

Our method was implemented and tested on several sets of data. Two sets of CFD data were obtained from a blunt fin which some fluids flowed through. The two sets of data were measured on the curvilinear grids illustrated in Fig. 3. The curvilinear grids are of dimension $32 \times 14 \times 14$ on three axes(object axis). The first set samples the pressure distribution of the fluid and the second samples the mach number distribution. The value of each sample was normalized in the range $[0, 1]$ in advance. To understand the distribution of these samples, several isosurfaces were rendered in a picture(several thresholds were investigated at a time) using our algorithm. Fig. 4 shows the pressure distribution in the blunt fin. The pressure originally ranged from 999.0 to 7700.0(normalized in advance). The contour of a cross section is shown in Fig. 4a(set rotation angle =0). The 3-D dimensional picture viewed from another angle is shown in Fig. 4b. Similarly, Fig. 5 shows the distribution of the Mach number. The original maximum and minimum values of the Mach number were 0.46 and 0.0, respectively. We selected several thresholds and rendered them in a image. The other image is shown in Fig. 5b.

The results of the experiments show that each image was generated in a reasonable amount of time. The computation time for each image was about 15 to 20 seconds, and it could be reduced to about 8 to 10 seconds when our speedup techniques were applied.

Table 1 shows the statistics of the running time for ray-casting and our algorithm for a set of CT data (regular) and a set of CFD data. Since regular data can be thought as a special case of curvilinear data. We implemented this algorithm on regular CT data, and the difference of the running time between ray-casting and our method was significant. As we mentioned above, locating the sample point is a time-consuming task for a ray-casting algorithm. The difference of running time between this two methods will be more significant for curvilinear data. Our implementation was developed on an IRIS workstation which had 16 megabytes of memory and a 25 MHZ CPU.

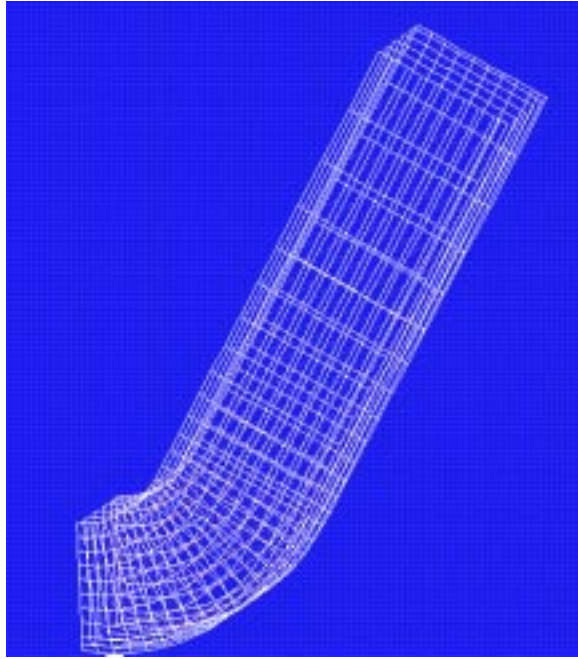


Fig. 3. Data distribution of a blunt fin.

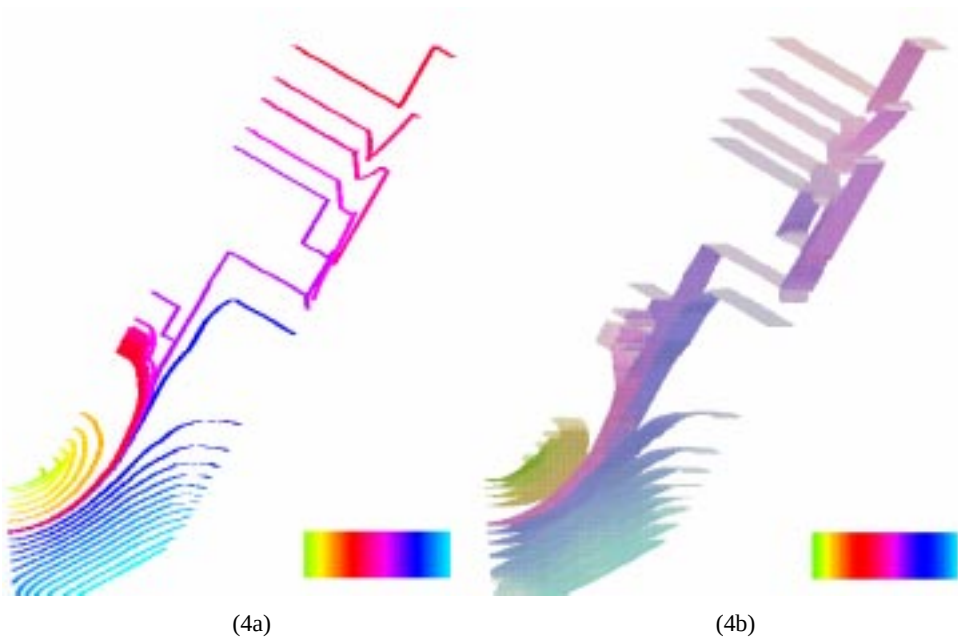


Fig. 4. The pressure distribution(4a) and the pressure distribution rotation about the y axis(4b).

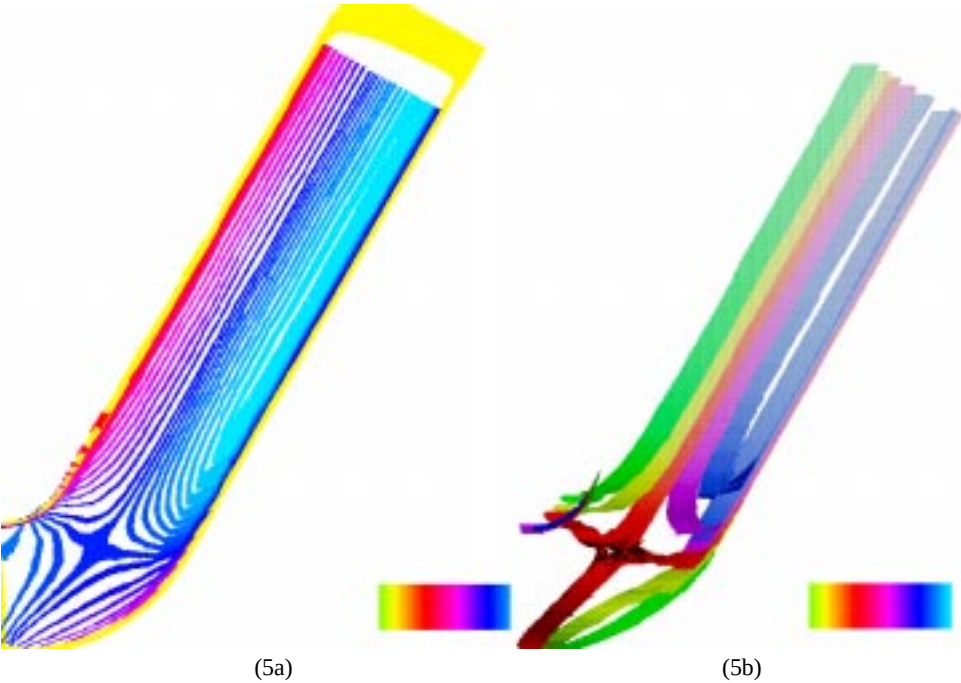


Fig. 5. The Mach number distribution(5a) and the Mach number distribution(rotation about the y axis) (5b).

Table 1. Statistics of running time for CT and CFD data.

	Rotation Angle	Scaling Factor	Ray Casting	Isosurface Probing
CT data 256×256×74	(0,90,0) (-20,120,0)	1×1×2	424 sec.	34 sec.
		2×2×4	4552 sec.	96 sec.
		1×1×2	461 sec.	65 sec.
		2×2×4	5487 sec.	130 sec.
CFD data 32×14×14	(0,0,0)	1×1×1		15 sec.
	(30,0,0)			18 sec.

5. CONCLUSIONS

In this paper, we have presented an approach to rendering curvilinear volume data. We construct isosurfaces in the curvilinear volume using analytic trilinear functions. In this method, processing of each cell is independent of processing of the others, thus, our method is easily parallelized. Speed-up techniques are included in our method. We have also proposed a *union-find* based algorithm for maintaining the frame buffer. The reduction in computing time is significant, especially when most of the points in the frame buffer are occupied. Almost always, only two comparisons are needed to determine whether a scan-line is invisible or not.

Furtherly, an error estimation and optimization method have been provided based on our method. Using our approach, each image can be generated in a reasonable amount of time after the volume data is preprocessed.

REFERENCES

1. A. Aho, J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company.
2. T. M. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, pp. 440-446, MIT press.
3. E. Horowitz, S. Sahni and S. A. Frad, *Fundamental of Data Structures in C*, pp. 238-247, Computer Science press.
4. A. Watt and M. Watt, *Advanced Animation and Rendering Techniques*, Addison-Wesley Publishing Company, 1992.
5. D. Akio and K. Akio, "A cell-traverse display algorithm for regular and rectilinear 3D grid data," *The Jornal of Visualization and Computer Animation*, Vol. 3, No. 1, 1992, pp. 129-143.
6. B. Tabatabai and E. A. Sessarego, "Volume rendering on non-regular grids," *EUROGRAPHICS*, Vol. 13, No. 3, 1994, pp. C247-C258.
7. C. T. Silva and S.B. Mitchell, "The lazy sweep ray casting algorithm for rendering irregular grids," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 3, No. 2, 1997.
8. P. Hanrahan, "Three pass affine transformation," *Computer Graphics*, Vol. 24, No. 5, 1990, pp. 71-78.
9. H. C. Homer and S. H. Thomas, "A survey of construction and manipulation of octrees," *Computer Vision , Graphics, and Image Processing*, Vol. 43, No. 3, 1988, pp. 409-429.
10. K. Wu and L. Hesselink, "Computer display of reconstructed 3D scalar data" *Applied Optics*, Vol. 27, No. 2, 1988, pp. 395-404.
11. M. Levoy, "Efficient ray tracing of volume rendering," *ACM Transactions on Graphics*, Vol. 9, No. 3, 1990, pp. 245-261.
12. M. Levoy, "Volume rendering by adaptive refinement," *The Visual Computer*, Vol. 6, No. 1, 1990, pp. 2-7.
13. C. C. Lin and Y. T. Ching, "An efficient volume rendering algorithm using an analytic approach," *Visual Computer*, Vol. 12, No. 10, 1996, pp. 515-526.
14. W. E. Lorensen, "Marching cubes: a high resolution 3D surface construction algorithm," *Computer Graphics*, Vol. 21, No. 4, 1987, pp. 163-169.
15. M. P. Garrity, "Raytracing irregular volume data," *Computer Graphics*, Vol. 24, No. 5, 1990, pp. 35-40.
16. B. K. Natarajan, "On generating topologically consist isosurfaces from uniform samples," *The Visual Computer*, Vol. 11, No.1, 1994. pp. 52-62.
17. G. M. Nielson, T. A. Foley, B. Hamann and D. Lane, "Visualizing and modeling scattered multivariate data," *IEEE Computer Graphics and Applications*, 1991, Vol. 11, No. 3, pp. 47-55.
18. K. L. Novins, F. X. Sillion and D. P. Greenberg, "An efficient method for volume rendering using perspective projection," *Computer Graphics*, Vol. 24, No. 5, 1990, pp. 95-101.

19. P. L. Williams, "Visibility ordering meshed polyhedra," *ACM Transactions on Graphics*, Vol. 11, No. 2, 1992, pp. 103-126.
20. P. L. Williams, "A high accuracy volume renderer for unstructured data," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, No. 1, 1998, pp. 37-54.
21. R. S. Gallagher and J. C. Nagtegaal, "An efficient 3D visualization technique for finite element models and other coarse volumes," *Computer Graphics*, Vol. 23, No. 3, 1989, pp. 185-193.
22. R. Yagel, D. S. Ebert, J. N. Scott and Y. Kurzion, "Grouping volume renderers for enhanced visualization in computational fluid dynamics," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 2, 1995, pp. 117-132.
23. T. Ulf, H. H. Karl, B. Michael, P. Andreas, R. Martin and W. Gunnar, "Investigation of medical 3D-rendering algorithms," *IEEE Computer Graphics and Applications*, Vol. 10, No. 2, 1990, pp. 41-53.
24. J. Wilhelms and A. V. Gelder, "Octrees for faster isosurfaces generation extended abstract," *Computer Graphics*, Vol. 24, No. 5, 1990, pp. 57-62.
25. J. Wilhelms and A. V. Gelder, "Topological considerations in isosurface generation extended abstract," *Computer Graphics*, Vol. 24, No. 5, 1990, pp. 79-82.
26. J. Wilhelms, J. Challinger, N. Alper and S. Ramamoorthy, "Direct volume rendering of curvilinear volumes," *Computer Graphics*, Vol. 24, No. 5, 1990, pp. 41-47.
27. J. Wilhelms and A. V. Gelder, "A coherent projection approach for direct volume rendering," *Computer Graphics*, Vol. 25, No. 4, 1991, pp. 275-283.
28. J. Wilhelms, "Pursuing interactive visualization of irregular grids," *The Visual Computer*, Vol. 7, No. 9, 1993, pp. 450-457.
29. X. Mao, "Splatting of non rectilinear volumes through stochastic resampling," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, No. 2, 1996, pp. 156-170.

Chyi-Cheng Lin (林祺政) was born on November 28, 1966, in Taipei, Taiwan, R.O.C. He received his BS degree from the Tatung Institute of Technology in 1989, and his MS and PhD degrees in 1992 and 1997, respectively, in Computer Science from National Chiao Tung University. He is currently with the Institute for Information Industry, Taipei, Taiwan. His current research interests include volume rendering, virtual reality and network multimedia.

Yu-Tai Ching (荆宇泰) was born on 20 February 1957 in Taipei, Taiwan, R.O.C. He received his BS degree in Industrial Engineering in 1980 from National Tsing Hua University, and his MS and PhD degrees in 1983 and 1987, respectively, in Computer Science from Northwestern University. He is currently with the Department of Computer and Information Science, National Chiao Tung University. His research interests are medical imaging, computer graphics, and design and analysis of computer algorithms.