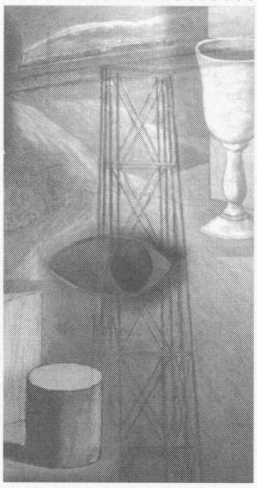
THE JOURNAL OF VISUALIZATION AND COMPUTER ANIMATION
J. Visual. Comput. Animat. **10**, 79–90 (1999)

High-quality Rendering of Smooth Isosurfaces



*By Eric LaMar, Bernd Hamann and Kenneth I. Joy**

Animation and visualization of rectilinear data require interpolation schemes for smooth image generation. Piecewise trilinear interpolation, the de facto standard for interpolating rectilinear data, usually leads to significant visual artifacts in the resulting imagery. These artifacts reduce the confidence in the resulting visualization and may even lead to false interpretations of the data. This paper is concerned with the generation of smooth isosurface image sequences, obtained by casting rays through the image plane and computing their intersections with an isosurface. We describe a novel solution to this problem: we replace trilinear interpolation by tricubic interpolation, smoothing out the artifacts in the images; and we simplify the ray–isosurface intersection calculations by rotating and resampling the original rectilinear data in a second rectilinear grid—a grid with one family of grid planes parallel to the image plane. Our solution significantly reduces artifacts in individual images and leads to smooth animations. Copyright © 1999 John Wiley & Sons, Ltd.

Received 25 May 1998; Revised 10 December 1998

KEY WORDS: animation; approximation; trilinear splines; volume visualization; ray casting; isosurfaces; resampling; Catmull–Rom splines; Hardy's multiquadric scheme

1. Introduction

With ever-increasing speed, data sets are being generated that represent three-dimensional, volumetric information. They arise as a result of complex computational simulations or empirical data collection procedures and present a challenge to current visualization techniques. As these are large data sets, the approximation techniques used to render the data produce artifacts that are visible in the resulting images, especially in the animations. These artifacts frequently present misleading information about the data and may reduce the reliability of the visualization. It is crucial to clearly display all information contained in a data set and not to simplify the data for the sake of rendering ease.

Scientific data sets are typically scalar- or vector-valued and defined on a grid in three-dimensional space. In many cases the grid is rectilinear and the data represent either point values associated with the nodes (or vertices) or constant values associated with the voxels (or grid cells). Three methods have been developed to visualize these data sets: the construction of isosurfaces, i.e. surfaces defining the region in space for which a particular field variable is constant;[1–3] volume visualization methods, where the algorithm casts rays through the volume, averaging colour and opacity values along the ray;[1] and splatting, which is based on projecting the voxel data onto the image plane and considering relative depth to simulate transparency.[4–6]

We describe a method based on a C^1-continuous interpolation scheme that leads to normal continuous isosurfaces. We construct a cubic tensor product spline interpolating the function values at the grid points, ensuring an overall C^1-continuous representation of the

*Correspondence to: K. I. Joy, Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California, Davis, CA 95616 8562, U.S.A. E-mail: joy@cs.uc.davis.

Contract/grant sponsor: National Science Foundation; Contract/grant number: ACI 9624034.

Contract/grant sponsor: Office of Naval Research; Contract/grant number: N00014-97-1-0222.

Contract/grant sponsor: Army Research Office; Contract/grant number: ARO 36598-MA-RIP.

Contract/grant sponsor: NASA Ames Research Center; Contract/grant number: NAG2-1216.

Contract/grant sponsor: Lawrence Livermore National Laboratory; Contract/grant numbers: W-7405-ENG-48, B335358, B347878.

Contract/grant sponsor: North Atlantic Treaty Organization (NATO); Contract/grant number: CRG.971628.

Contract/grant sponsor: Silicon Graphics, Inc.

CCC 1049–8907/99/020079–12$17.50

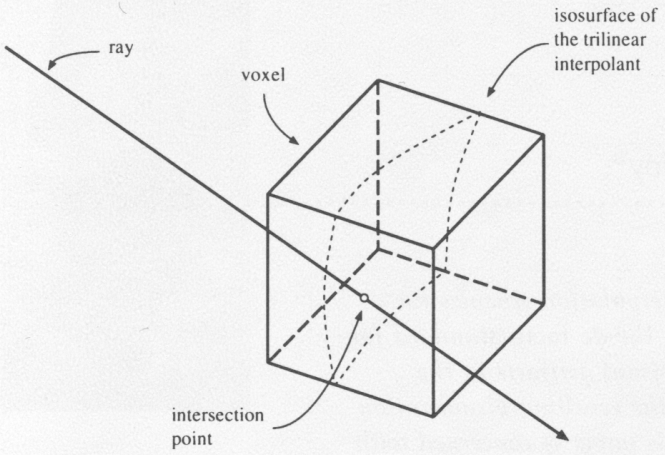


Figure 1. Ray–isosurface intersection in a voxel. The shaded area shows the isosurface of the trilinear approximant. The bounding curves of the isosurface on the faces of the voxel are hyperbolic arcs

field. The algorithm extends the *de facto* standard of piecewise trilinear interpolation of the data in a cell to piecewise tricubic interpolation.

Piecewise trilinear interpolation (Figure 1) of rectilinear data interpolates the eight data values at the corners of a cell and yields an overall C^0-continuous approximation of a field. The resulting isosurfaces have normal discontinuities on the shared boundary faces of each pair of neighbouring voxels. The resulting normal discontinuities are clearly visible when rendering isosurfaces. Images obtained by ray casting also suffer from the lack of smoothness in the trilinear spline interpolation. The resulting artifacts, 'ring patterns', are particularly disturbing in animations of a data set.

Piecewise tricubic interpolation utilizes the 64 vertices at the corners of a $3 \times 3 \times 3$ array of cells and calculates a tricubic approximant to the values within the cell. To determine a point on an isosurface, we insert a parametric ray equation

$$\mathbf{r}(t) = \mathbf{p}_0 + t\vec{\mathbf{d}} = \begin{pmatrix} x_0 + tx_d \\ y_0 + ty_d \\ z_0 + tz_d \end{pmatrix}$$

into the individual tricubic polynomial segments characterized the field over each voxel, i.e.

$$f(x,y,z) = \sum_{k=0}^{3}\sum_{j=0}^{3}\sum_{i=0}^{3} c_{i,j,k}x^i y^j z^k$$

Thus we have to solve the equation

$$\sum_{k=0}^{3}\sum_{j=0}^{3}\sum_{i=0}^{3} (x_0 + tx_d)^i (y_0 + ty_d)^j (z_0 + tz_d)^k = \bar{f}$$

to determine points on the isosurface $f=\bar{f}$. This is an equation of degree nine in t, so the intersection of a ray with the isosurface of a tricubic polynomial requires the solution of a ninth-degree polynomial—a difficult task.

However, if the direction vector œ**d** of the ray **r** is parallel to one of the axes of the co-ordinate system in which $f(x,y,z)$ is defined, then two of the values of x_d, y_d and z_d must be zero and the intersection problem leads to a (solvable) third-degree equation. Thus we reduce the general ray–isosurface intersection problem by rotating the given rectilinear grid such that one of its co-ordinate 'directions' is parallel to the ray direction, and by re-sampling the field at the vertices of the 'ray-aligned' grid. Thus each intersection calculation is reduced to root finding for a third-degree polynomial.

In Section 2 we review research related to this problem. In Section 3 we review the tricubic spline that we use to approximate a three-dimensional field. In Section 4 we discuss the rotation and resampling steps that allow us to efficiently render smooth isosurfaces by using axis-aligned rays. The ray–isosurface intersection method is discussed in Section 5. In Section 6 we discuss ray–isosurface intersection and implementation details of the algorithm. We present results of our approach in Section 7 and conclude with an outlook on future work in Section 8.

2. Related Work

Two basic approaches are used in scientific visualization for the identification, extraction and rendering of isosurfaces: those that generate an intermediate representation of the isosurfaces for rendering purposes; and those that render the data directly.

The marching cubes algorithm[2,3,7] generates an intermediate triangular mesh from a set of trilinear interpolants. Considering each trilinear interpolant independently, the algorithm determines a set of triangles that approximates the isosurface within each voxel. The resulting triangular mesh is displayed using conventional rendering hardware. The number of cases to be considered for generating the mesh within a voxel can be simplified by symmetry and rotation to 14 cases, and the algorithm uses a look-up table to quickly generate the triangular mesh approximating an isosurface. The marching cubes method has been adapted to tetrahedral meshes by Shekhar *et al.*,[8] where only three cases must be considered.

Hamann *et al.*[9] describe a method for constructing a 'smoother' isosurface by fitting rational quadratic surface patches to the triangular mesh resulting from the marching cubes algorithm. Since the contour of the bilinear interpolant on a face of a grid cell is a hyperbolic arc, rational quadratic curves can represent this curve exactly. These boundary curves, together with boundary curves in the interior of a grid cell, can be used to generate the control nets of triangular rational quadratic patches that approximate the isosurface within the cell. The use of this type of surface patch yields a 'smoother' isosurface, but tangent plane discontinuities still exist at the boundaries of each grid cell.

Levoy[1] introduced the concept of casting rays directly through volumetric data to obtain transparent images. His algorithm uses trilinear spline interpolation along with the spline gradient to assign normal vector points along the rays for shading purposes. In order to emphasize regions in a three-dimensional data set where the (scalar) function values are in a certain range, one can assign higher weights to data in these regions by increasing their opacity values. Furthermore, one can use the gradient to emphasize boundaries between different 'tissue types': high gradients imply such boundaries and can therefore be used to increase the opacity of boundary regions. Levoy describes an efficiency improvement of his algorithm in References 10 and 11.

Several volume visualization algorithms that utilize and render the data directly are based on transforming a rectilinear data set so that it coincides with an 'image space cube'.[12–14] These methods factor the viewing matrix into a three-dimensional shear and a two-dimensional 'warp', thus projecting the given data set into image space. The shear–warp algorithm is a resampling scheme which is advantageous for algorithms utilizing SIMD architectures.[15]

Yagel and Kaufman[16] present an algorithm for volume rendering that is based on exploiting coherency between rays in parallel projections. Rays are cast from a base plane, which simplifies the paths through the volume in a serial image-order algorithm, insuring uniform sampling of the volume.

Webber[17] describes an algorithm that casts rays through the grid directly. This algorithm constructs a biquadratic isosurface over each grid cell and intersects the ray with this surface. This biquadratic surface is derived by examining the $3 \times 3 \times 3$ volume of voxels around the target voxel and using one of the 3×3 sides of this voxel array as a base for a biquadratic function. This approximating function is used for ray–isosurface intersection tests. This method allows the derivation of a precise intersection and a precise normal over a voxel.

Our approach combines ray-casting methods with a tricubic spline interpolation scheme for high-quality image generation and animation. To eliminate the root finding for ninth-degree polynomials, we rotate and resample the original rectilinear data set at the vertices of a grid with one family of grid planes parallel to the image plane. We cast rays that are normal to the image plane and thus perform ray intersection calculations for third-degree polynomials. These methods are all independent calculations and can be efficiently implemented on a parallel processing system.

3. Tricubic Interpolation

To solve the general ray–isosurface intersection problem, we must compute the intersection of a (parametric) ray and an isosurface of a tricubic approximant. We utilize cubic tensor-product Catmull–Rom splines[18,19] to represent the scalar field for all grid cells. We briefly review this spline scheme for the univariate case.

Given a set of scalar values $f_0, f_1, \ldots, f_n$, the univariate cubic Catmull–Rom spline is a piecewise cubic polynomial function

$$f(x) = \sum_{i=0}^{n} f_i(x) F_i(x)$$

where each $f_i(x)$ is a function of the values f_i, and $F_i(x)$ denotes a set of blending functions.[18] The functions $f_i(x)$ depend on the f_i values but are allowed to vary with x. Catmull and Rom discovered that certain choices of the functions $f_i(x)$ allowed the curve to interpolate the control points f_i.

For our application it is sufficient to assume that the knots of the spline curve are the integers; then, by utilizing the basis functions $F_i(x) = (x - \lfloor x \rfloor)^i$, we can obtain an interpolating spline segment (see the discussion in Reference 18) by setting $f_j(x) = 0$ for $j < \lfloor x \rfloor - 1$ and $j > \lfloor x \rfloor + 2$, and

$$\begin{bmatrix} f_{k-1}(x) \\ f_k(x) \\ f_{k+1}(x) \\ f_{k+2}(x) \end{bmatrix} = M_C \begin{bmatrix} f_{k-1} \\ f_k \\ f_{k+1} \\ f_{k+2} \end{bmatrix}$$

where $k = \lfloor x \rfloor$* and

*$\lfloor x \rfloor$ denotes the floor of x, the greatest integer less than x.

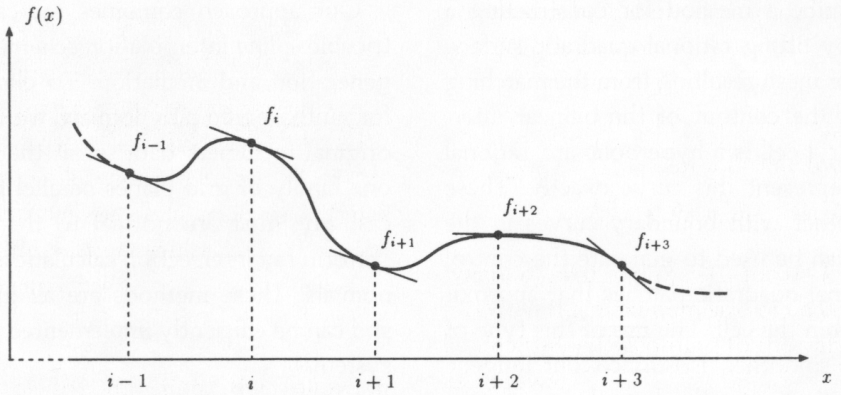


Figure 2. A Catmull–Rom spline. The curve interpolates the control points at integer values of the parameter. The slope of the curve at the integer knots is given by the central difference of the two values on either side of each knot

$$M_C = \frac{1}{2}\begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

Thus one can write $f(x)$ as

$$f(x) = [1 \quad x-k \quad (x-k)^2 \quad (x-k)^3] \times M_C \begin{bmatrix} f_{k-1} \\ f_k \\ f_{k+1} \\ f_{k+2} \end{bmatrix}$$

where $k = \lfloor x \rfloor$. For $x = k$ the spline interpolates the point f_k, and the derivative at $x = k$ is $\frac{1}{2}(f_{k+1} - f_{k-1})$. Figure 2 illustrates a Catmull–Rom spline scheme.

A segment of the Catmull–Rom spline can also be written as

$$f(x) = \sum_{i=0}^{3} f_{\lfloor x \rfloor + i - 1} C_i(x - \lfloor x \rfloor)$$

where

$$[C_0(x) \quad C_1(x) \quad C_2(x) \quad C_3(x)] = [1 \quad x \quad x^2 \quad x^3]\, M_C$$

and the functions $C_j(x)$ denote the Catmull–Rom basis functions.

We can bound the values of a Catmull–Rom spline for a particular knot interval by converting it to Bernstein–Bézier form. This conversion is given by

$$f(x) = [B_0(x) \quad B_1(x) \quad B_2(x) \quad B_3(x)] \times M_B^{-1} M_C \begin{bmatrix} f_{k-1} \\ f_k \\ f_{k+1} \\ f_{k+2} \end{bmatrix}$$

where

$$M_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & 6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

The matrix M_B converts from the cubic power basis to the Bernstein basis $B_j(x)$.[19] The convex hull property of Bézier curves ensures that each segment of the spline lies between the minimal and maximal values of its four defining Bernstein–Bézier control points. The Bernstein–Bézier 'control coefficients' are given by

$$\begin{bmatrix} b_{k-1} \\ b_k \\ b_{k+1} \\ b_{k+2} \end{bmatrix} = M_B^{-1} M_C \begin{bmatrix} f_{k-1} \\ f_k \\ f_{k+1} \\ f_{k+2} \end{bmatrix}$$

We define I as the smallest interval containing all four values b_{k-1}, b_k, b_{k+1} and b_{k+2}. The convex hull property ensures that the curve $f(x)$ is contained within the interval I.

The univariate Catmull–Rom spline can be extended to the trivariate case by constructing a tensor-product spline. In the trivariate case an individual tricubic segment is given by

$$f(x,y,z) = \sum_{k=0}^{3} \sum_{j=0}^{3} \sum_{i=0}^{3} f_{\lfloor x \rfloor + i - 1, \lfloor y \rfloor + j - 1, \lfloor z \rfloor + k - 1} C_i(x - \lfloor x \rfloor) C_j(y - \lfloor y \rfloor) C_k(z - \lfloor z \rfloor)$$

We use this spline scheme to represent a piecewise tricubic approximation of a scalar field. We must consider a stencil

of 64 data values to define the Catmull–Rom spline for a particular grid cell. These 64 data values include the scalar values at the corners of a grid cell C and the scalar values at the corners of the 26 neighbouring cells sharing at least one vertex with C. The 64 values are the set

$$\{f_{i,j,k}: i \in \{i_0-1, i_0+2\}, \\ j \in \{j_0-1, j_0+2\}, k \in \{k_0-1, k_0+2\}\}$$

where f_{i_0,j_0,k_0} is the value at the vertex of C with minimum i, j, k indices. Again it is possible to determine the range of function values for a particular tricubic polynomial by converting it to Bernstein–Bézier form and computing the minimal and maximal coefficients.

4. Rotation and Resampling

We use ray tracing as the mechanism for finding and rendering an isosurface of a trivariate scalar function. Suppose we are to render the isosurface defined by the expression

$$f(x,y,z)=\sum_{k=0}^{3}\sum_{j=0}^{3}\sum_{i=0}^{3}c_{i,j,k}x^iy^jz^k=\bar{f} \qquad (1)$$

for a particular voxel. We must intersect the ray

$$\mathbf{r}(t)=\mathbf{p}_0+t\vec{\mathbf{d}}=\begin{pmatrix}x_0+tx_d\\ y_0+ty_d\\ z_0+tz_d\end{pmatrix} \qquad (2)$$

with the isosurface $f=\bar{f}$. To intersect $\mathbf{r}(t)$ with the isosurface, we substitute the x, y and z values of equation (2) into equation (1) and obtain, in general, a polynomial of degree nine in t. We reduce this problem to solving a third-degree equation by using rays that are aligned with the z axis, e.g. $x_d=y_d=0$.

Our algorithm is an image space algorithm, i.e. we cast rays through each pixel in the image plane I and find intersections with an isosurface. A ray perpendicular to the image plane I is cast through the source grid S (see Figure 3). We replace the grid S by a second rectilinear grid T, a grid with one family of grid planes parallel to the image plane. Rotating grid S into grid T allows us to perform ray–isosurface computations with axis-aligned rays (see Figure 4).

The rotation of S into T is effectively performed by resampling the scalar field approximation at the vertices of T, i.e. we must approximate new function values for the

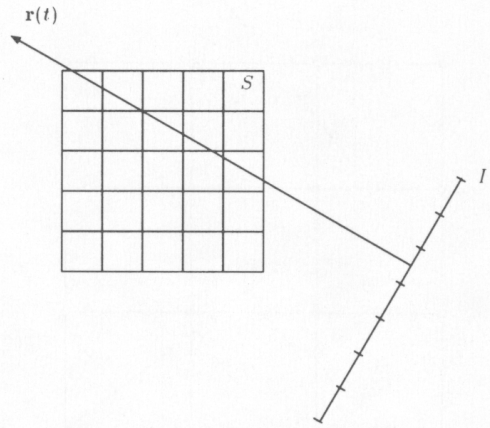


Figure 3. Casting rays through the grid S. *Ray* **r**(t) *passes through a pixel centre in the image plane* I *and intersects the arbitrarily oriented grid cells of* S

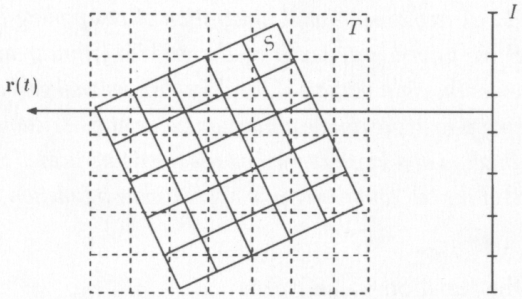


Figure 4. Definition of the grid T. *One family of grid planes of* T *is parallel to the image space plane* I. *Ray* **r**(t) *is axis-aligned with* T. *After resampling at the vertices of* T, *ray tracing can be performed with axis-aligned rays*

vertices of each cell in T from the original values at the vertices in S. These resampling problems arise in many visualization applications and can be resolved in a number of ways.[13,16,20–22] We compare three different approximation techniques for the estimation of function values at the vertices of T: (1) trilinear approximation; (2) tricubic spline approximation using the Catmull–Rom spline; and (3) Hardy's multiquadric method.[23,24]

Trilinear interpolation is the standard technique used to estimate function values for rectilinear data sets. In this method the function value for a vertex $\mathbf{p}$ of T is approximated by trilinearly interpolating the eight values at the vertices of the cell in S that contains $\mathbf{p}$ (see Figure 5). Denoting the co-ordinates of $\mathbf{p}$ in S by x_p, y_p and z_p, $\mathbf{p}$'s local parameter values with respect to the cell in S containing $\mathbf{p}$ are

$$u=x_p-\lfloor x_p\rfloor \\ v=y_p-\lfloor y_p\rfloor \\ w=z_p-\lfloor z_p\rfloor \qquad (3)$$

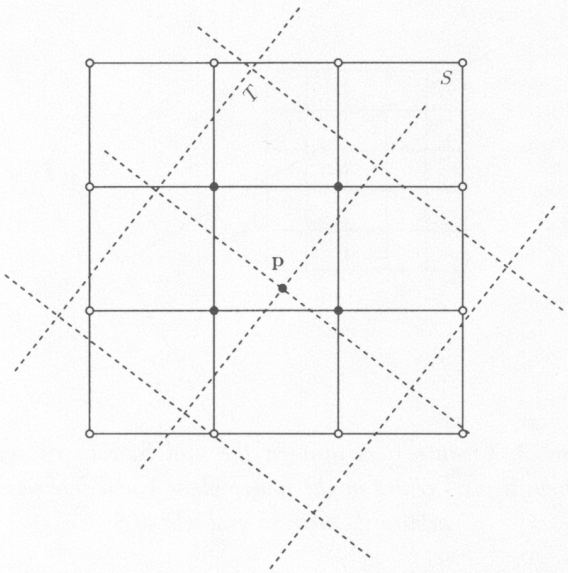


Figure 5. A two-dimensional illustration of resampling in three dimensions. Given a vertex $\mathbf{p}$ *of the grid* T, *trilinear approximation uses the eight vertices of the three-dimensional grid cell in* S *containing* $\mathbf{p}$ *to approximate a function value at* $\mathbf{p}$. *Tricubic spline approximation and Hardy's method use the 64 vertices of* S *in the stencil of the cell containing* $\mathbf{p}$ *to approximate a function value*

Thus the function value $f(\mathbf{p})$ is

$$f(\mathbf{p}) = \sum_{k=0}^{1}\sum_{j=0}^{1}\sum_{i=0}^{1} f_{\lfloor x_{p_1}\rfloor+i,\lfloor y_{p_1}\rfloor+j,\lfloor z_{p_1}\rfloor+k} \times L_i(u)L_j(v)L_k(w)$$

where the Lagrange polynomials are $L_0(t)=1-t$ and $L_1(t)=t$.

We can extend the trilinear spline approximation scheme to tricubic spline approximation by using the Catmull–Rom spline. In this case the function value for a vertex $\mathbf{p}$ of T is approximated by considering the 64 vertices of the 'stencil cell' in S associated with the cell containing $\mathbf{p}$. If u, v and w denote the local parameter values of $\mathbf{p}$ (equation (3)), then the function value $f(\mathbf{p})$ is

$$f(\mathbf{p}) = \sum_{k=0}^{3}\sum_{j=0}^{3}\sum_{i=0}^{3} f_{\lfloor x_{p_1}\rfloor+i,\lfloor y_{p_1}\rfloor+j,\lfloor z_{p_1}\rfloor+k} \times C_i(u)C_j(v)C_k(w)$$

where $C_i(t)$ denotes the cubic Catmull–Rom blending functions.

Hardy's multiquadric method[22–25] is a standard scattered data approximation scheme. It is used in cases where the data sites are randomly distributed and (typically) no connectivity information is known for the data sites. Hardy's approximant smoothly interpolates the values at the data sites and we use it for estimating function values for the vertices of the T grid.

Given a vertex $\mathbf{p}$ in the T grid and n vertices $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$ of S, with associated function values $f_0, f_1, \ldots, f_n$, Hardy's multiquadric interpolant is defined by the linear system

$$f_j = \sum_{i=0}^{n} \alpha_i \sqrt{(\|\mathbf{p}_j - \mathbf{p}_i\|)^2 + R^2}, \quad j=0,\ldots,n$$

and the estimate at $\mathbf{p}$ is

$$f(\mathbf{p}) = \sum_{i=0}^{n} \alpha_i \sqrt{(\|\mathbf{p} - \mathbf{p}_i\|)^2 + R^2}$$

where R^2 is some positive constant. The computation of the coefficients α_i requires the inversion of an $(n+1)\times(n+1)$ matrix.

If $\mathbf{p}$ is a vertex of the grid T, we consider the 64 data sites given by the vertices of the 'stencil' in S of the cell containing $\mathbf{p}$ (see Figure 5). Assuming that the spacing of the S grid is uniform, the distances $\|\mathbf{p}_j - \mathbf{p}_i\|$ are the same throughout the grid and one needs to invert the 64×64 matrix only once. Hardy's method is considerably slower than the tricubic spline approximation owing to the computation of the square roots. The visual results of the two approximation schemes are similar.

5. Ray–Isosurface Intersection Tests

To intersect the parametric ray $\mathbf{r}(t)=(x_0,y_0,z_0+tz_d)$ with the isosurface

$$f(x,y,z) = \sum_{k=0}^{3}\sum_{j=0}^{3}\sum_{i=0}^{3} c_{i,j,k}x^i y^j z^k = 0 \qquad (4)$$

we substitute the co-ordinate functions for x, y and z into equation (4), leading to the (normalized) cubic equation

$$t^3 + at^2 + bt + c = 0 \qquad (5)$$

To find the roots of this equation, we use Cardan's solution,[26] which first transforms equation (5) by substituting $s=t-a\,3$, leading to the polynomial

$$s^3 + ps + q \qquad (6)$$

where

$$p = -\frac{a^3}{3} + b$$

$$q = 2\left(\frac{a}{3}\right)^3 - \frac{ab}{3} + c$$

Equation (6) has three roots. The characteristics of these roots are determined by the discriminant of the cubic equation

$$D=4p^3+27q^2$$

which leads to three cases.

- If $D>0$, then the cubic equation has one real root and two complex roots. The single real root is given by

$$s_1=A+B$$

where

$$A=\sqrt[3]{\frac{q}{2}+\sqrt{\frac{D}{27}}}$$

$$B=\sqrt[3]{\frac{q}{2}+\sqrt{\frac{D}{27}}}$$

- If $D<0$, then the equation has three different real roots. The roots are given by

$$s_1=k\cos\left(\frac{\alpha}{3}\right)$$

$$s_2=k\cos\left(\frac{2\pi+\alpha}{3}\right)$$

$$s_3=k\cos\left(\frac{4\pi+\alpha}{3}\right)$$

where

$$k=\sqrt{-\frac{4p}{3}}$$

$$\cos\alpha=-\frac{4q}{k^3}$$

- If $D=0$, then the equation has three real roots, of which two are equal. The roots are given by

$$s_1=\sqrt[3]{\frac{q}{2}}$$

$$s_2=-2\sqrt[3]{\frac{q}{2}}$$

where s_1 is the double root.

To accelerate the ray–isosurface intersection computations, we store an interval with each grid cell in T, which represents a lower and an upper bound of the function values within the cell. The Catmull–Rom interpolants are polynomial splines and we find the lower and upper bounds by determining minimal and maximal spline coefficients of the corresponding Bernstein–Bézier representation (see Section 3). If, for a given grid cell, the desired isosurface value is outside the interval associated with this cell, then the cell cannot contain any part of the isosurface.

We also store intervals for each 'column of grid cells'.* Thus a ray can intersect an isosurface in a grid cell of a particular column only if the column-specific interval contains the particular isovalue. The calculation of lower and upper bounds turns out to be beneficial for 'sparse data sets', i.e. data sets containing large homogeneous areas or thin features.

6. Implementation

The algorithm requires an initial grid S and an isovalue f and it outputs an image of the isosurface $f=f$. To reduce sampling artifacts, we use a resolution for the grid T that is at least twice the resolution of the S grid (Nyquist limit) in each direction.[27,28] We estimate function values in T from those in S by using one of the sampling methods discussed in Section 4. We compute intervals for each grid cell in T and intervals for each column of grid cells in T.

For each pixel in the image plane we case a ray $\mathbf{r}(t)$ into the grid T. If the column-specific interval of the column of cells intersected by $\mathbf{r}(t)$ does not contain the specific isovalue, we skip to the next pixel. Otherwise, for each grid cell intersected by the ray, examine the intervals for each cell and determine whether the isosurface exists within the cell. If the interval bounds the isovalue, trace the ray through the grid cell and determine the possible intersections with the isosurface. If the ray intersects the isosurface, then we compute the intersection closest to the image plane and shade the surface using the gradient of the isosurface. Ray tracing terminates when all pixels have been processed.

We have implemented our algorithm on several Silicon Graphics workstations, including systems with multiple processors. Our implementation takes advantage of multiple processors by executing all compute-intensive activities in parallel. We have found a near-linear speed-up between the implementation on an SGI O2 (single-processor system) and an Origin 2000 (16-processor system).

*A column of grid cells in T consists of all the cells of T intersected by an axis-aligned ray.

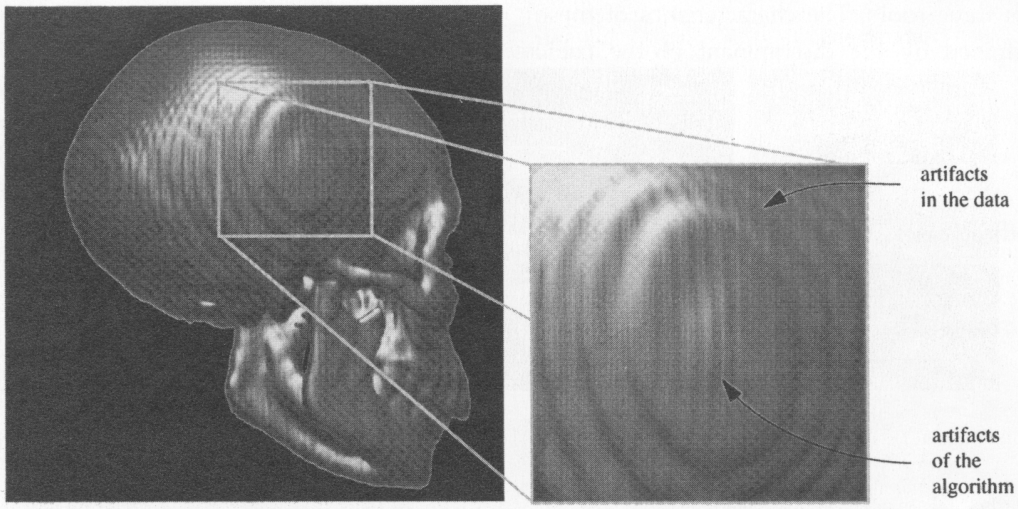


Figure 6. Artifacts generated from the algorithm and from the data in the skull data set. The artifacts inherent to the data are the ring patterns, while the artifacts with the algorithm are the vertical patterns in the image

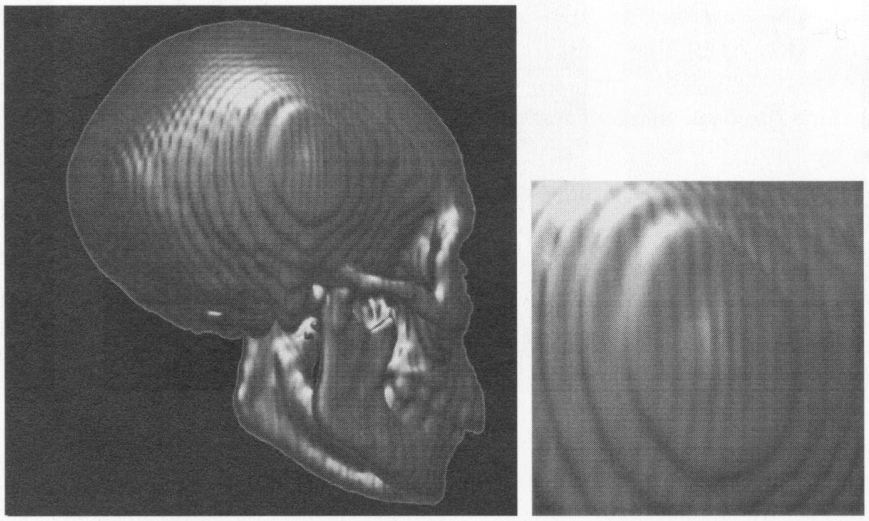


Figure 7. Skull data set rendered with trilinear sampling in the rotation step and tricubic interpolation in the ray-tracing step. Notice that most of the artifacts remain in the image

The mapping of source grid vertices to target grid vertices is done in parallel. The mapping of vertices is structured such that the size of the needed memory stride is kept at a minimum. Each processor runs to completion and waits until the other processors are done. We have experienced that the difference in completion time, considering all processors, is usually less than 10 per cent.

The ray–isosurface intersection calculations are parallelizable as well. Since rays may pass through different numbers of grid cells and since different types of computations may be performed, we use a more sophisticated load-balancing scheme: we use a job queue that consists of scan line indices. A processor reads a scan line index from the job queue and casts rays for all the pixels in the scan line. The time needed to acquire and update the job queue is very small compared with line scanning.

7. Results

Figure 6 shows the data set and artifacts that brought this research problem to our attention. This data set is the result of a computerized axial tomography (CAT) scan of a skull. The size of the data set is $64 \times 64 \times 68$ and its small resolution produces artifacts in the renderings. These artifacts are the 'circular' patterns shown in Figure 6. However, when we rendered the skull using conventional techniques,

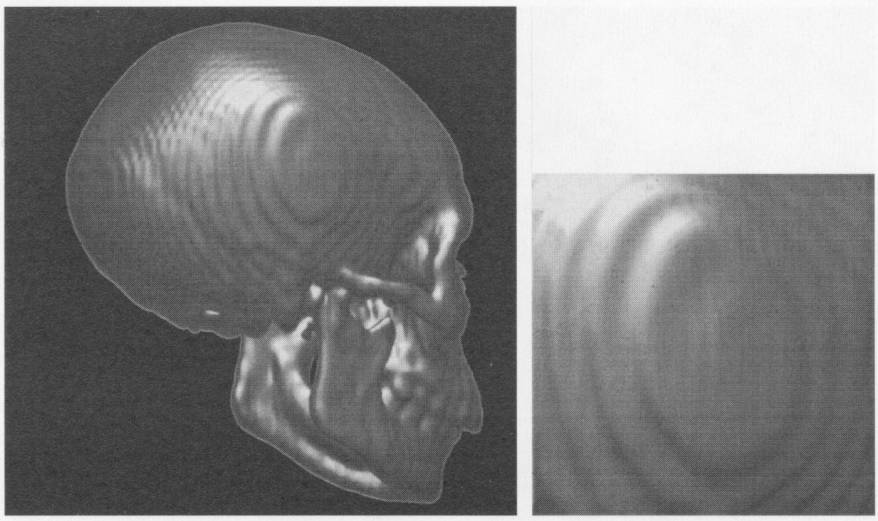


Figure 8. Skull data set rendered with Hardy sampling in the rotation step and tricubic interpolation in the ray-tracing step. Note that the artifacts are reduced but are slightly blurred

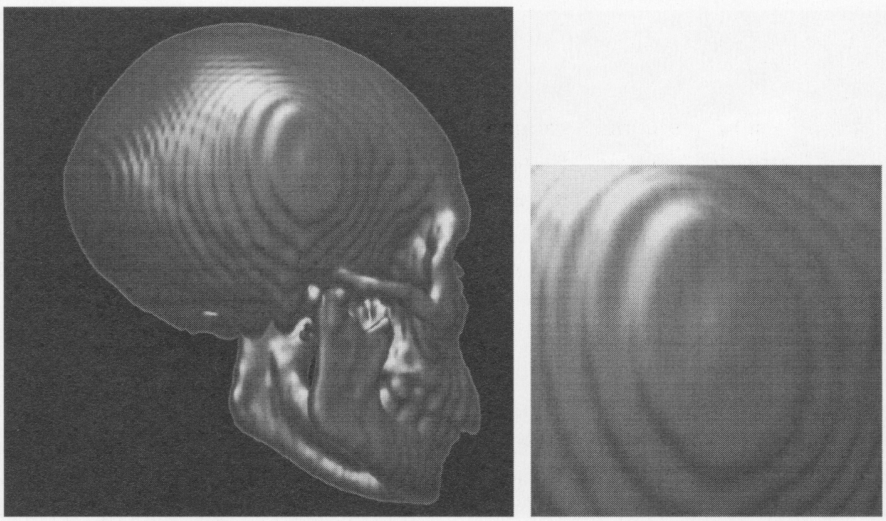


Figure 9. Skull data set rendered with tricubic sampling in the rotation step and linear interpolation in the ray-tracing step. One can see a substantial reduction in the artifacts generated by the algorithm

we obtained artifacts that were not inherent to the data. These are also visible in Figure 6 as the vertical 'stripe' pattern. When animated, the artifacts inherent to the data set stay fixed but the stripe artifacts ripple and move.

Figures 7–10 illustrate our methods as applied to this data set. Figure 7 illustrates the effect of trilinear sampling in the rotation step and tricubic interpolation in the ray-tracing step. We note that most of the artifacts due to the algorithm remain in the image. Figure 8 illustrates the effect of tricubic sampling in the rotation step and trilinear interpolation in the ray-tracing step. Here most of the artifacts are muted but still visible. Figure 9 illustrates the effect of Hardy sampling in the rotation step and tricubic interpolation in the ray-tracing step. In this image the artifacts due to the algorithm are muted but the data artifacts look blurred. Figure 10 illustrates the effect of tricubic sampling in the rotation step and tricubic interpolation in the ray-tracing step. The artifacts have nearly disappeared.

In animation these vertical bands appear to move and pulse across the isosurface. In an animation generated with tricubic sampling and tricubic interpolation in the ray-tracing step, the images appear smooth and very few artifacts are visible.

8. Conclusions

We have discussed a new method for rendering smooth isosurfaces of trivariate scalar fields using C^1-continuous

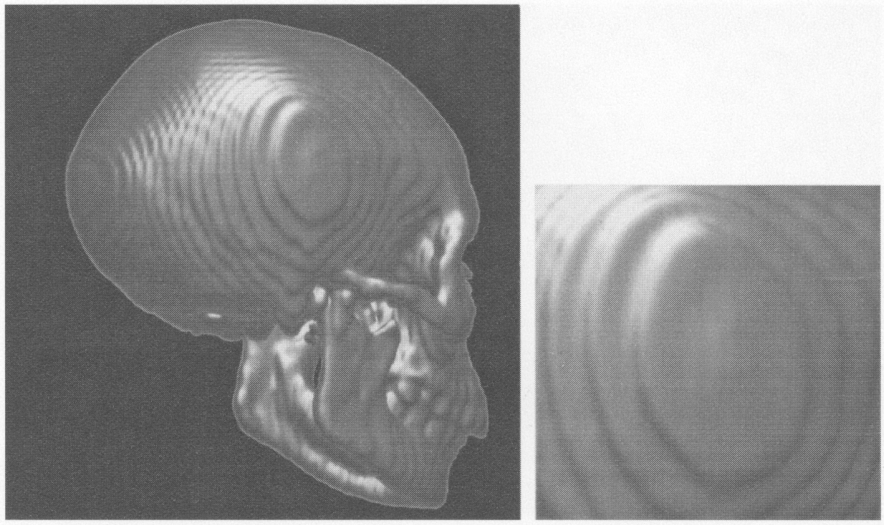


Figure 10. Skull data set rendered with tricubic sampling in the rotation step and tricubic interpolation in the ray-tracing step. The artifacts are nearly gone

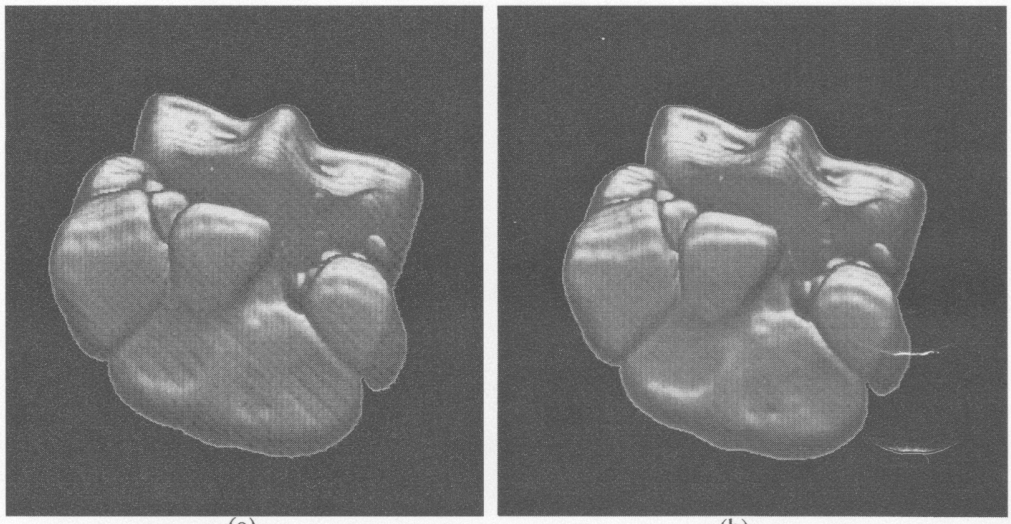
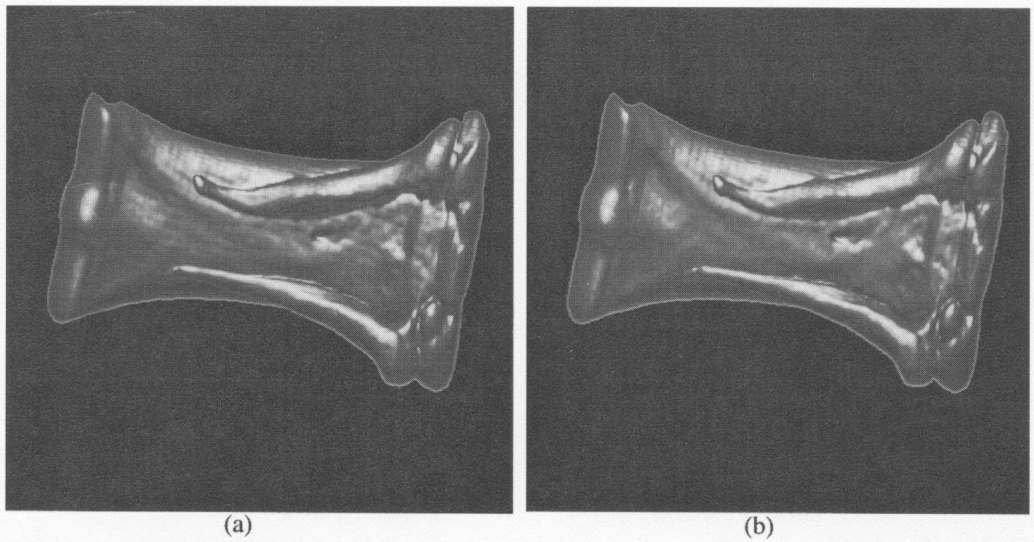


Figure 11. Equine metacarpal data set. (a) Using tricubic sampling and trilinear interpolation in the ray-tracing step. Note the bands running diagonally through the image. (b) Using tricubic sampling and tricubic interpolation in the ray-tracing step. The banding has effectively disappeared

spline functions with an efficient ray-tracing scheme. We have shown how a rotation–resampling approach can be used to reduce the complexity of ray–isosurface intersection calculations.

We plan to develop error metrics to characterize and quantify the error introduced during the rotation–resampling phase. We shall develop better methods to identify grid cells containing isosurfaces, more efficient ones than our current linear search scheme. We intend to apply our technique to other volume visualization techniques, e.g. Levoy's method, and possibly to the rendering of vector fields. We also will investigate the elimination of the rotation steps in this algorithm.[29]

If a data set has the same value over large regions, methods to smoothly approximate these regions by high-degree schemes do not improve smoothness: they require many unnecessary coefficients to be stored. Adaptive methods, such as an octree-based approach, could possibly be used to determine regions that hardly vary, and we could use low-degree schemes in such regions. Additional work must be done to improve the ray-tracing phase of the algorithm: there is a high degree of correlation among ray–isosurface intersections for adjacent pixels, which we do not consider at this point. We plan to investigate this in the future.



(a) (b)

Figure 12. A second view of the equine metacarpal data set. (a) Using tricubic sampling and trilinear interpolation. (b) Using tricubic sampling and tricubic interpolation. Again the banding virtually disappears

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award); the Office of Naval Research under contract N00014-97-1-0222; the Army Research Office under contract ARO 36598-MA-RIP; the NASA Ames Research Center through an NRA award under contract NAG2-1216; the Lawrence Livermore National Laboratory through an ASCI ASAP Level-2 under contract W-7405-ENG-48 and B335358, B347878; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of Silicon Graphics, Inc.

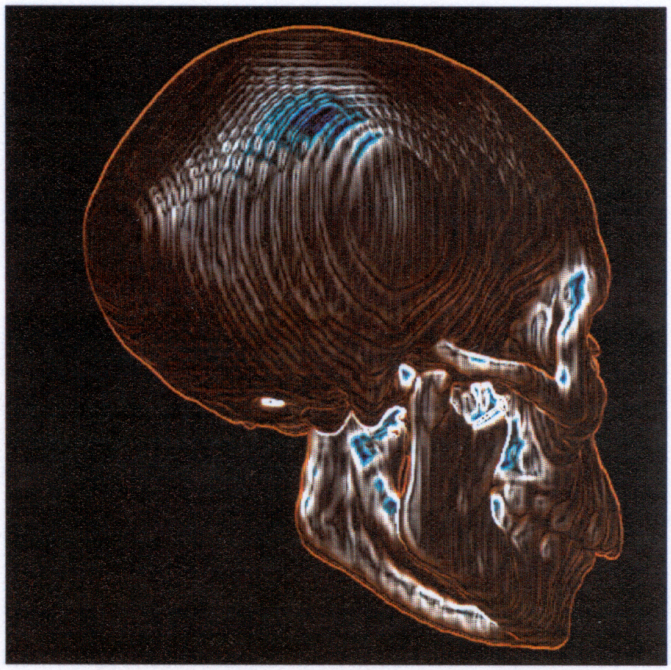
The equine metacarpal data set was provided by Dr C. M. Les of the JD Wheat Veterinary Orthopedic Research Lab at the University of California, Davis. We would like to thank all members of the Visualization Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis for their help.

References

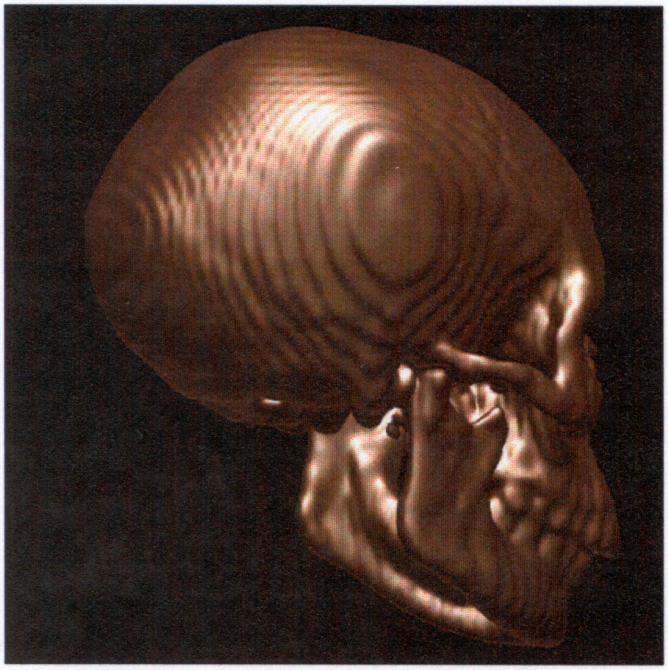
1. M. Levoy, 'Display of surfaces from volume data', *IEEE Computer Graphics and Applications*, **8**(3), 29–37 (1987).
2. W. E. Lorensen and H. E. Cline, 'Marching cubes: a high resolution 3D surface construction algorithm', *Computer Graphics (SIGGRAPH '87 Proceedings)*, **21**(4), 163–170 (1987).
3. G. M. Nielson and B. Hamann, 'The asymptotic decider: removing the ambiguity in marching cubes', *Proceedings of Visualization '91*, 1991, pp. 83–91.
4. H. E. Cline, W. E. Lorensen, S. Ludke, C. R. Crawford and B. C. Teeter, 'Two algorithms for the reconstruction of surfaces from tomograms', *Medical Physics*, **15**, 320–327 (1988).
5. D. Laur and P. Hanrahan, 'Hierarchical splatting: a progressive refinement algorithm for volume rendering', *Computer Graphics (SIGGRAPH '91 Proceedings)*, **25**(4), 285–288 (1991).
6. J. Wilhelms and A. Van Gelder, 'A coherent projection approach for direct volume rendering', *Computer Graphics*, **25**(4), 275–284 (1991).
7. B. Wyvill, C. McPheeters and G. Wyvill, 'Animating soft objects', *The Visual Computer*, **2**(4), 235–242 (1986).
8. R. Shekhar, E. Fayad, R. Yagel and F. Cornhill, 'Octree-based decimation of marching cubes surfaces', *Proceedings of Visualization '96*, 1996, pp. 335–342.
9. B. Hamann, I. J. Trotts and G. E. Farin, 'On approximating contours of the piecewise trilinear interpolant using triangular rational-quadratic Bézier patches', *IEEE Transactions on Visualization and Computer Graphics*, **VCG-3**, 215–227 (1997).
10. M. Levoy, 'Efficient ray tracing of volume data', *ACM Transactions on Graphics*, **9**(3), 245–261 (1990).
11. M. Levoy, 'Volume rendering by adaptive refinement', *The Visual Computer*, **6**(1), 2–7 (1990).
12. P. Lacroute, 'Real-time volume rendering on shared memory multiprocessors using the shear–warp factorization', in S. M. Spencer (ed.), *Proceedings of the 1995 Parallel Rendering Symposium*, ACM Press, 1995, pp. 15–22.
13. P. Lacroute, 'Analysis of a parallel volume rendering system based on the shear–warp factorization', *IEEE Transactions on Visualization and Computer Graphics*, **VCG-2**, 218–231 (1996).
14. P. Schröder and G. Stoll, 'Data parallel volume rendering as line drawing, 25–32.
15. T. Toffoli and J. Quick, 'Three-dimensional rotations by three shears', *Graphical Models and Image Processing*, **59**, 89–96 (1997).
16. R. Yagel and A. Kaufman, 'Template-based volume viewing', *Computer Graphics Forum*, **11**(3), 153–167 (1992).
17. R. E. Webber, 'Ray tracing voxel data via biquadratic local surface interpolation', *The Visual Computer*, **6**(1), 8–15 (1990).

18. E. Catmull and R. Rom, 'A class of local interpolating splines', in R. Barnhill and R. Riesenfeld (eds), *Computer Aided Geometric Design*, Academic Press, New York, 1974, pp. 317–326.
19. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design* 4th edn, Academic Press, Boston, MA, 1997.
20. R. Franke and G. Nielson, 'Smooth interpolation of large sets of scattered data', *International Journal for Numerical Methods in Engineering*, **15**, 1691–1704 (1980).
21. P. Lacroute and M. Levoy, 'Fast volume rendering using a shear–warp factorization of the viewing transformation', in A. Glassner (ed.), *Proceedings of SIGGRAPH '94*, ACM Press, 1994, pp. 451–458.
22. G. M. Nielson, T. A. Foley, B. Hamann and D. A. Lane, 'Visualizing and modeling scattered multivariate data', *IEEE Computer Graphics and Applications*, **11**(3), 47–55 (1991).
23. R. L. Hardy, 'Multiquadric equations of topography and other irregular surfaces', *Journal of Geophysical Research*, **76**, 1906–1915 (1971).
24. R. L. Hardy, 'Theory and applications of the multiquadric-biharmonic method: 20 years of discovery 1968–1988', *Computers and Mathematics with Applications*, **19**, 163–208 (1990).
25. R. Franke and G. M. Nielson, 'Scattered data interpolation and applications: a tutorial and survey', in H. Hagen, H. Mueller and G. Nielson (eds), *Focus on Scientific Visualization*, Springer, New York, 1995, pp. 131–159.
26. D Littlewood, *A University Algebra*, Heinemann, London, 1950.
27. A. S. Glassner, *Uniform Sampling and Reconstruction*, Morgan Kaufmann, San Francisco, CA, 1995, pp. 331–368.
28. A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
29. J. G. Cleary and G. Wyvill, 'Analysis of an algorithm for fast ray tracing using uniform space subdivision', *The Visual Computer*, **4**(2), 65–83 (1988).
30. C. L. Bajaj, V. Pascucci and D. R. Schikore, 'Fast isocontouring for improved interactivity', *IEEE Volume Visualization Symposium*, 1996, pp. 39–46.
31. T. D. DeRose and B. A. Barsky, 'Geometric continuity and shape parameters for Catmull–Rom splines', *Graphics Interface '84 Proceedings*, 1984, pp. 57–64.

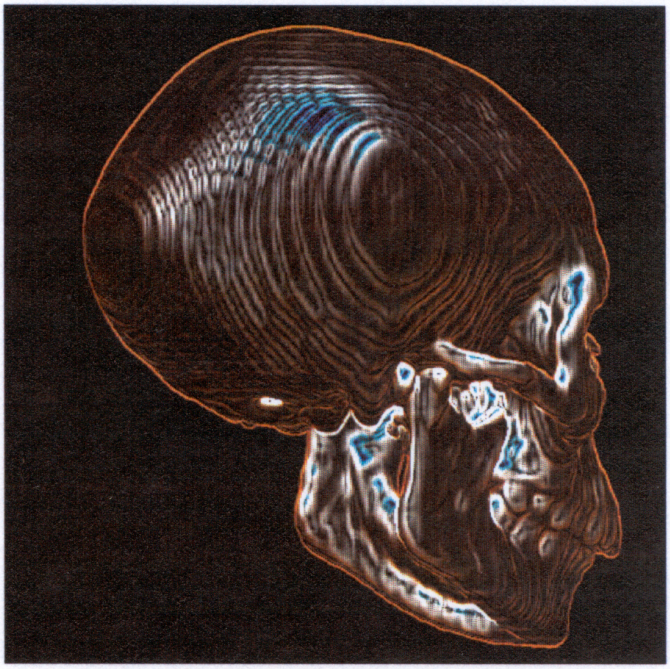
(a)



(b)



(c)



(d)

FIGURE 6: A comparison of trilinear and tricubic sampling using the skull data set: (a) Trilinear sampling; (b) tricubic sampling; (c) the image in (a) filtered with a Sobel operator; and (d) the image in (b) filtered with a Sobel operator. The two images were rendered using trilinear interpolation in the ray-tracing step.