Volume 21 (2002), number 1 pp. 19–31

Exact Isosurfaces for Marching Cubes

Holger Theisel

University of Rostock, Computer Science Department, PO Box 999, 18051 Rostock, Germany
theisel@informatik.uni-rostock.de

Abstract
In this paper we study the exact contours of a piecewise trilinear scalar field. We show how to represent these contours exactly as trimmed surfaces of triangular rational cubic Bézier patches. As part of this, we introduce an extension of the marching cubes algorithm which gives a topologically exact triangular approximation of the contours for any case. Finally, we modify the exact contours to be globally G^1 continuous without changing their topologies. We test the algorithm on both theoretical and practical data sets.

Keywords: iso-surface extraction, Marching Cubes, cubic Bezier patches, topological correctness

ACM CSS: I.4.8 Scene Analysis—*Surface fitting*; I.3.5 Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*

1. Introduction

The marching cubes (MC) algorithm [1,2] is one of the most popular methods of producing contours (isosurfaces) of a given volume data set. Given a scalar field on a regular hexahedral grid, MC yields a triangular approximation of the contour for an assumed trilinear interpolation of the scalar field between the grid points. This is obtained by considering the grid cells independently of each other and computing a triangular approximation of the isosurface for each of the cells.

The resulting triangular mesh might be too coarse or too fine. If the mesh is too fine (i.e. the number of triangles is too high), a variety of mesh reduction algorithms exist [3–6].

It is the purpose of this paper to deal with the opposite problem: to obtain a finer representation of the contour if the triangular mesh from MC is too coarse. This problem appears

- with low resolution volume data;
- when exploring details in high resolution volume data;
- with fractal volume data.

Figure 1(a) shows an example of a volume data set (property of Siemens Medical Systems Inc., Iselin, NJ) where MC gives a triangular mesh which is too fine. Figure 1(b) shows a detail of the inner part of the surface shown in Figure 1(a). Here the mesh is too coarse.

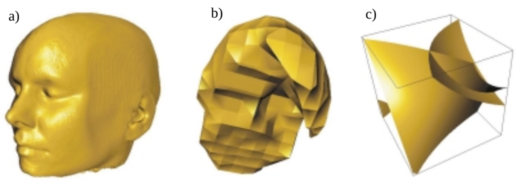

Figure 1: *(a) CT head consisting of* 423.963 *triangles—a candidate for mesh reduction algorithms. (b) Detail inside the same CT head—the triangular mesh is too coarse. (c) The contour of a triquadratically interpolated scalar field may have self-intersections and complicated topologies.*

In order to improve the quality of a surface shown in Figure 1(b), higher order interpolations between the grid points of the scalar field may be applied. This approach may end in new classes of surfaces containing self-intersections and complicated topologies. Figure 1(c) gives an illustration.

Another possible solution is to consider the exact nature of the contours of a trilinearly interpolated scalar field and find better surface representations than a triangular approximation produced by the MC algorithm. In [7], the contour is approximated by a number of bicubic patches.

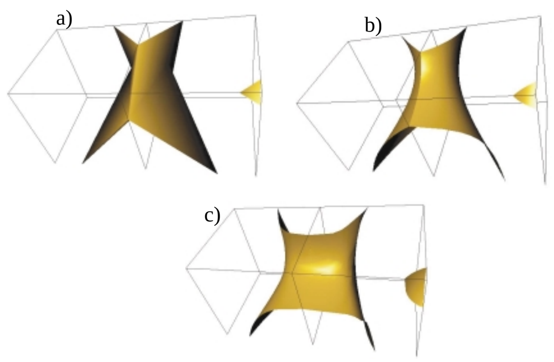

Figure 2: *Illustration of the main ideas of this paper. (a) Two cells and the triangular approximation of a certain contour consisting of two unconnected parts using MC. (b) The exact contour represented by a number of trimmed surfaces of rational cubic triangular patches. (c) The globally G^1 modification of the contour without changing its topology.*

The approximated surface is G^0 continuous across the cell boundaries. Piecewise bicubic patches are also used in [8] to refine the results of a "contouring and connecting" approach. In [9], the contour is approximated using patches with 4, 5 or 6 boundary curves. Hamann *et al.* [10] approximates the contours by rational quadratic triangular Bézier surface patches. This approach represents the boundary curves of the contour on the faces of the cells exactly but yields only G^0 continuous junctions of the patches both inside a cell and across the cell boundaries. All these approaches yield only approximations of the exact isosurface of a piecewise trilinear scalar field.

In this paper we describe two main ideas. First we find the exact representation of the trilinear contour as a piecewise parametric surface. It turns out that the contour can be described by a number of trimmed surfaces of rational cubic triangular patches. We show how to construct these surface patches.

The exact contour of a piecewise trilinear scalar field (described by rational cubic patches) is G^∞ continuous inside a cell but only G^0 continuous across the cell faces. In the second part of this paper we describe how to modify the exact contour of a piecewise trilinear scalar field in such a way that the result is a globally G^1 approximation of the contour. This modification does not change the topology of the original contour. This is obtained by applying an appropriate reparametrization of the domain of the scalar field, i.e. a cell. Figure 2 illustrates the main ideas of this paper.

This paper is organized in the following way: in Section 2 we study the exact contours of a trilinear scalar field. We show how to describe them as trimmed surfaces of rational cubic triangular Bézier patches. Section 3 introduces an extension of the MC method which gives a topologically exact triangular approximation of any trilinear contour. This is a necessary precondition for the approach of Section 2. Section 4 introduces a domain reparametrization of the piecewise trilinear scalar field in such a way that the resulting contours are globally G^1 continuous and have the same topology as the trilinear contours. Section 5 applies the methods to both constructed and real data sets.

2. The Exact Contour of a Trilinear Scalar Field

A first approach on constructing the exact contour of a trilinear scalar field can be found in [11] where it is constructed as a subdivision surface. In this section we introduce a parametric description of it.

Given is a trilinear scalar field

$$\begin{aligned} s(x,y,z) = &(1-x)(1-y)(1-z)\,c_{000} \\ &+(1-x)(1-y)\,z\,c_{001} \\ &+(1-x)\,y\,(1-z)\,c_{010} + (1-x)\,y\,z\,c_{011} \\ &+x\,(1-y)(1-z)\,c_{100} + x\,(1-y)\,z\,c_{101} \\ &+x\,y\,(1-z)\,c_{110} + x\,y\,z\,c_{111} \qquad (1) \end{aligned}$$

where c_{ijk} $(i,j,k \in \{0,1\})$ are the scalar values in the vertices of the unit cube. A contour is given by specifying a threshold r; it consists of all points $(x,y,z)^{\mathrm{T}}$ with

$$s(x,y,z) = r. \qquad (2)$$

Given is a point $\mathbf{a} = (x_{\mathbf{a}}, y_{\mathbf{a}}, z_{\mathbf{a}})^{\mathrm{T}}$. If we want to compute the intersection of a ray starting from $\mathbf{a}$ with the contour defined by (1) and (2), in general we have to solve a cubic equation. For the special case that the ray is parallel to one of the coordinate axes, the problem simplifies to the solution of a linear equation: let $\mathbf{p}_x(\mathbf{a})$ be the intersection of the ray $\mathbf{a} + \lambda(1,0,0)^{\mathrm{T}}$ with the contour defined by (1) and (2). Furthermore, let $\mathbf{p}_y(\mathbf{a})$ be the intersection of the ray $\mathbf{a} + \lambda(0,1,0)^{\mathrm{T}}$ with the contour, and let $\mathbf{p}_z(\mathbf{a})$ be the intersection of the ray $\mathbf{a}+\lambda(0,0,1)^{\mathrm{T}}$ with the contour. Then we obtain from (1) and (2):

$$\mathbf{p}_x(\mathbf{a}) = \left(\frac{\begin{array}{c} r - c_{000}\,(1-y_{\mathbf{a}})(1-z_{\mathbf{a}}) \\ -\,c_{001}\,(1-y_{\mathbf{a}})\,z_{\mathbf{a}} \\ -\,c_{010}\,y_{\mathbf{a}}\,(1-z_{\mathbf{a}}) - c_{011}\,y_{\mathbf{a}}\,z_{\mathbf{a}} \end{array}}{\begin{array}{c} (c_{100}-c_{000})(1-y_{\mathbf{a}})(1-z_{\mathbf{a}}) \\ +\,(c_{101}-c_{001})(1-y_{\mathbf{a}})\,z_{\mathbf{a}} \\ +\,(c_{110}-c_{010})\,y_{\mathbf{a}}\,(1-z_{\mathbf{a}}) \\ +\,(c_{111}-c_{0}11)\,y_{\mathbf{a}}\,z_{\mathbf{a}} \end{array}},\ y_{\mathbf{a}},\ z_{\mathbf{a}} \right)^{\mathrm{T}}$$

$$\mathbf{p}_y(\mathbf{a}) = \left(x_{\mathbf{a}},\ \frac{\begin{array}{c} r - c_{000}\,(1-x_{\mathbf{a}})(1-z_{\mathbf{a}}) \\ -\,c_{100}\,x_{\mathbf{a}}\,(1-z_{\mathbf{a}}) \\ -\,c_{001}\,(1-x_{\mathbf{a}})\,z_{\mathbf{a}} - c_{101}\,x_{\mathbf{a}}\,z_{\mathbf{a}} \end{array}}{\begin{array}{c} (c_{010}-c_{000})(1-x_{\mathbf{a}})(1-z_{\mathbf{a}}) \\ +\,(c_{110}-c_{100})\,x_{\mathbf{a}}\,(1-z_{\mathbf{a}}) \\ +\,(c_{011}-c_{001})(1-x_{\mathbf{a}})\,z_{\mathbf{a}} \\ +\,(c_{111}-c_{101})\,x_{\mathbf{a}}\,z_{\mathbf{a}} \end{array}},\ z_{\mathbf{a}} \right)^{\mathrm{T}}$$

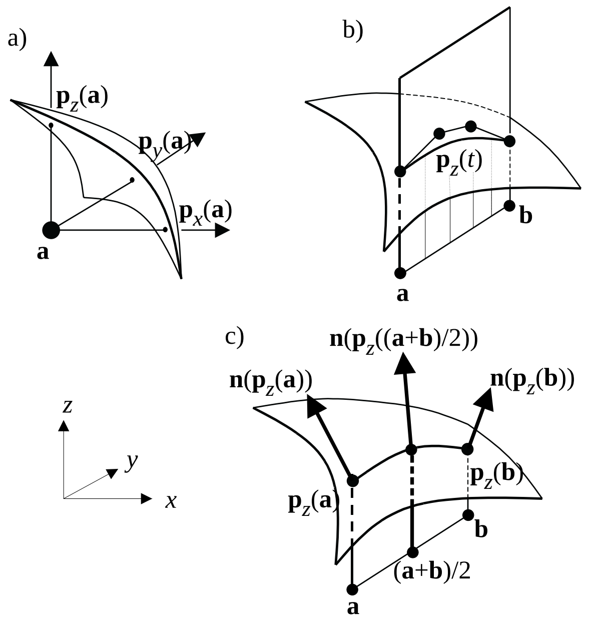
a)
$\mathbf{p}_z(\mathbf{a})$
$\mathbf{p}_y(\mathbf{a})$
$\mathbf{p}_x(\mathbf{a})$
a
b)
$\mathbf{p}_z(t)$
b
a
c)
$\mathbf{n}(\mathbf{p}_z((\mathbf{a}+\mathbf{b})/2))$
$\mathbf{n}(\mathbf{p}_z(\mathbf{a}))$
$\mathbf{n}(\mathbf{p}_z(\mathbf{b}))$
z
y
x
$\mathbf{p}_z(\mathbf{a})$
$\mathbf{p}_z(\mathbf{b})$
b
$(\mathbf{a}+\mathbf{b})/2$
a

Figure 3: *(a) Projections $\mathbf{p}_x(\mathbf{a})$, $\mathbf{p}_y(\mathbf{a})$, $\mathbf{p}_z(\mathbf{a})$ of a point* **a** *onto the contour in x-, y- and z-direction. (b) Projection $\mathbf{p}_z(t)$ of the line segment $(1-t)\,\mathbf{a} + t\,\mathbf{b}$ onto the contour is a rational cubic curve. (c) Configuration for computing the control points of $\mathbf{p}_z(t)$.*

$$\mathbf{p}_z(\mathbf{a}) = \left(x_\mathbf{a},\ y_\mathbf{a},\ \frac{\begin{array}{c} r - c_{000}\,(1-x_\mathbf{a})(1-y_\mathbf{a}) \\ - c_{010}\,(1-x_\mathbf{a})\,y_\mathbf{a} \\ - c_{100}\,x_\mathbf{a}\,(1-y_\mathbf{a}) - c_{110}\,x_\mathbf{a}\,y_\mathbf{a} \end{array}}{\begin{array}{c} (c_{001}-c_{000})(1-x_\mathbf{a})(1-y_\mathbf{a}) \\ + (c_{011}-c_{010})(1-x_\mathbf{a})\,y_\mathbf{a} \\ + (c_{101}-c_{100})\,x_\mathbf{a}\,(1-y_\mathbf{a}) \\ + (c_{111}-c_{110})\,x_\mathbf{a}\,y_\mathbf{a} \end{array}} \right)^{\mathrm{T}}.$$

We call $\mathbf{p}_x(\mathbf{a})$, $\mathbf{p}_y(\mathbf{a})$, $\mathbf{p}_z(\mathbf{a})$ the *projections of* **a** *onto the contour* in x-, y- and z-direction. This means that we can construct three points on the contour for a given point **a** in a simple way. Figure 3(a) gives an illustration.

Given a point **a**, there is one and only one contour defined by (1) through it. We can compute its (unnormalized) normal vector $\mathbf{n}(\mathbf{a})$ in **a** by

$$\mathbf{n}(\mathbf{a}) = (s_x(x_\mathbf{a}, y_\mathbf{a}, z_\mathbf{a}), s_y(x_\mathbf{a}, y_\mathbf{a}, z_\mathbf{a}), s_z(x_\mathbf{a}, y_\mathbf{a}, z_\mathbf{a}))^{\mathrm{T}} \tag{3}$$

where s_x, s_y, s_z are the partial derivatives of s defined in (1).

Now we construct curves on the contour by projecting line segments onto it. Given is the line segment $\mathbf{x}(t) = (1-t)\,\mathbf{a} + t\,\mathbf{b}$. Then the curves

$$\mathbf{p}_x(t) = \mathbf{p}_x(\mathbf{x}(t)), \quad \mathbf{p}_y(t) = \mathbf{p}_y(\mathbf{x}(t)), \quad \mathbf{p}_z(t) = \mathbf{p}_z(\mathbf{x}(t)) \tag{4}$$

are obtained by projecting each point of $\mathbf{x}(t)$ onto the contour in x-, y-, or z-direction. Figure 3(b) gives an illustration for $\mathbf{p}_z(t)$.

It is a straightforward exercise in algebra to show that the curves $\mathbf{p}_x(t)$, $\mathbf{p}_y(t)$, $\mathbf{p}_z(t)$ on the contour defined by (1) and (2) are rational cubics. The curve $\mathbf{p}_z(t)$ can be expressed as

$$\mathbf{p}_z(t) = \frac{\sum_{i=0}^{3} w_i\,\mathbf{b}_i\,B_i^3(t)}{\sum_{i=0}^{3} w_i\,B_i^3(t)} \tag{5}$$

where $B_i^3(t)$ are the Bernstein polynomials (see [12]) and

$$\begin{aligned} & w_0 = z_{\mathbf{n}(\mathbf{p}_z(\mathbf{a}))}, \quad w_1 = \tfrac{4}{3}\, z_{\mathbf{n}(\mathbf{p}_z(\frac{\mathbf{a}+\mathbf{b}}{2}))} - \tfrac{1}{3}\, z_{\mathbf{n}(\mathbf{p}_z(\mathbf{b}))} \\ & w_3 = z_{\mathbf{n}(\mathbf{p}_z(\mathbf{b}))}, \quad w_2 = \tfrac{4}{3}\, z_{\mathbf{n}(\mathbf{p}_z(\frac{\mathbf{a}+\mathbf{b}}{2}))} - \tfrac{1}{3}\, z_{\mathbf{n}(\mathbf{p}_z(\mathbf{a}))} \\ & \mathbf{b}_0 = \mathbf{p}_z(\mathbf{a}), \quad \mathbf{b}_3 = \mathbf{p}_z(\mathbf{b}) \end{aligned} \tag{6}$$

$$\mathbf{b}_1 = \begin{pmatrix} \left(1 - \dfrac{w_0}{3\,w_1}\right) x_{\mathbf{b}_0} + \dfrac{w_0}{3\,w_1}\, x_{\mathbf{b}_3} \\ \left(1 - \dfrac{w_0}{3\,w_1}\right) y_{\mathbf{b}_0} + \dfrac{w_0}{3\,w_1}\, y_{\mathbf{b}_3} \\ \left(1 + \dfrac{w_3}{3\,w_1}\right) z_{\mathbf{p}_z(\frac{\mathbf{a}+\mathbf{b}}{2})} - \dfrac{w_3}{3\,w_1}\, z_{\mathbf{b}_3} \end{pmatrix}$$

$$\mathbf{b}_2 = \begin{pmatrix} \dfrac{w_3}{3\,w_2}\, x_{\mathbf{b}_0} + \left(1 - \dfrac{w_3}{3\,w_2}\right) x_{\mathbf{b}_3} \\ \dfrac{w_3}{3\,w_2}\, y_{\mathbf{b}_0} + \left(1 - \dfrac{w_3}{3\,w_2}\right) y_{\mathbf{b}_3} \\ \left(1 + \dfrac{w_0}{3\,w_2}\right) z_{\mathbf{p}_z(\frac{\mathbf{a}+\mathbf{b}}{2})} - \dfrac{w_0}{3\,w_2}\, z_{\mathbf{b}_0} \end{pmatrix}.$$

Note that in (6) the weights $w_0, \ldots, w_3$ are defined by the z-coordinates of certain normal vectors of the contour. For example, w_0 is defined by the z-component of the contour normal at $\mathbf{p}_z(\mathbf{a})$. Figure 3(c) gives an illustration of the components used in (6). The curves $\mathbf{p}_x(t)$, $\mathbf{p}_y(t)$ can be computed as rational cubics in a similar way.

Now we extend the concept of curves on the contour to parametric surfaces on the contour defined by (1) and (2). Given is a triangle

$$\mathbf{x}(u, v, w) = u\,\mathbf{a} + v\,\mathbf{b} + w\,\mathbf{c} \tag{7}$$

in barycentric coordinates of the vertices **a**, **b**, **c**, i.e. $u + v + w = 1$. Then $\mathbf{p}_x(\mathbf{x}(u, v, w))$, $\mathbf{p}_y(\mathbf{x}(u, v, w))$, $\mathbf{p}_z(\mathbf{x}(u, v, w))$ are the projections of **x** onto the contour defined by (1) and (2). Figure 4(a) gives an illustration for $\mathbf{p}_z(\mathbf{x})$.

The surfaces $\mathbf{p}_x(\mathbf{x}(u, v, w))$, $\mathbf{p}_y(\mathbf{x}(u, v, w))$, $\mathbf{p}_z(\mathbf{x}(u, v, w))$ are rational cubics. For example, $\mathbf{p}_z(\mathbf{x}(u, v, w))$ can be described as a rational Bézier triangle

$$\mathbf{p}_z(\mathbf{x}(u, v, w)) = \frac{\sum_{i+j+k=3} w_{ijk}\,\mathbf{b}_{ijk}\,B_{ijk}^3(u, v, w)}{\sum_{i+j+k=3} w_{ijk}\,B_{ijk}^3(u, v, w)} \tag{8}$$

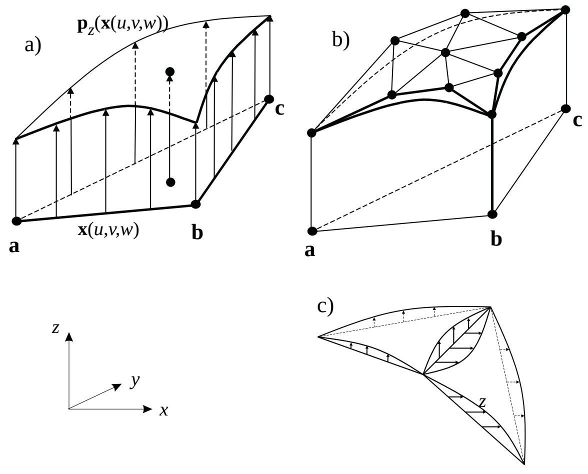

Figure 4: *(a)* $\mathbf{p}_z(\mathbf{x}(u,\ v,\ w))$ *is obtained by projecting every point of* $\mathbf{x}(u,\ v,\ w) = u\,\mathbf{a} + v\,\mathbf{b} + w\,\mathbf{c}$ *into z-direction onto the contour defined by* (1) *and* (2). *(b)* $\mathbf{p}_z(\mathbf{x}(u,\ v,\ w))$ *is a rational cubic surface. (c) Two adjacent MC triangles projected in different directions: the resulting contour patches have gaps.*

where the Bézier points and their weights on the boundary curves can be computed as in (6) above, and

$$w_{111} = \frac{w_{201} + w_{102} + w_{021} + w_{012} + w_{120} + w_{210}}{4} - \frac{w_{300} + w_{030} + w_{003}}{6}$$

$$\begin{aligned} x_{\mathbf{b}_{111}} = {} & \left(\frac{w_{012} + w_{021}}{4\,w_{111}} - \frac{w_{030} + w_{003}}{12\,w_{111}}\right) x_{\mathbf{a}} \\ & + \left(\frac{w_{102} + w_{201}}{4\,w_{111}} - \frac{w_{300} + w_{003}}{12\,w_{111}}\right) x_{\mathbf{b}} \\ & + \left(\frac{w_{120} + w_{210}}{4\,w_{111}} - \frac{w_{300} + w_{030}}{12\,w_{111}}\right) x_{\mathbf{c}} \end{aligned}$$

$$\begin{aligned} y_{\mathbf{b}_{111}} = {} & \left(\frac{w_{012} + w_{021}}{4\,w_{111}} - \frac{w_{030} + w_{003}}{12\,w_{111}}\right) y_{\mathbf{a}} \\ & + \left(\frac{w_{102} + w_{201}}{4\,w_{111}} - \frac{w_{300} + w_{003}}{12\,w_{111}}\right) y_{\mathbf{b}} \\ & + \left(\frac{w_{120} + w_{210}}{4\,w_{111}} - \frac{w_{300} + w_{030}}{12\,w_{111}}\right) y_{\mathbf{c}} \end{aligned}$$

$$z_{\mathbf{b}_{111}} = \frac{\begin{array}{c} w_{210}\, z_{\mathbf{b}210} + w_{201}\, z_{\mathbf{b}201} + w_{120}\, z_{\mathbf{b}120} \\ + w_{021}\, z_{\mathbf{b}021} + w_{012}\, z_{\mathbf{b}012} + w_{102}\, z_{\mathbf{b}102} \end{array}}{4\,w_{111}} - \frac{w_{300}\, z_{\mathbf{p}_z(\mathbf{a})} + w_{030}\, z_{\mathbf{p}_z(\mathbf{b})} + w_{003}\, z_{\mathbf{p}_z(\mathbf{c})}}{12\,w_{111}}.$$

Figure 4(b) gives an illustration. The surfaces $\mathbf{p}_x(\mathbf{x}),\ \mathbf{p}_y(\mathbf{x})$ can be described in a similar way.

The basic idea of representing a contour defined by (1) and (2) is to apply the MC algorithm and project each of the resulting triangles onto the contour. Unfortunately, this may produce gaps for adjacent triangles if their projection directions differ. Figure 4(c) gives an example. To overcome this problem, we use trimmed surfaces of the triangular rational cubics instead of the triangular patches themselves. This way the domain of the patches is not a triangle but a more complex shape which is bounded by three rational cubic curves. We use the following algorithm to compute the surface patch over an MC triangle:

Given is the triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ which is obtained from the MC algorithm. This means that $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are on the contour.

1. Determine the projection directions $q_{\mathbf{ab}}, q_{\mathbf{bc}}, q_{\mathbf{ca}} \in \{x, y, z\}$ of the boundary curves.
2. Determine the projection direction $q_{\mathbf{abc}} \in \{x, y, z\}$ of the whole triangle.
3. Project the boundaries of the triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ in the directions defined in step 1 onto the contour. We obtain the curves

$$\begin{aligned} \mathbf{x}_{\mathbf{ab}}(t) &= \mathbf{p}_{q_{\mathbf{ab}}}((1-t)\,\mathbf{a} + t\,\mathbf{b}) \\ \mathbf{x}_{\mathbf{bc}}(t) &= \mathbf{p}_{q_{\mathbf{bc}}}((1-t)\,\mathbf{b} + t\,\mathbf{c}) \\ \mathbf{x}_{\mathbf{ca}}(t) &= \mathbf{p}_{q_{\mathbf{ca}}}((1-t)\,\mathbf{c} + t\,\mathbf{a}) \end{aligned}$$

   on the contour. The curves $\mathbf{x}_{\mathbf{ab}}, \mathbf{x}_{\mathbf{bc}}, \mathbf{x}_{\mathbf{ca}}$ are the boundary curves of the final patch over the triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$.
4. Project $\mathbf{x}_{\mathbf{ab}}, \mathbf{x}_{\mathbf{bc}}, \mathbf{x}_{\mathbf{ca}}$ in the direction $q_{\mathbf{abc}}$ into the plane defined by $\mathbf{a}, \mathbf{b}, \mathbf{c}$. We obtain the curves $\mathbf{y}_{\mathbf{ab}}, \mathbf{y}_{\mathbf{bc}}, \mathbf{y}_{\mathbf{ca}}$ in the plane $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ which are the boundary curves of the domain of the final patch.
5. Compute the trimmed surface of the projected patch $\mathbf{p}_{q_{\mathbf{abc}}}(u\,\mathbf{a} + v\,\mathbf{b} + w\,\mathbf{c})$ with $u + v + w = 1$. The domain of the trimmed surface is given by the boundary curves $\mathbf{y}_{\mathbf{ab}}, \mathbf{y}_{\mathbf{bc}}, \mathbf{y}_{\mathbf{ca}}$.

Figure 5 illustrates an example of this algorithm.

To complete the algorithm, we have to answer two questions:

(a) how to choose the projection directions $q_{\mathbf{ab}}, q_{\mathbf{bc}}, q_{\mathbf{ca}}$ and $q_{\mathbf{abc}}$;
(b) how to make sure that the cubic surfaces have all positive (or all negative) weights, i.e. no zeros in the denominator functions.

To (a): given the points $\mathbf{a} = (x_{\mathbf{a}}, y_{\mathbf{a}}, z_{\mathbf{a}})^{\mathrm{T}}$ and $\mathbf{b} = (x_{\mathbf{b}}, y_{\mathbf{b}}, z_{\mathbf{b}})^{\mathrm{T}}$, we choose $q_{\mathbf{ab}} = x$ if $\|x_{\mathbf{a}} - x_{\mathbf{b}}\| < \|y_{\mathbf{a}} - y_{\mathbf{b}}\|$ and $\|x_{\mathbf{a}} - x_{\mathbf{b}}\| < \|z_{\mathbf{a}} - z_{\mathbf{b}}\|$ and $\mathbf{a}$ and $\mathbf{b}$ are not on the same face of the MC cell, i.e. $\neg((x_{\mathbf{a}} = 0$ and $x_{\mathbf{b}} = 0)$ or $(x_{\mathbf{a}} = 1$ and $x_{\mathbf{b}} = 1))$. If the last named condition is false, the projection direction has to be chosen between y and z.

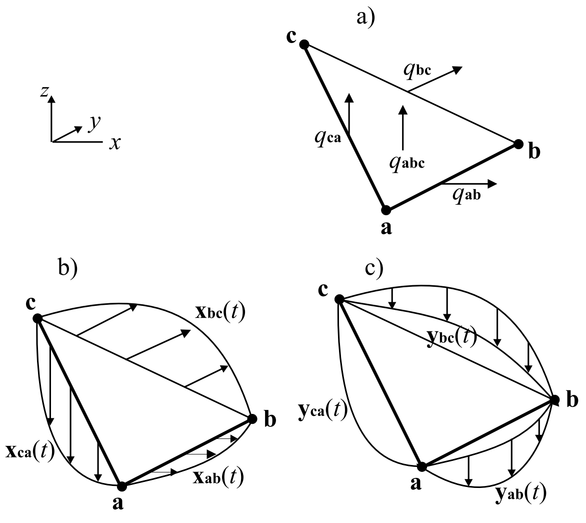

Figure 5: *Example of the algorithm for constructing a triangular patch over an MC triangle* $(\mathbf{a},\mathbf{b},\mathbf{c})$. *(a) Determine the projection directions; here we have chosen* $q_{\mathbf{ab}} = x$, $q_{\mathbf{bc}} = y$, $q_{\mathbf{ca}} = z$, $q_{\mathbf{abc}} = z$. *(b) Project the boundaries of the triangle onto the contour; here we obtain the boundary curves* $\mathbf{x_{ab}}(t) = \mathbf{p}_x((1-t)\,\mathbf{a} + t\,\mathbf{b})$, $\mathbf{x_{bc}}(t) = \mathbf{p}_y((1-t)\mathbf{b} + t\,\mathbf{c})$, $\mathbf{x_{ca}}(t) = \mathbf{p}_z((1-t)\,\mathbf{c} + t\,\mathbf{a})$ *on the contour. (c) Project* $\mathbf{x_{ab}}, \mathbf{x_{bc}}, \mathbf{x_{ca}}$ *in z-direction onto the plane defined by the triangle* $(\mathbf{a},\mathbf{b},\mathbf{c})$*; we obtain the planar curves* $\mathbf{y_{ab}}, \mathbf{y_{bc}}, \mathbf{y_{ca}}$ *which are the boundaries of the domain of the trimmed surface.*

The projection directions $q_{\mathbf{bc}}$ and $q_{\mathbf{ca}}$ are chosen in a similar way.

To compute $q_{\mathbf{abc}}$, we consider the normal $\mathbf{n} = (x_{\mathbf{n}}, y_{\mathbf{n}}, z_{\mathbf{n}})^{\mathrm{T}}$ of the triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$. We chose $q_{\mathbf{abc}} = x$ if $\|x_{\mathbf{n}}\| > \|y_{\mathbf{n}}\|$ and $\|x_{\mathbf{n}}\| > \|z_{\mathbf{n}}\|$.

To (b): to avoid zeros in the denominator functions of the rational patches, we have to make sure that the projections of a MC triangle onto the contour consists of one connected surface patch, i.e. it does not have any discontinuities. To achieve this, we have to make sure that the triangular approximation of the contour obtained by the MC algorithm is topologically equivalent to the contour itself. To ensure this for any contour on any trilinear scalar field, we have to introduce an extension of the MC algorithm of [1] and [2]. This extension is treated in the next section.

3. Topologically Exact Marching Cubes

After the introduction of the original MC algorithm in [1] it became evident that there are topological ambiguities in it. One way of finding a topologically exact approximation of the intersection curves of the contour and the faces of the cells has been introduced in [2] and [13]. There the results

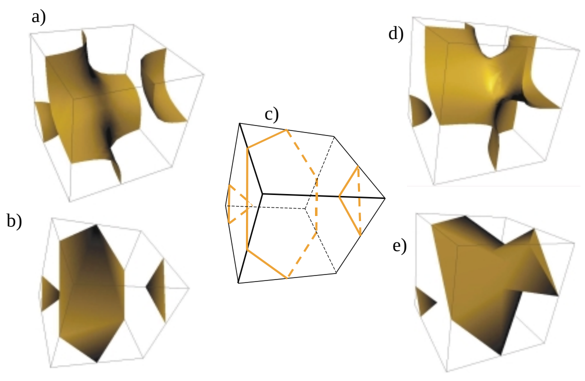

Figure 6: *(a and d). Two possible contours of* (1) *and* (2) *where the MC algorithm of* [1] *and* [2] *gives the same set of closed polygons on the faces of the cell shown in (c). Depending on certain inner points, the topologically exact triangulation is either (b) or (e).*

of the algorithm are a number of closed polygons on the faces of the cells which are triangulated in a certain way. Other approaches to solve the ambiguity problems of MC include [14–18]. All these approaches do not incorporate inner points into the triangulation. However, for some cases the incorporation of inner points is necessary for constructing a topologically exact approximation of the contour, i.e. an approximation which always has the same topology as the original contour. Figure 6 gives an example.

It is the purpose of this section to find a set of inner points which are sufficient to get a topologically exact triangulation for every case. To explain the main idea we start with an example. Given is the contour shown in Figure 7(a). The MC algorithm of [1] and [2] gives the closed polygon shown in Figure 7(b). We name the vertices of the polygon $\mathbf{v}_1, \ldots, \mathbf{v}_6$. Triangulating this polygon, the edges $(\mathbf{v}_2, \mathbf{v}_6)$ and $(\mathbf{v}_3, \mathbf{v}_5)$ must not be used because the contour does not have edges between these vertices on the upper face of the cell. Here it makes sense to define one inner point $\mathbf{v}$ on the contour and apply a triangulation shown in Figure 7(e). A good candidate for $\mathbf{v}$ is the point on the contour which has a contour normal in z-direction (see Figure 7a).

The example of Figure 7 gives the key to finding a set of inner points which are sufficient for a topologically exact triangulation. We consider all points of the contour which have a normal direction either in x-, y-, or z-direction.

Given a contour defined by (1) and (2), there are at most two points $\mathbf{x}_0$, $\mathbf{x}_1$ on the contour with a normal in x-direction. Similarly, there are at most two points $\mathbf{y}_0$, $\mathbf{y}_1$ on the contour with a normal in y-direction, and there are at most two points $\mathbf{z}_0$, $\mathbf{z}_1$ on the contour with a normal in

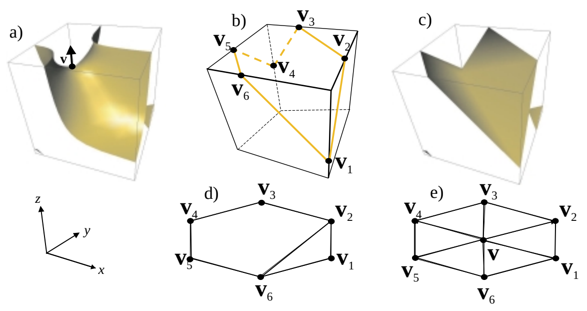

Figure 7: *(a) Contour and point* **v** *with normal in z-direction on it. (b) Closed polygons resulting from the MC algorithm of* [1] *and* [2]*. (d) Part of a wrong triangulation of (b). (e) Triangulation applied here. (c) Triangulation of (e) in* 3*D.*

z-direction. These six points can be computed as

$$\mathbf{x}_0 = \begin{pmatrix} x_m - x_p\sqrt{d} \\ y_m + y_p\sqrt{d} \\ z_m + z_p\sqrt{d} \end{pmatrix}, \quad \mathbf{x}_1 = \begin{pmatrix} x_m + x_p\sqrt{d} \\ y_m - y_p\sqrt{d} \\ z_m - z_p\sqrt{d} \end{pmatrix}$$
$$\mathbf{y}_0 = \begin{pmatrix} x_m + x_p\sqrt{d} \\ y_m - y_p\sqrt{d} \\ z_m + z_p\sqrt{d} \end{pmatrix}, \quad \mathbf{y}_1 = \begin{pmatrix} x_m - x_p\sqrt{d} \\ y_m + y_p\sqrt{d} \\ z_m - z_p\sqrt{d} \end{pmatrix} \qquad (9)$$
$$\mathbf{z}_0 = \begin{pmatrix} x_m + x_p\sqrt{d} \\ y_m + y_p\sqrt{d} \\ z_m - z_p\sqrt{d} \end{pmatrix}, \quad \mathbf{z}_1 = \begin{pmatrix} x_m - x_p\sqrt{d} \\ y_m - y_p\sqrt{d} \\ z_m + z_p\sqrt{d} \end{pmatrix}$$

with

$$x_m = \frac{\begin{array}{c}(c_{111}-c_{011})(r-c_{000})-(c_{110}-c_{010})\\ \times(r-c_{001})+(c_{100}-c_{000})(r-c_{011})\\ -(c_{101}-c_{001})(r-c_{010})\end{array}}{\begin{array}{c}2\,((c_{111}-c_{011})(c_{100}-c_{000})\\ -(c_{110}-c_{010})(c_{101}-c_{001}))\end{array}}$$

$$y_m = \frac{\begin{array}{c}(c_{111}-c_{101})(r-c_{000})-(c_{110}-c_{100})\\ \times(r-c_{001})+(c_{010}-c_{000})(r-c_{101})\\ -(c_{011}-c_{001})(r-c_{100})\end{array}}{\begin{array}{c}2\,((c_{111}-c_{101})(c_{010}-c_{000})\\ -(c_{110}-c_{100})(c_{011}-c_{001}))\end{array}}$$

$$z_m = \frac{\begin{array}{c}(c_{111}-c_{110})(r-c_{000})-(c_{101}-c_{100})\\ \times(r-c_{010})+(c_{001}-c_{000})(r-c_{110})\\ -(c_{011}-c_{010})(r-c_{100})\end{array}}{\begin{array}{c}2\,((c_{111}-c_{110})(c_{001}-c_{000})\\ -(c_{101}-c_{100})(c_{011}-c_{010}))\end{array}}$$

$$x_p = \frac{1}{\begin{array}{c}2\,((c_{111}-c_{011})(c_{100}-c_{000})\\ -(c_{110}-c_{010})(c_{101}-c_{001}))\end{array}}$$

$$y_p = \frac{1}{\begin{array}{c}2\,((c_{111}-c_{101})(c_{010}-c_{000})\\ -(c_{110}-c_{100})(c_{011}-c_{001}))\end{array}}$$

$$z_p = \frac{1}{\begin{array}{c}2\,((c_{111}-c_{110})(c_{001}-c_{000})\\ -(c_{101}-c_{100})(c_{011}-c_{010}))\end{array}}$$

$$\begin{aligned}
d &= a\,r^2 + b\,r + c \qquad (10)\\
a &= (-c_{111}+c_{110}+c_{101}-c_{100}\\
&\quad -c_{001}+c_{000}+c_{011}-c_{010})^2\\
b &= 2\,(c_{001}\,c_{110}+c_{011}\,c_{100}+c_{101}\,c_{010}+c_{111}\,c_{000})\\
&\quad \cdot(c_{111}+c_{000}+c_{101}+c_{110}\\
&\quad +c_{100}+c_{010}+c_{011}+c_{001})\\
&\quad -4\,(c_{000}\,c_{111}\,(c_{111}+c_{000})+c_{001}\,c_{110}\,(c_{110}+c_{001})\\
&\quad +c_{010}\,c_{101}\,(c_{101}+c_{010})+c_{011}\,c_{100}\,(c_{100}+c_{011}))\\
&\quad -4\,(c_{000}\,c_{110}\,c_{101}+c_{000}\,c_{011}\,c_{110}\\
&\quad +c_{000}\,c_{011}\,c_{101}+c_{110}\,c_{101}\,c_{011})\\
&\quad -4\,(c_{111}\,c_{001}\,c_{010}+c_{111}\,c_{100}\,c_{001}\\
&\quad +c_{010}\,c_{100}\,c_{111}+c_{001}\,c_{010}\,c_{100})\\
c &= -(c_{001}\,c_{110}+c_{011}\,c_{100}+c_{101}\,c_{010}+c_{111}\,c_{000})^2\\
&\quad +2\,(c_{111}^2\,c_{000}^2+c_{101}^2\,c_{010}^2+c_{001}^2\,c_{110}^2+c_{011}^2\,c_{100}^2)\\
&\quad +4\,(c_{000}\,c_{110}\,c_{101}\,c_{011}+c_{010}\,c_{100}\,c_{111}\,c_{001}).
\end{aligned}$$

Depending on the value d, all these six points are either real points on the contour, or all have imaginary values. If they are real (i.e. if $d > 0$), then (9) gives that the closed polygon $(\mathbf{x}_0, \mathbf{y}_1, \mathbf{z}_0, \mathbf{x}_1, \mathbf{y}_0, \mathbf{z}_1)$ lies on the edges of a box, as illustrated in Figure 8(a). Note that this polygon lies completely on the contour. We call this closed polygon *inner ring* in order to distinguish it from the closed polygons on the cell faces obtained by the MC algorithm of [1] and [2]. We call the closed polygons obtained there *outer rings*.

Now we can describe a topologically exact MC algorithm:

1. Create the closed polygons on the cell faces following [1] and [2]. We obtain up to three closed polygons and call them outer rings.
2. Compute the points of the inner ring by applying (9) and (10).
3. If the inner ring is not real or if the inner ring is completely outside the cell, then triangulate the outer rings independently of each other; otherwise continue with 4.
4. Check the connectivity between the inner ring and each of the outer rings. If the inner ring and one of the outer rings belong to the same contour segment: triangulate the area between the inner ring and this outer ring.
5. If only one outer ring was connected to the inner ring, the inner ring itself has to be triangulated.

Figure 8(b–f) illustrates this algorithm.

To check the connectivity between the inner ring and one outer ring, we intersect the lines of the inner ring with the

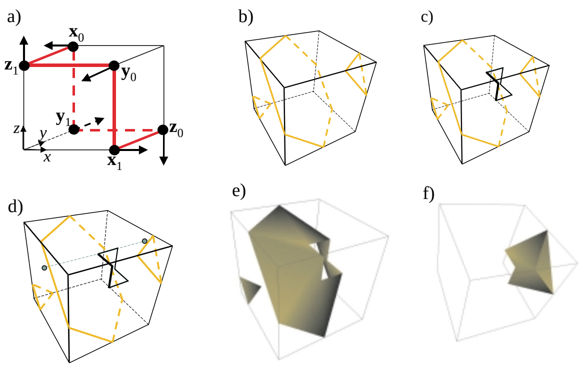

Figure 8: *(a) The points* $(\mathbf{x}_0, \mathbf{y}_1, \mathbf{z}_0, \mathbf{x}_1, \mathbf{y}_0, \mathbf{z}_1)$ *on the contour which have normals either in x-, y-, or z-direction form a closed polygon on the edges of a box—the inner ring; this inner ring lies completely on the contour. (b–f) Illustration of a topologically exact MC algorithm. (b) Create outer rings following* [1] *and* [2]*. (c) Compute inner ring. (d) Check connectivity between inner ring and outer rings by intersecting the lines of the inner ring with all faces of the cell; here the inner ring is connected to two outer rings. (e, f) Triangulate the areas between inner ring and outer rings.*

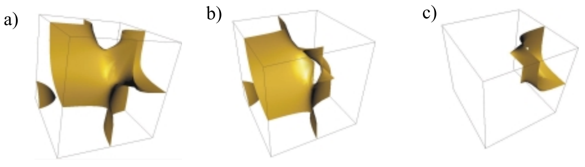

Figure 9: *(a) Exact contour over the triangulation shown in Figures* 8*(e) and (f). (b) Exact contour over triangulation in Figure* 8*(e). (c) Exact contour over triangulation in Figure* 8*(f).*

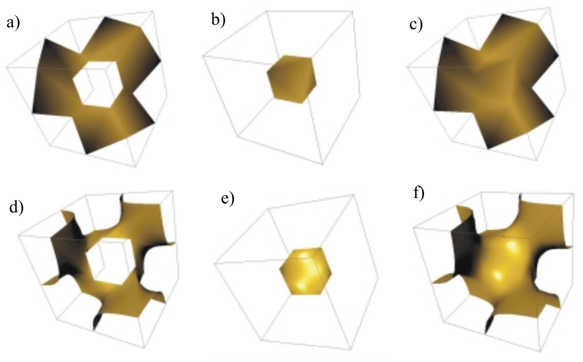

Figure 10: *Example of a contour with one outer ring consisting of* 12 *edges and the inner ring being completely inside the cell. (a) Triangulation between inner ring and outer ring. (b) Triangulation of inner ring. (c) Whole triangulation. (d–f) Exact contours over the triangulations (a–c).*

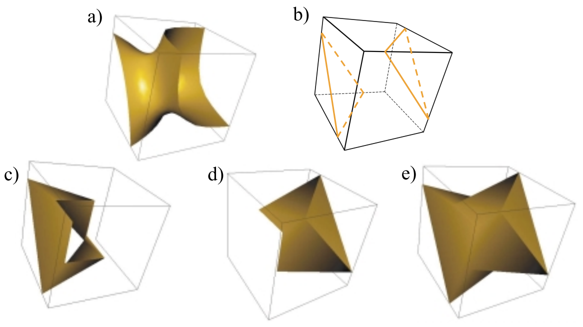

Figure 11: *(a) Contour which gives two outer rings and the inner ring completely inside the cell. (b) Outer rings. (c) Triangulation between inner ring and one outer ring. (d) Triangulation between inner ring and the other outer ring. (e) Whole triangulation.*

faces of the cell. If one of these intersection points lies on the outer ring, it is connected to the inner ring. Figure 8(*d*) illustrates this.

Figures 9–11 show examples of the application of the topologically exact MC algorithm as well as the computation of the exact contours based on these triangulations. In the examples of Figures 9 and 11 the MC algorithm of [1] and [2] would fail, i.e. gives topologically wrong triangular approximations.

Supplied with a topologically exact MC algorithm, we are able to compute the exact contour given by (1) and (2) as shown in Section 2.

In the next section we want to describe a method to achieve globally G^1 approximations of the contour.

4. Globally G^1 Contours

As already illustrated in Figure 2(*b*), the exact contour of a piecewise trilinear scalar field is G^∞ continuous inside a cell but only G^0 continuous across the cell boundaries. These discontinuities have a significant visual effect on the resulting contours. Hence a number of approaches to obtain globally G^1 continuous contours have been proposed.

One way of getting G^1 continuous contours is the application of higher order interpolation schemes of the cells. In [14] a piecewise tricubic interpolation of the cells is suggested. This way the authors obtain a globally C^1 continuous scalar field (and thus G^1 continuous contours). Beside the fact that piecewise tricubic interpolation is rather time consuming, no appropriate triangulation scheme seems to be known for this kind of scalar field. In [19], 3D blending functions are used to eliminate the “overshooting effect” of the piecewise tricubic interpolation. Here too, appropriate surface extraction algorithms are not available. In [20], the application of higher order interpolations was studied for 2D scalar fields. Here bicubic interpolations of the scalar field were used. To preserve the topology of the original scalar field, “damped partial derivatives” were used to restrict the interpolation parameters. In general, the application of higher order interpolation schemes may lead to complicated contours including new topologies and self-intersections which cannot be handled with an MC-like algorithm. Figure 1(c) shows an example.

Another way to obtain G^1 continuous contours is the construction of a triangular surface over each triangle of the MC triangulation applying approaches like a Clough–Tocher interpolant [12,21] a Powell–Sabin interpolant [12,22], or Nielson’s C^1 interpolant [23]. These approaches work entirely on the MC triangulation and neglect the underlying trilinear scalar field. In particular, topological changes of the contour due to new self-intersections are possible.

In this section we introduce another approach which preserves the topology of the contours of a piecewise trilinear scalar field but gives globally G^1 contours. We achieve this by applying an appropriate reparametrization of the domain of the scalar field.

Given is a trilinear scalar field $s(x, y, z)$ defined by (1) and (2) over the domain $[0, 1]^3$. The definition of a monotonically increasing continuous one-to-one map

$$\mathbf{g} : [0,1]^3 \to [0,1]^3$$
$$(x, y, z) \to (x_{\mathbf{g}}(x, y, z),\ y_{\mathbf{g}}(x, y, z),\ z_{\mathbf{g}}(x, y, z))$$

creates a new scalar field over $[0, 1]^3$:

$$s_{\mathbf{g}}(x, y, z) = s(\mathbf{g}^{-1}(x, y, z)).$$

This new scalar field $s_{\mathbf{g}}(x, y, z)$ is a reparametrization of the scalar field $s(x, y, z)$. If a point (x, y, z) lies on the contour $s(x, y, z) = r$, the point $\mathbf{g}(x, y, z)$ lies on the contour $s_{\mathbf{g}}(x, y, z) = r$. Thus the contours of $s_{\mathbf{g}}(x, y, z) = r$ can be computed in a simple way: apply the map $\mathbf{g}$ to all contour points of $s(x, y, z) = r$. Since $\mathbf{g}$ is continuous and one-to-one, the contours of s and $s_{\mathbf{g}}$ have the same topology for any r. Figure 12 gives an example for the reparametrization $\mathbf{g}$ in 2D.

The idea is now to find appropriate maps $\mathbf{g}$ for each cell of the piecewise trilinear scalar field which makes it globally G^1 continuous. This way all contours of the scalar field become G^1 continuous as well. We consider an 1D example to explain how to choose the maps $\mathbf{g}$. (Since in the 1D example $\mathbf{g}$ simplifies to a scalar function from $\mathbb{R}$ to $\mathbb{R}$, we simply write g instead of $\mathbf{g}$ here.) Given are the scalar values c_i which define a piecewise linear 1D scalar field

$$s(x) = (1-t)\,c_i + t\,c_{i+1} \quad \text{with} \quad i = [x],\ t = x - [x].$$

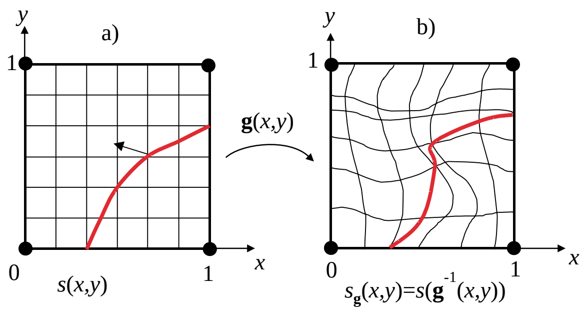

Figure 12: *(a) 2D bilinear scalar field over the unit square, shown are isoparametric lines of the domain and one contour curve. (b) Apply a regular reparametrization* $\mathbf{g}(x, y)$ *of the domain onto itself: isoparametric lines and contour line may change their shape but not their topology.*

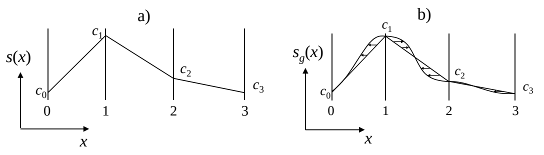

Figure 13: *(a) The piecewise linear curve* $(x, s(x))^{\mathrm{T}}$ *is* G^0 *continuous. (b) The curve* $(g(x), s(x))^{\mathrm{T}}$ *is* G^1 *continuous, it has the same shape (but another parameterization) as* $(x, s_g(x))^{\mathrm{T}}$.

See Figure 13(a) for an illustration. The curve $(x, s(x))^{\mathrm{T}}$ is G^0 continuous. We have to find a domain reparametrization $g(x)$ in such a way that the curve $(g(x), s(x))^{\mathrm{T}}$ is G^1 continuous and $g(i) = i$ and g is continuous and one-to-one. Figure 13(b) illustrated this.

To define g, we first have to estimate the tangent directions of the curve $(g(x), s(x))^{\mathrm{T}}$ at the junction points $x = i$. Let $\dot{c}_i$ be the estimated slope of $(g(x), s(x))^{\mathrm{T}}$ at $x = i$. Then $\dot{c}_i$ should have the following properties:

- If $(x, s(x))^{\mathrm{T}}$ is monotonically increasing both in the intervals $[i-1, i]$ and $[i, i+1]$, $\dot{c}_i$ should be positive. Figure 14(a) illustrates this.
- If $(x, s(x))^{\mathrm{T}}$ is monotonically decreasing both in the intervals $[i-1, i]$ and $[i, i+1]$, $\dot{c}_i$ should be negative.
- If $(x, s(x))^{\mathrm{T}}$ has the same slope in the intervals $[i-1, i]$ and $[i, i+1]$, $\dot{c}_i$ should have this slope as well. (This property ensures the linear precision of the contour, i.e. the contour of a globally linear scalar field will not effected by the reparametrization.) Figure 14(b) illustrates this.
- If $(x, s(x))^{\mathrm{T}}$ has a zero slope in either $[i-1, i]$ or $[i, i+1]$, $\dot{c}_i$ should be zero. Figure 14(c) illustrates this.

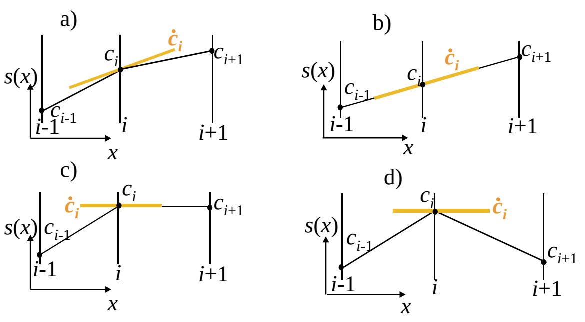

Figure 14: *Estimate the slope* $\dot c_i$ *in the point* $x = i$. *(a)* $\dot c_i > 0$ *if* $(c_i - c_{i-1})$ *and* $(c_{i+1} - c_i)$ *are positive. (b)* $\dot c_i = c_i - c_{i-1}$ *if* $c_i - c_{i-1} = c_{i+1} - c_i$. *(c)* $\dot c_i = 0$ *if* $c_i = c_{i-1}$ *or* $c_{i+1} = c_i$. *(d)* $\dot c_i > 0$ *if* $(c_i - c_{i-1})$ *and* $(c_{i+1} - c_i)$ *have different sign.*

- If the sign of the slopes of $(x, s(x))^{\mathrm T}$ in the intervals $[i-1, i]$ and $[i, i+1]$ differs, $\dot c_i$ should be zero. (This avoids an overshooting effect of the curve $(g(x), s(x))^{\mathrm T}$ and therefore the creation of new contours.) Figure 14(d) illustrates this.
- $\dot c_i$ should be continuous concerning changes of c_{i-1}, c_i, and c_{i+1}.

One estimation of $\dot c_i$ which fulfills the conditions above is

$$\dot c_i = \begin{cases} \frac{2(c_i - c_{i-1})(c_{i+1}-c_i)}{c_{i+1}+c_{i-1}} & \text{if } \quad (c_i - c_{i-1}) \\ & \quad \cdot (c_{i+1} - c_i) \geqslant 0 \\ 0 & \text{else.} \end{cases} \tag{11}$$

Now we consider the interval $[i, i+1]$ with the given values c_i, c_{i+1} and the estimated slopes $\dot c_i$, $\dot c_{i+1}$ using (11). Furthermore, we use a local parameter $t \in [0,1]$ for this interval which gives $s(t) = (1-t)\, c_i + t\, c_{i+1}$. Since

$$\dot c_i = \frac{\dot s(0)}{\dot g(0)}, \quad \dot c_{i+1} = \frac{\dot s(1)}{\dot g(1)}$$

and $\dot s(0) = \dot s(1) = c_{i+1} - c_i$, we obtain the following conditions for the monotonous function $g(t)$ in $t \in [0,1]$:

$$\begin{aligned} g(0) &= 0, \quad g(1) = 1 \\ \dot g(0) &= \frac{c_{i+1} - c_i}{\dot c_i}, \quad \dot g(1) = \frac{c_{i+1} - c_i}{\dot c_{i+1}}. \end{aligned} \tag{12}$$

To find an appropriate function g which fulfills (12), we have to keep in mind that $\dot g(0)$ and $\dot g(1)$ can attain any value between 0 and $+\infty$. Thus $g(t)$ cannot be described by a polynomial function of a fixed degree. Instead we define $g(t)$ as a combination of 4 basis functions $g_1(t)$, $g_2(t)$, $g_4(t)$, $g_5(t)$. These functions are illustrated in Figure 15.

The function $g_1(t)$ is obtained by reparametrizing the parabola defined by the Bézier points $\mathbf b^1_0 = (0,0)^{\mathrm T}$, $\mathbf b^1_1 = (x^1_1, 0)^{\mathrm T}$, $\mathbf b^1_2 = (1, 0.5)^{\mathrm T}$. This gives $g_1(t) = y_1({x_1}^{-1}(t))$ with $(x_1(t), y_1(t))^T = \sum_{i=0}^2 \mathbf b^1_i\, B^2_i(t)$. Moving x^1_1 between 0.5 and 1, $\dot g_1(1)$ ranges between 1 and $+\infty$. The function $g_2(t) = \sum_{i=0}^3 y^2_i\, B^3_i(t)$ is a cubic with $y^2_0 = 0$, $y^2_1 = 0$, $y^2_3 = 0.5$. Moving y^2_2 between 1/6 and 0.5, $\dot g_2(1)$ ranges between 1 and 0. Furthermore we have $\dot g_1(0) = \dot g_2(0) = 0$.

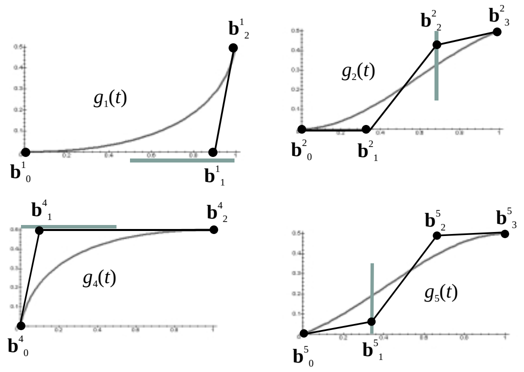

Figure 15: *Auxiliary functions of formulas* (13) *for defining* $g(t)$*; the functions* $g_1(t)$, $g_4(t)$ *are reparametrization of parabola segments;* $g_2(t)$, $g_5(t)$ *are cubic functions.*

While $g_1(t)$ and $g_2(t)$ are used to control the slope of g at $t = 1$, the functions $g_4(t)$ and $g_5(t)$ are used to control the slope at $t = 0$. The function $g_4(t)$ is a reparametrization of the parabola $(x_4(t), y_4(t))^{\mathrm T} = \sum_{i=0}^2 \mathbf b^4_i\, B^2_i(t)$ with $\mathbf b^4_0 = (0,0)^{\mathrm T}$, $\mathbf b^4_1 = (x^4_1, 0.5)$, $\mathbf b^4_2 = (1, 0.5)^{\mathrm T}$ which gives $g_4(t) = y_4({x_4}^{-1}(t))$. Moving x^4_1 between 0 and 0.5, $\dot g_4(0)$ ranges between $+\infty$ and 1. The function $g_5(t) = \sum_{i=0}^3 y^5_i\, B^3_i(t)$ is a cubic with $y^5_0 = 0$, $y^5_2 = 0.5$, $y^5_3 = 0.5$. Moving y^5_1 between 0 and 1/3, $\dot g_5(0)$ ranges between 0 and 1. Furthermore we have $\dot g_4(1) = \dot g_5(1) = 0$.

Now we can model the function $g(t)$ with $g(0) = 0$ and $g(1) = 1$ and given values for $\dot g(0)$ and $\dot g(1)$ between 0 and $+\infty$ as a linear combination of $g_1(t)$, $g_2(t)$, $g_4(t)$, $g_5(t)$:

$$\begin{aligned}
g_1(t) &= \frac{1}{8}\left(\frac{1 - 2\,\dot g(1) + \sqrt{(1-2\,\dot g(1))^2 + 4\,t\,\dot g(1)\,(1-\dot g(1))}}{(1-\dot g(1))}\right)^2 \\
g_2(t) &= 3\,t^2\,(1-t)\left(\tfrac12 - \tfrac13\,\dot g(1)\right) + \tfrac12\,t^3 \\
g_3(t) &= \begin{cases} g_1(t) & \text{for} \quad \dot g(1) > 1 \\ g_2(t) & \text{for} \quad 0 \leqslant \dot g(1) \leqslant 1 \end{cases} \\
g_4(t) &= \frac{2\,t\,\dot g(0)\,(1-\dot g(0)) + (1 - 2\,\dot g(0)) \cdot (1 - \sqrt{1 - 4\,t\,\dot g(0)\,(1-\dot g(0))})}{4\,(1-\dot g(0))^2} \\
g_5(t) &= t\,(1-t)^2\,\dot g(0) + \tfrac32\,t^2\,(1-t) + \tfrac12\,t^3
\end{aligned} \tag{13}$$

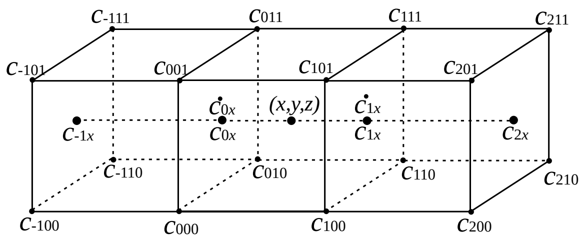

Figure 16: *The reparametrization $x_{\mathbf{g}}(x,y,z)$ in x-direction is computed using $\dot{c}_{0x}$ and $\dot{c}_{1x}$. $\dot{c}_{0x}$ is estimated by c_{-1x}, c_{0x}, c_{1x} while $\dot{c}_{1x}$ is estimated by c_{0x}, c_{1x}, c_{2x}.*

$$g_6(t) = \begin{cases} g_4(t) & \text{for} \quad \dot{g}(0) > 1 \\ g_5(t) & \text{for} \quad 0 \leqslant \dot{g}(0) \leqslant 1 \end{cases}$$
$$g(t) = g_3(t) + g_6(t).$$

Note that

$$\lim_{\dot{g}(1)=1} g_1(t) = \tfrac{1}{2}\,t^2, \quad \lim_{\dot{g}(1)=+\infty} g_1(t) = 1 - \sqrt{1-t} - \tfrac{1}{2}\,t$$
$$\lim_{\dot{g}(0)=1} g_4(t) = t - \tfrac{1}{2}\,t^2, \quad \lim_{\dot{g}(0)=+\infty} g_4(t) = \sqrt{t} - \tfrac{1}{2}\,t.$$

Now we can apply this reparametrization to the 3D cells: given is a scalar field defined by (1) and (2) in the domain $[0,1]^3$. To find $\mathbf{g}(x,y,z) = (x_{\mathbf{g}}(x,y,z),\ y_{\mathbf{g}}(x,y,z),\ z_{\mathbf{g}}(x,y,z))$, we consider the functions

$$c_{0x}(y,z) = s(0,y,z), \quad c_{1x}(y,z) = s(1,y,z)$$
$$c_{-1x}(y,z) = s(-1,y,z), \quad c_{2x}(y,z) = s(2,y,z)$$

where c_{-1x} and c_{2x} are computed from the scalars adjacent to the cell considered here. The slope $\dot{c}_{0x}(y,z)$ on the cell face $x = 0$ in x-direction is estimated by (11) using the values $c_{-1x}(y,z)$, $c_{0x}(y,z)$, $c_{1x}(y,z)$. In a similar way, the slope $\dot{c}_{1x}(y,z)$ on the cell face $x = 1$ in x-direction is estimated by (11) using the values $c_{0x}(y,z)$, $c_{1x}(y,z)$, $c_{2x}(y,z)$. Then $x_{\mathbf{g}}(x,y,z)$ can be computed using (13) with

$$t = x$$
$$\dot{g}(0) = \frac{c_{1x}(y,z) - c_{0x}(y,z)}{\dot{c}_{0x}(y,z)}$$
$$\dot{g}(1) = \frac{c_{1x}(y,z) - c_{0x}(y,z)}{\dot{c}_{1x}(y,z)}$$
$$x_{\mathbf{g}}(x,y,z) = g(t).$$

Figure 16 gives an illustration. The reparametrization in y- and z-direction, $y_{\mathbf{g}}(x,y,z)$ and $z_{\mathbf{g}}(x,y,z)$, are computed in a similar way.

The reparametrization introduced above is applied locally to all cells of the data volume. The global reparametrization $\mathbf{g}(x,y,z)$ obtained this way gives a globally G^1 scalar field $s_{\mathbf{g}}(x,y,z)$. Thus all contours of $s_{\mathbf{g}}$ are G^1 as well.

5. Results

We applied the algorithms to construct the exact contour of (1) and (2) as well as its G^1-modification both to theoretical and practical data sets.

Figure 17 shows the scalar field $s(x,y,z) = x^2 + y^2 + z^2$ which is sampled by a $3 \times 3 \times 3$ grid in the domain $[-1,1]^3$. Obviously the contours of s are concentric spheres. Figure 17(*a*) shows the result of the MC algorithm for $r = 0.9$: the sphere is approximated by 8 triangles. Figure 17(*b*) shows the exact contours of the piecewise trilinear interpolation. Although this shape comes closer to a sphere, we can still see discontinuities of the surface across the faces of the cell. Figure 17(*c*) shows the G^1 modification of the exact contour of Figure 17(*b*). Note that this is not exactly a sphere although it almost looks like it.

Figure 18 shows a $5 \times 5 \times 5$ hexahedral grid with random scalar values between 0 and 1 at the grid points, and $r = 0.5$. Figure 18(*a*) shows the result of the topologically exact MC algorithm where the triangles inside a cell are Phong shaded. Figure 18(*b*) shows the exact contour of the scalar field. We can clearly see the discontinuities of the surface across the cell faces. These discontinuities disappear in the G^1 modification shown in Figure 18(*c*).

Figure 19 shows a downsampled version of the data set of Figure 1(*a*). Originally consisting of $256 \times 256 \times 109$ grid points, this version has only a $51 \times 51 \times 35$ grid resolution. The result of the MC algorithm shown in Figure 19(*a*) consists of 93.636 triangles and shows clearly a "staircase effect" due to the low sample rate. The exact contour in Figure 19(*b*) shows hardly any visual differences to Figure 19(*a*). The G^1 modification of Figure 19(*c*) looks smoother but still has the "staircase effects."

Figures 20 and 21 show inner details of the data set shown in Figure 1(*a*). There are only few visual differences between the MC results (Figures 20*a* and 21*a*) and the exact contours (Figures 20*b* and 21*b*). The G^1 modifications (Figures 20*c* and 21*c*) look significantly smoother.

Figure 22 shows a magnified detail of Figure 21. Again the G^1 contour (Figure 22*c*) looks smoother than the exact contour (Figure 22*b*) and its MC approximation (Figure 22*a*).

We conclude that in most cases the exact contour of a piecewise trilinear scalar field gives only slight visual improvements against the MC approximation if the MC triangles are rendered using Phong shading. The G^1 contour gives improved visual results for low resolution volume data.

Acknowledgements

The author would like to thank Professor Heidrun Schumann from the University of Rostock for her constant support and encouragement of this work.

References

1. W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3d surface reconstruction algorithm. *Computer Graphics*, 21:163–169, 1987.

2. G. M. Nielson and B. Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proceedings of the IEEE Visualization '91*, Los Alamitos: IEEE Computer Society Press, pp. 83–91. 1991.

3. M. G. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the Siggraph '97*, pp. 209–216. 1997.

4. H. Hoppe. Progressive meshes. In *Proceedings of the Siggraph '96*, pp. 99–108. 1996.

5. K. J. Renze and J. H. Oliver. Generalized unstructured decimation. *IEEE Computer Graphics and Applications*, 16(6):24–32, 1996.

6. W. J. Schroeder, J. A. Zarge and W. E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–70, 1992.

7. R. S. Gallagher and J. C. Nagtegaal. An efficient 3D visualization technique for finite element models and other coarse volumes. In *Proceeding of the Siggraph '89*. 1989.

8. J. K. Johnstone and K. R. Sloan. Tensor product surfaces guided by minimal surface area triangulations. In G. Nielson and D. Silver (eds), *Proceedings of the IEEE Visualization '95*, Los Alamitos: pp. 254–261. 1995.

9. C. M. Grimm and J. F. Hughes. Smooth isosurface approximation. In B. Wyvill and M. P. Gascuel (eds), *Implicit Surface '95, Eurographics Workshop*, Blackwell Publishers, pp. 57–76. 1995.

10. B. Hamann, I. J. Trotts and G. Farin. Approximating contours of the piecewise trilinear interpolant using triangular rational quadratic Bézier patches. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):215–227, 1997.

11. P. Cignoni, F. Ganovelli, C. Montani and R. Scopigno. Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Computers & Graphics*, 24:399–418, 2000.

12. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. 4th edn. Academic Press, Boston, 1997.

13. B. Hamann. Modelling contours of trivariate data. *Modélisation Mathématique et Analysis Numérique*, 26(1):51–75, 1992.

14. A. van Gelder and J. Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337–375, 1994.

15. C. Montani, R. Scateni and R. Scopigno. A modified look-up table for implicit disambiguities of marching cubes. *The Visual Computer*, 10(6):353–355, 1994.

16. P. Ning and J. Bloomenthal. An evaluation of implicit surface tilers. *IEEE Computer Graphics and Applications*, 13(6):33–41, 1993.

17. Y. Zhou, W. Chen and Z. Tang. An elaborate ambiguity detection method for constructing isosurfaces within tetrahedral meshes. *Computers & Graphics*, 19(3):355–364, 1995.

18. B. K. Natarajan. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer*, 11(1):52–62, 1994.

19. K. Brodlie and P. Mashwama. Controlled interpolation for scientific visualization. In G. M. Nielson, H. Hagen and H. Müller (eds), *Scientific Visualization*, IEEE Computer Society Press, pp. 253–276. 1997.

20. C. L. Bajaj, V. Pascucci and D. R. Schikore. Visualization of scalar topology for structural enhancement. In D. Ebert, H. Hagen and H. Rushmeier (eds), *Proceedings of the IEEE Visualization '98*, Los Alamitos: IEEE Computer Society Press, pp. 51–58. 1998.

21. R. Barnhill and G. Farin. C^1 quintic interpolation over triangles: two explicit representations. *International Journal for Numerical Methods in Engineering*, 17:1763–1778, 1981.

22. M. Powell and M. Sabin. Piecewise quadratic approximation on triangles. *ACM Transactions on Mathematical Software*, 3(4):316–325, 1977.

23. G. M. Nielson. A method for interpolating scattered data based upon a minimum norm network. *Mathematics of Computation*, 40:253–271, 1983.

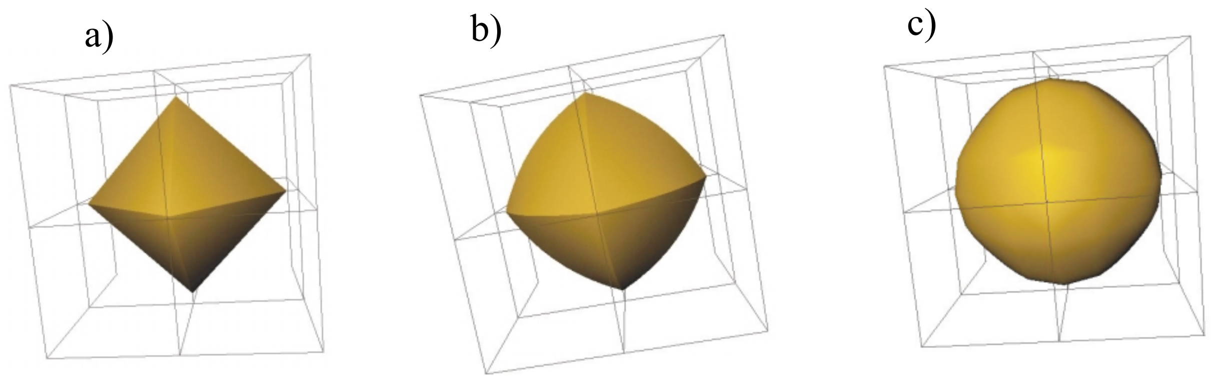

Figure 17: *Scalar field* $s(x, y, z) = x^2 + y^2 + z^2$*, sampled by a* $3 \times 3 \times 3$ *grid in the domain* $[-1, 1]^3$, $r = 0.9$. *(a) MC. (b) Exact contours. (c) Globally* G^1 *contours.*

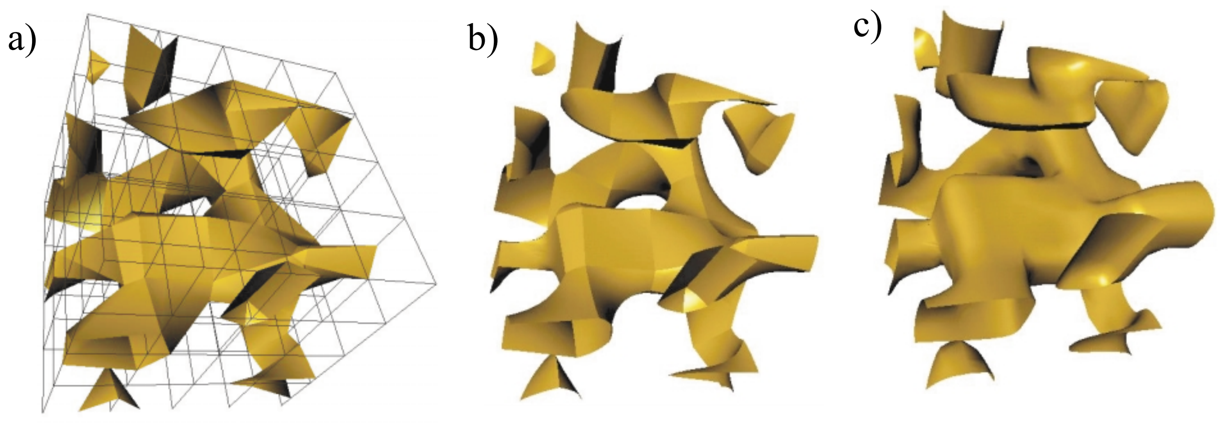

Figure 18: $5 \times 5 \times 5$ *random volume data set. (a) MC. (b) Exact contours. (c) Globally* G^1 *contours.*

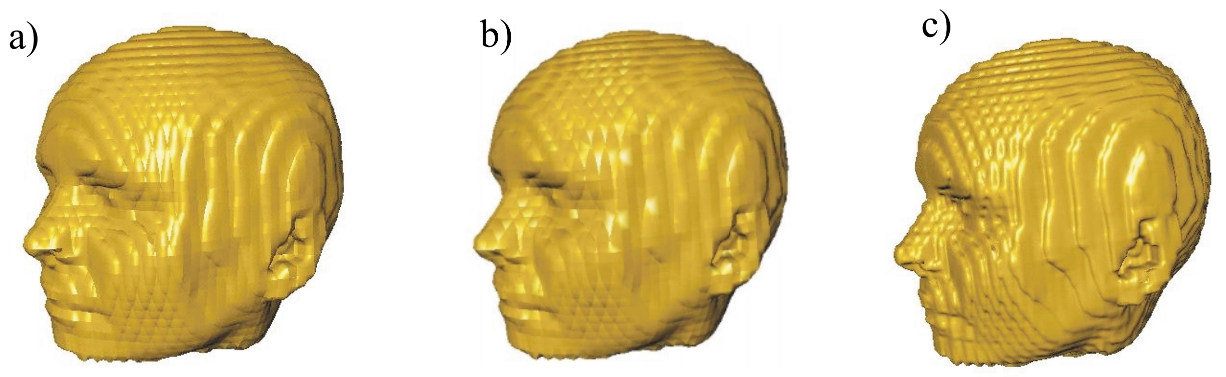

Figure 19: *Downsampled data set of Figure* 1*(a). (a) MC. (b) Exact contours. (c) Globally* G^1 *contours.*

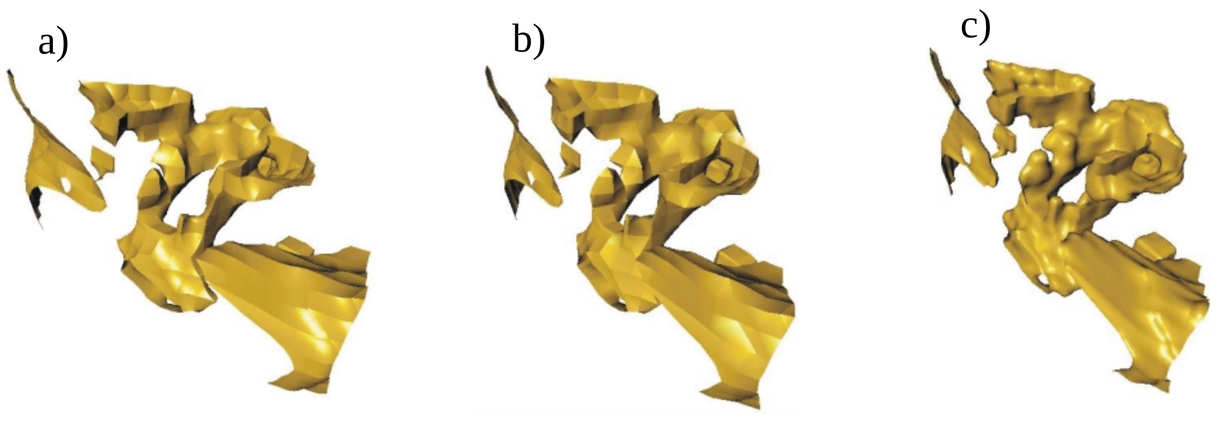

Figure 20: *Inner detail of the data set of Figure* 1*(a). (a) MC. (b) Exact contours. (c) Globally G^1 contours.*

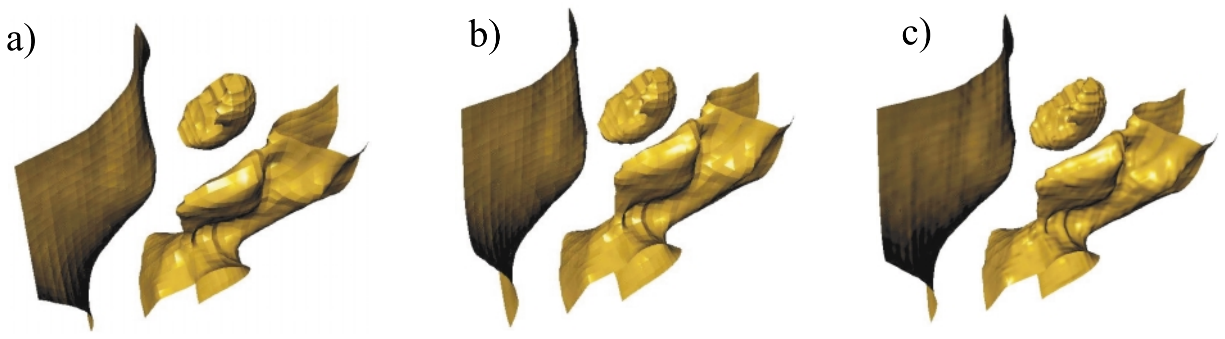

Figure 21: *Inner detail of the data set of Figure* 1*(a); (a) MC. (b) Exact contours. (c) Globally G^1 contours.*

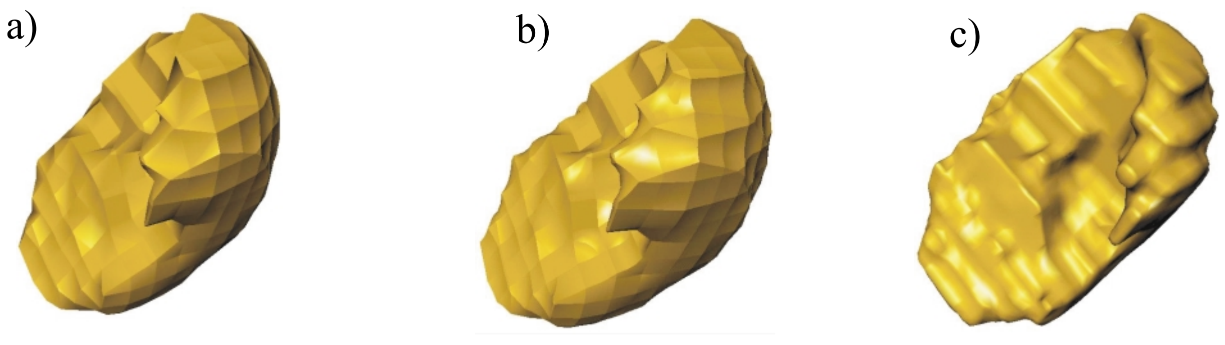

Figure 22: *Detail of Figure* 21*. (a) MC. (b) Exact contours. (c) Globally G^1 contours.*