Volume Refinement Fairing Isosurfaces

Martin Bertram*
TU Kaiserslautern

Abstract

We propose an interpolating refinement method for two- and three-dimensional scalar fields defined on hexahedral grids. Iterative fairing of the underlying contours (isosurfaces) provides the function values of new grid points. Our method can be considered as a non-linear variational subdivision scheme for volumes. It can be applied locally for adaptive mesh refinement in regions of high geometric complexity. We use our scheme to increase the quality of low-resolution data sets and to reduce interpolation artifacts in texture-based volume rendering.

CR Categories: G.1.2 [Numerical Analysis]: Approximation—Approximation of Surfaces and Contours; G.1.6 [Numerical Analysis]: Optimization—Constrained Optimization; I.4.3 [Image Processing and Computer Vision]: Enhancement—Smoothing

Keywords: adaptive mesh refinement, isosurfaces, subdivision, variational modeling, volume fairing.

1 Introduction

Volume rendering of scalar fields defined on regular hexahedral grids is mostly based on trilinear interpolation. In regions of high geometric complexity where the sampling rate is close to the Nyquist frequency, visualization often suffers from interpolation artifacts. These artifacts are particularly visible in isosurfaces, since the representation of the underlying scalar field with locally supported basis functions is not optimized for the representation of smooth contours. Isosurfaces can be emphasized in volume renderings by a proper transfer function [8] or they can be extracted by Marching Cubes [10].

Figures 1a) and b) illustrate the lacking smoothness of isolines based on bilinear and bicubic interpolation, despite of the fact that bicubic spline surfaces minimize thin-plate energy. All linear filtering and subdivision methods we have tested so far exhibit more or less the same problem. Hence, the interpolation artifacts of contours (isolines and -surfaces) may only be reduced effectively by a non-linear optimization method. In a previous work [1], we have presented an iterative variational fairing approach for 2D scalar fields based on bicubic splines, see figure 1c). The method increases the resolution by knot insertion and iteratively smoothes all isolines based on variational principles. The results of this method are promising, at the expense of extremely high computational cost (about 20 seconds for a 7×7 data set).

In the present work, we discretize the variational fairing method for smoothing piecewise linear scalar fields. Since the considered contours are piecewise linear, fairing is only necessary on the edges of the triangulated domain. In a further step, we adapt our method to the refinement of bilinear surfaces and trilinear volumes. The refined volumes interpolate the original data, deferring the most significant interpolation artifacts to finer scales where they can be eliminated by recursion. Our method can be globally applied, for example in combination with texture-based volume rendering [9, 5], or it can be used for local refinement.

These are the contents of our work: In section 2, we review related work. Our algorithm for the bivariate case (linear and bilinear scalar fields) is described in section 3, including numerical examples. The extension to trivariate scalar fields and its use for volume rendering is discussed in section 4. In section 5, we conclude our work.

2 Related Work

Visualization of three-dimensional scalar fields is either done by extraction of isosurfaces or by volume rendering, see for example [17, 6, 8]. In both cases, isosurface quality has a significant impact. Not only geometric smoothness, but also isosurface topology depends on the representation of the underlying scalar fields. In the case of trilinear scalar fields, isosurface topology is quite complicated. A variant of Marching Cubes [10], extracting topologically correct isosurfaces was recently devised [11].

For trilinear fields, *critical points* where isosurface topology changes are efficiently detected [14]. Unfortunately, the topology of a trilinear interpolant is often different from the topology of an original scalar field prior to discretization. The task is to find the best reconstruction of the original shape consistent with the discrete data.

Image processing techniques like anisotropic diffusion [15, 4] are capable of recognizing local features, but they often modify the data. Diffusion and filtering methods are mostly useful for smoothing noisy data. Diffusion-based fairing techniques are also applicable to the fairing of geometric shapes [3, 2].

Variational modeling [16, 7] is often used for fairing parametric surfaces. Using smooth basis functions, like B-splines or quadratic splines on tetrahedra [13], interpolation and fairness constraints can be specified for the scalar field. Only few approaches are capable of fairing implicit surfaces. Nielson et al. [12] propose a fairing method for single isosurfaces by constrained fairing of curves. The problem of fairing all isolines in a two-dimensional scalar field has been solved at the expense of high computational cost [1]. In the present work, we discretize the underlying representation providing a more efficient approach applicable to volume fairing.

*e-mail: bertram@informatik.uni-kl.de

IEEE Visualization 2004
October 10-15, Austin, Texas, USA
0-7803-8788-0/04/$20.00 ©2004 IEEE

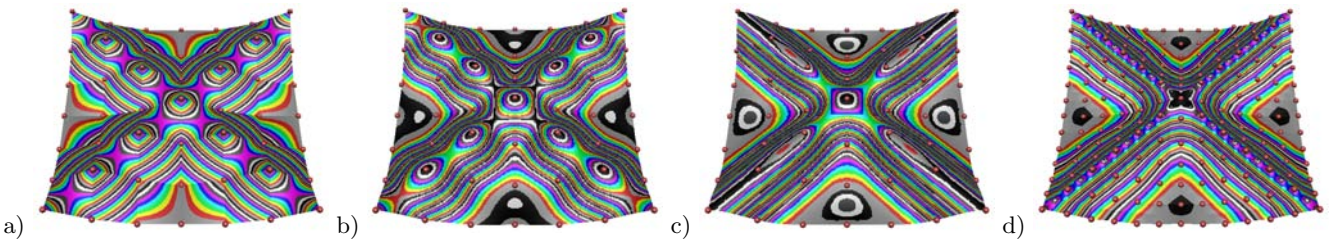

Figure 1: Isolines of 2D scalar field with diagonal features, defined by samples on a rectilinear grid. a) bilinear interpolation; b) bicubic thin-plate minimization; c) non-linear optimization with bicubic splines; d) proposed refinement method with bilinear interpolation.

3 Surface Refinement Fairing Isolines

In this section we develop an iterative fairing method for the contours of piecewise linear and bilinear scalar fields. This algorithm is extended to the trivariate case, later.

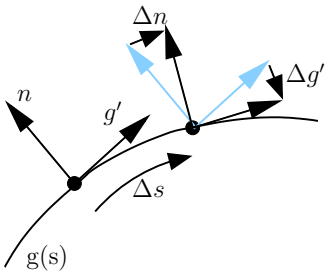

Figure 2: Variation of g' and n are equal in absolute value.

In a previous work [1], we considered the problem of minimizing the second derivative of an isoline $g(s)$ with arc-length parametrization,

$$\int \|g''(s)\|^2 \, ds \;\longrightarrow\; \min. \tag{1}$$

This is equivalent to

$$\int \|n \times \nabla n\|^2 \, ds \;\longrightarrow\; \min, \tag{2}$$

where $n(s)$ is the contour's normal corresponding to the scalar field's gradient, see figure 2.

This integral minimizes the normal's deviation along g. When fairing all contours of a scalar field, the arc-length parameter s can be eliminated by substituting the scalar field's domain parameters for s. The resulting optimization process is non-linear and requires expensive convolutions of smooth bivariate basis functions, like bicubic splines. (These convolutions do not have a simple algebraic solution due to a gradient-based weighting function.) To accelerate this approach, we consider piecewise linear scalar fields and redefine the optimization for discrete gradients.

3.1 Fairing Piecewise Linear Isolines

Consider a triangulated domain defined by a set of points $\mathbf{P}$ and a set of triangles $\mathbf{T}$. Every point $p_i \in \mathbf{P}$ has an associated scalar value f_i. For a subset $\mathbf{Q} \subset \mathbf{P}$ the corresponding scalar values are degrees of freedom (DOF's) used for optimization. The remaining function values are fixed and represent the interpolated data. Let α_{ab} be the adjacency relation for $a, b \in \mathbf{T}$, i.e., α_{ab} is true iff a and b are adjacent triangles.

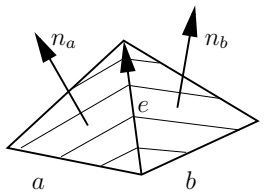

Figure 3: Isolines have constant normal vectors for every triangle.

We propose the following optimization:

$$\begin{aligned} r := \sum_{a,b:\ \alpha_{ab}} w_{ab}\ (n_a - n_b)^2 \;&\longrightarrow\; \min, \\ w_{ab} \;&=\; \left(\frac{1}{2}(n_a + n_b) \cdot e_{ab}\right)^2 \end{aligned} \tag{3}$$

where e_{ab} is the vector along the common edge, and n_a, n_b denote the scalar field's gradient in the triangles a and b, respectively. The gradient of a piecewise linear function is piecewise constant, see figure 3. Hence, the normal vectors of isolines vary only across the triangle boundaries. We note that we do not normalize the gradients n_a and n_b in our residual, emphasizing regions of great slope in the optimization.

The residual r in equation (3) is a function of the DOF's, i.e. of all scalar values $f_i : \ p_i \in \mathbf{Q}$. Their optimal values can be found by satisfying the necessary constraints

$$\frac{\partial r}{\partial f_i} \;=\; 0 \quad \forall i : \ p_i \in \mathbf{Q} \tag{4}$$

The weighting terms w_{ab} make this optimization non-linear, since for constant w_{ab}, above constraints would form a linear system of equations. The weights are quadratic proportional to the amount of isolines intersecting the edge e_{ab}. If the normals n_a and n_b are (more or less) orthogonal to the edge, the weight is small, since the isolines are (nearly) parallel to e_{ab}. The weight w_{ab} is maximal, if the average normal is parallel to e_{ab}. We take the square of the weighting term rather than its absolute value to make the residual a polynomial.

We propose the following algorithm:

1 initialize the DOF's, for example by linear interpolation of the given data $f_i : p_i \in \mathbf{P}\backslash\mathbf{Q}$.

2 For every $i : p_i \in \mathbf{Q}$ compute the coefficients of $\partial r/\partial f_i$, which is a cubic polynomial in f_i. Among the zero points of this partial derivative is the minimum for $r = f(f_i)$. Replace f_i by the zero point where the minimum is obtained.

3 Repeat step 2 either for a fixed number of iterations, or until the maximal correction of r is below a certain threshold.

3.2 Rectilinear Grid Refinement

In the case of a rectilinear grid with associated function values, a refinement step doubles the resolution in both directions. The function values at the new points representing the DOF's are initialized by bilinear interpolation. In a first approach, we triangulate the quadrilaterals of the refined grid by inserting consistently oriented diagonals. for this triangulation, we apply the algorithm described above.

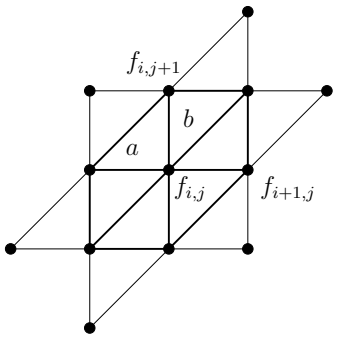

Figure 4: The residual at f_{ij} depends on all twelve edges of the six incident triangles.

In the following, we provide all relevant implementation details for step 2 of the algorithm updating a variable value f_{ij}. Relevant for the residual r as a function of f_{ij} are only the terms of equation (3) corresponding to the twelve edges of incident triangles, as illustrated in figure 4. All twelve terms are non-negative quartic polynomials of f_{ij}. To compute the coefficients of the local residual $r(f_{ij})$, we need to compute every term with its four partial derivatives with respect to f_{ij}. As an example, we provide these derivatives for the edges $a = e_{p_{i,j+1},p_{ij}}$ and $b = e_{p_{i+1,j+1},p_{ij}}$, according to figure 4.

For the term associated with edge a, we get

$$\begin{aligned} r_a &= w\ d, \\ w &= \left(\frac{1}{2}(n_1+n_2)\cdot(0,\ 1)^T\right)^2, \\ d &= (n_1-n_2)^2, \\ n_1 &= (f_{i,j}-f_{i-1,j},\ f_{i,j+1}-f_{i,j})^T, \\ n_2 &= (f_{i+1,j+1}-f_{i,j+1},\ f_{i,j+1}-f_{i,j})^T. \end{aligned} \tag{5}$$

Inserting the gradients n_1 and n_2 provides w, d and their derivatives with respect to f_{ij}:

$$\begin{aligned} w &= (f_{i,j+1}-f_{i,j})^2, \\ w' &= -2(f_{i,j+1}-f_{i,j}), \\ w'' &= 2, \\ d &= (f_{i,j}-f_{i-1,j}-f_{i+1,j+1}+f_{i,j+1})^2, \\ d' &= -2(f_{i,j}-f_{i-1,j}-f_{i+1,j+1}+f_{i,j+1}), \\ d'' &= 2. \end{aligned} \tag{6}$$

Since r_a is a quartic polynomial in f_{ij}, its coefficients may be determined from its value and four derivatives with respect to f_{ij}:

$$\begin{aligned} r_a' &= w'd\ +\ d'w, \\ r_a'' &= w''d\ +\ 2w'd'\ +\ wd'', \\ r_a^{(3)} &= 3(w''d'\ +\ w'd''), \\ r_a^{(4)} &= 6w''d''. \end{aligned} \tag{7}$$

In analogy to the term for edge a, all other terms of the local residual associated with horizontal and vertical edges are computed. As an example for a diagonal edge, we provide the term for edge b in figure 4:

$$\begin{aligned} r_b &= w\ d, \\ w &= \left(\frac{1}{2}(n_1+n_2)\cdot(1,\ 1)^T\right)^2, \\ d &= (n_1-n_2)^2, \\ n_1 &= (f_{i+1,j+1}-f_{i,j+1},\ f_{i,j+1}-f_{i,j})^T, \\ n_2 &= (f_{i+1,j}-f_{i,j},\ f_{i+1,j+1}-f_{i+1,j})^T. \end{aligned} \tag{8}$$

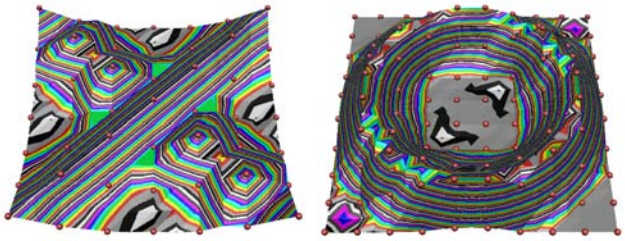

Figure 5: The triangulation has a significant impact on the solution, since features across triangle edges often cannot be properly represented.

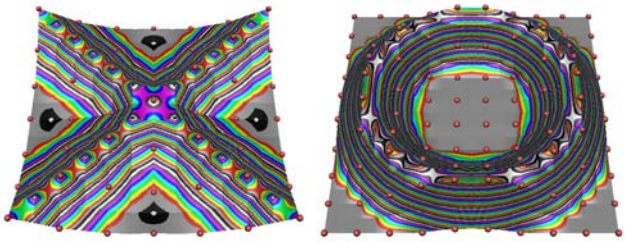

Figure 6: The combined minimization of both residuals overcomes the problem shown in figure 5. The results were obtained after ten iterations.

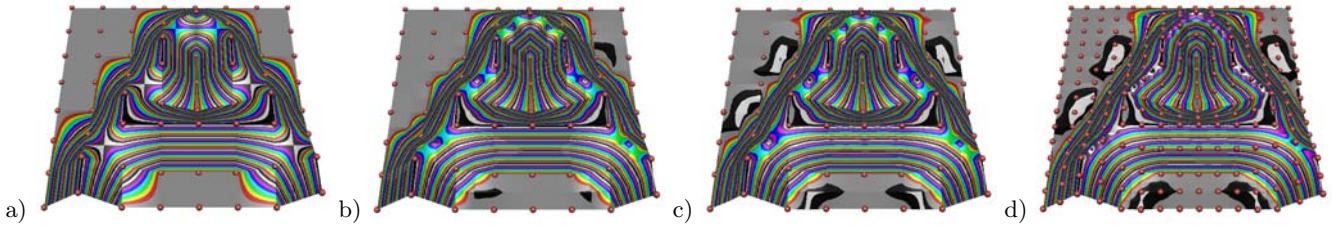

Figure 7: Results for 8×8 dataset. a) bilinear refinement; b) first iteration; c) ten iterations; d) additional refinement of c) after five iterations.

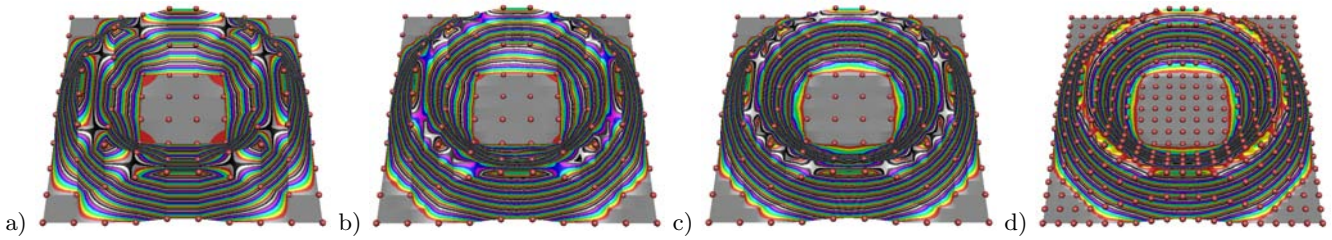

Figure 8: Results for 10×10 dataset, as in figure 7

Again, inserting the gradients n_1 and n_2 provides w and d:

$$\begin{aligned} w &= \frac{1}{4}(f_{i+1,j+1} - f_{i,j+1} + f_{i+1,j} - f_{i,j})^2 \\ &+ \frac{1}{4}(f_{i,j+1} - f_{i,j} + f_{i+1,j+1} - f_{i+1,j})^2, \\ d &= (f_{i+1,j+1} - f_{i,j+1} - f_{i+1,j} + f_{i,j})^2 \\ &+ (f_{i,j+1} - f_{i,j} - f_{i+1,j+1} + f_{i+1,j})^2. \end{aligned} \tag{9}$$

The coefficients for r_b are obtained in analogy to equation (7). The sum of the terms of all twelve edges of triangles incident with p_{ij} provide the coefficients for the local residual $r(f_{ij})$. These are all terms depending on f_{ij}.

Now, we need to modify f_{ij}, such that the local residual is minimized. Since $r(f_{ij})$ is a quartic non-negative polynomial, it can have at most two minima. The global minimum is among the roots of

$$\frac{\partial r}{\partial f_{ij}} = 0. \tag{10}$$

These can be found by solving a cubic equation. In our implementation, we used Newton iteration with the two starting points $min\{f_{i\pm1,j\pm1}\}$ and $max\{f_{i\pm1,j\pm1}\}$, providing an accurate estimate for both minima after five iterations on average.

When implementing and testing our algorithm, we found that the triangulation has a significant impact, see figure 5. When orienting all edges from lower left to upper right, features in the opposite direction can not be represented properly, and vice versa. To ensure that our algorithm provides a symmetric solution, we compute two local residuals, $r_{left}(f_{ij})$ and $r_{right}(f_{ij})$ where all edges have left and right orientation, respectively. We compute the minimum for both residuals and update f_{ij} with the value computed for the minimum of both.

By minimizing the smallest residual, the algorithm imposes the best orientation of edges for local optimization, see figure 6. We do not store and optimize the triangulation, since for adjacent points opposing orientations might be optimal, which would result in alternating edge flips during the iteration.

Our combined optimization algorithm for rectilinear grid refinement is summarized as follows:

1 subdivide the grid in both directions and initialize the new grid points using bilinear interpolation. These new values represent the DOF's for optimization, where the data located at the old grid points is fixed.

2 For every new grid point p_{ij}, compute the coefficients for both local residuals $r_{left}(f_{ij})$ and $r_{right}(f_{ij})$. Compute the minimum for both and replace f_{ij} by the corresponding value for the smaller residual.

3 Repeat step 2 for a fixed number of iterations.

For adaptive refinement on a rectilinear grid, this algorithm may be applied recursively. In the subsequent optimization steps less iterations are necessary, since the data becomes smoother after refinement.

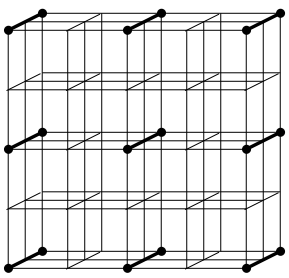

Figure 9: The intermediate slice does not contain data. To avoid diffusion of detail, we keep the scalar values located on the fat lines fixed and use all other values as DOF's.

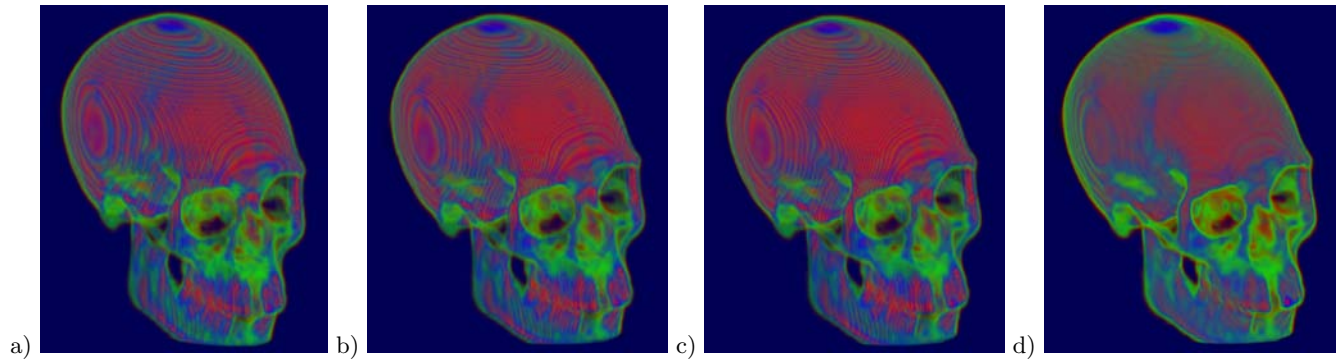

Figure 10: Texture-based volume rendering of skull data. a) trilinear interpolation; b) result after one iteration of the volume method; c) five volume iterations; d) additional refinement with two more iterations.

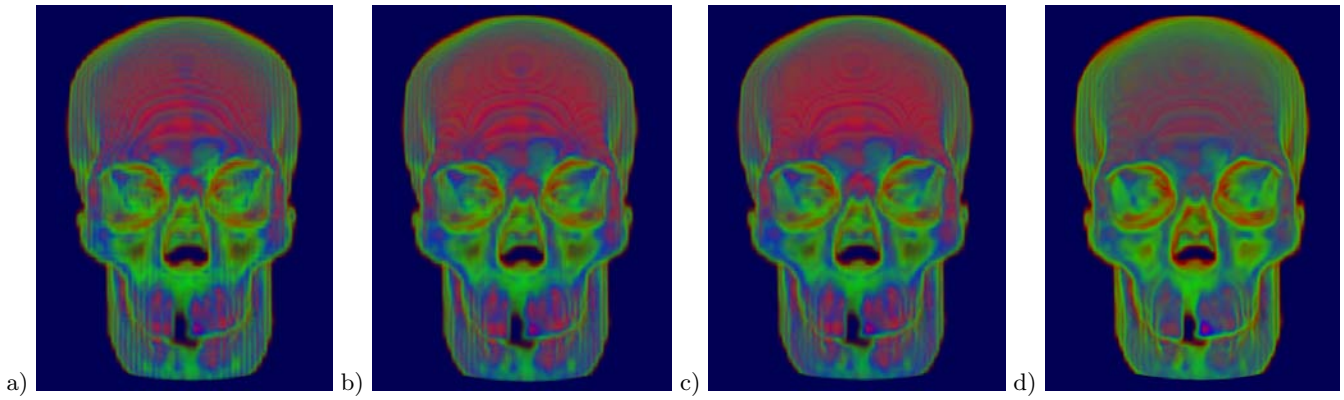

Figure 11: Skull volume from the front (same as figure 10).

3.3 Numerical Examples

We have implemented and tested our method for different data sets, see figures 7 and 8. Since bilinear interpolation is used for rendering, the interpolation artifacts are shifted to a finer scale where they can be removed by additional refinement steps.

The algorithm converges quickly, such that ten iterations for all DOF's are more than enough. Table 1 provides computation times for ten iterations obtained on a PC with a 2GHz processor.

data set	resolution	no. dof's	time [sec]
"X"	13×13	120	0.035
"A"	15×15	161	0.041
"O"	19×19	261	0.060
"X"	25×25	456	0.097
"A"	29×29	616	0.125
"O"	37×37	1008	0.200

Table 1: Computation times in seconds for ten iterations on small data sets. The resolutions are obtained after subdivision.

4 Volume Refinement

In the following, we extend our method to the fairing of isosurfaces in trivariate scalar fields. We provide numerical examples using texture-based volume rendering.

4.1 Fairing Isosurfaces

One way of generalizing the algorithm to hexahedral grids would be a decomposition into tetrahedra. In this case, the edge terms for triangles would simply become face terms. However, there are many choices decomposing hexahedra. First, every voxel could be subdivided into six tetrahedra, leaving a great number of possible triangulations for which residuals had to be computed resulting in prohibitive computational cost.

Second, every voxel could be subdivided into five tetrahedra, one located in the middle and four at certain corners. There exist two such decompositions, which had to be used alternately on the grid, to avoid cracking. In analogy to the planar case, two possible triangulations would be constructed (depending on the choice for the first voxel). However, the drawback of this decomposition is that every other vertex has 32 incident tetrahedra, and the remaining vertices have only eight. The two competing residuals would be based on a different number of faces, such that this choice of a grid would cause severe artifacts.

Facing this problem, we decided to apply the bivariate approach to the three sets of slices orthogonal to the canonical directions. This approach was already used for the fairing of single isosurfaces by smoothing three sets of isolines [12]. In our work, we use this approach for volume fairing, rather than considering one particular contour.

In contrast to updating every flexible grid value once per

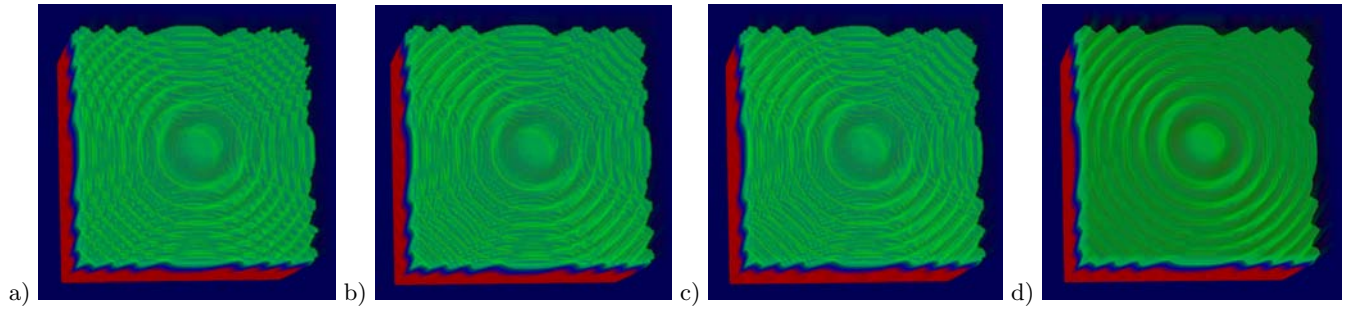

Figure 12: Marschner/Lobb volume rendering (same refinement as in figure 10).

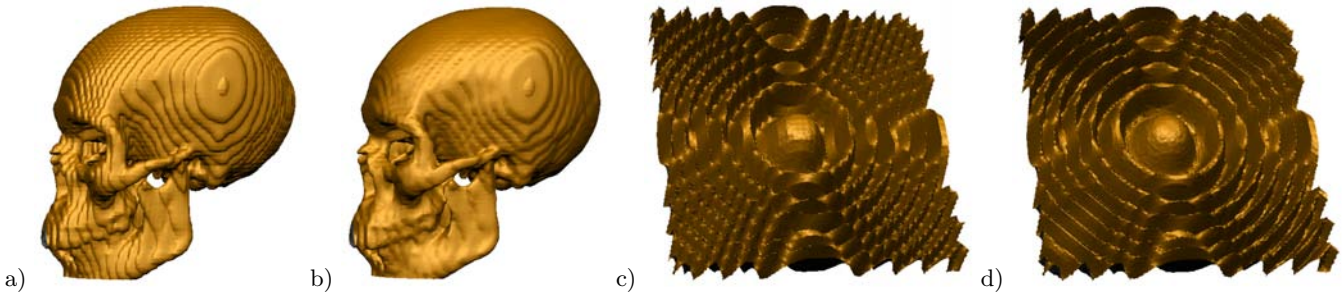

Figure 13: Effect of the algorithm on an isosurface extracted after two steps of refinement. For rendering, we use flat shading. a) skull with trilinear subdivision; b) optimized result (using 5 iterations before and 2 iterations after refinement); c) Marschner/Lobb with trilinear subdivision; d) optimized.

iteration, we compute one iteration of the planar method for each slice orthogonal to the x-, y-, and z-axis, respectively. When processing these slices, every other slice does not contain data, i.e. all grid values are flexible. To avoid a diffusion of geometric detail, we keep those values fixed that are located between two data points, see figure 9. These are initially determined by trilinear interpolation and are alterated by the passes for the two other directions. Every volume iteration is now composed of three passes, such that every DOF is updated two or three times, depending on its location on the grid.

4.2 Numerical Examples

We have applied our algorithm to the fairing of a 64^3 down-sampled computer tomography of a human skull and to the 41^3 Marschner/Lobb data set (see `http://www.volvis.org/`), which is interesting due to its high-frequency details.

Figures 10–12 show volume rendered results obtained by transparent textures [9]. For rendering, we do not resample the data, however, to avoid additional artifacts. Rearranging the texture planes is only necessary, when the volume is rotated by more than 45 degrees, such that the viewer is facing a different side of the cube. The colors in figures 10–12d) are slightly different, since these images were rendered at twice the resolution of texture planes.

Figures 10a-c) were rendered by 127 planes, each with a transparent 128×128 texture. The color and opacity of the texture (RGBA) is generated by a color-coded transfer function. We did not use gradient mapping, which would also be possible. For higher efficiency, the texture generation is independent of the viewpoint. We use different display lists for the front, side, and top views of the data set. On a consumer-grade PC, we obtained frame rates up to four frames per second.

The effect of our volume fairing method on a single isosurface is illustrated in figure 13. The Marschner/Lobb data set shows impressively that details above the Nyquist frequency cannot be reconstructed properly. For all other features, we obtain visually pleasing results. Computation times for one volume iteration (on the three sets of slices) are summarized in table 2.

data set	resolution	no. dof's	time [sec]
"mlobb"	81^3	462520	22.9
"skull"	127^3	1786293	85.3
"mlobb"	161^3	3641840	178.3
"skull"	253^3	14145894	676.0

Table 2: Computation times in seconds for one volume iteration obtained on a PC with a 2GHz processor. The resolutions after subdivision are listed.

5 Conclusions

We presented an iterative refinement algorithm fairing contours (isolines and isosurfaces) of scalar fields defined on regular grids. Our method significantly reduces the interpolation artifacts and reconstructs features in regions where the sampling distance is close to the Nyquist limit.

Numerical examples show that the quality of gridded data is significantly improved in regions of high geometric complexity. Coarse features traversing the grid diagonally appear also smoother. The algorithm can be used for adaptive

refinement, for example also when zooming into smaller regions of large-scale data sets. We apply our method for texture-based volume rendering exploiting graphics hardware to obtain nearly interactive frame rates.

References

[1] M. Bertram. Fairing scalar fields by variational modeling of contours. In *Proceedings of Visualization'03*, pages 387–392. IEEE, 2003.

[2] U. Clarenz, U. Diewald, and M. Rumpf. Nonlinear anisotropic diffusion in surface processing. In *Proceedings of Visualization'00*, pages 397–405 & 580. IEEE, 2000.

[3] M. Desbrun, M. Meyer, P. Schroeder, and A. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of Siggraph'99*, Computer Graphics, pages 317–324. ACM, 1999.

[4] U. Diewald, T. Preusser, and M. Rumpf. Anisotropic diffusion in vector field visualization on euclidean domains and surfaces. In *Transactions on Visualization and Computer Graphics*, volume 6, pages 139–149. IEEE, 2000.

[5] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of Eurographics / Siggraph Workshop on Graphics Hardware'01*, pages 9–16. ACM, 2001.

[6] T. Gerstner. Fast multiresolution extraction of multiple transparent isosurfaces. In *Proceedings of VisSym'03, Joint Eurographics and IEEE TCVG Symposium on Visualization*, Data Visualisation, pages 35–44 & 336. IEEE, 2001.

[7] H. Hagen, G. Brunnett, and P. Santarelli. Variational principles in curve and surface design. In *Surveys on Mathematics for Industry*, volume 3, pages 1–27, 1993.

[8] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Mller. Curvature-based transfer functions for direct volume rendering: methods and applications. In *Proceedings of Visualization'03*, pages 513–520. IEEE, 2003.

[9] E.C. LaMar, M.A. Duchaineau, B. Hamann, and K.I. Joy. Multiresolution techniques for interactive texture-based rendering of arbitrarily oriented cutting planes. In *Proceedings of VisSym'02, Joint Eurographics and IEEE TCVG Symposium on Visualization*, Data Visualisation, pages 105–114, 2000.

[10] W.E. Lorensen and H.E. Cline. Marching cubes: a high resolution 3d surface construction algorithm. In *Proceedings of Siggraph'87*, Computer Graphics, pages 163–169. ACM, 1987.

[11] G.M. Nielson. On marching cubes. In *Transactions on Visualization and Computer Graphics*, volume 9, pages 283–297. IEEE, 2003.

[12] G.M. Nielson, G. Graf, R. Holmes, A. Huang, and M. Phielipp. Shrouds: optimal separating surfaces for enumerated volumes. In *Proceedings of VisSym'03, Joint Eurographics and IEEE TCVG Symposium on Visualization*, Data Visualisation, pages 75–84 & 287, 2003.

[13] C. Roessl, F. Zeilfelder, G. Nuernberger, and H.-P. Seidel. Visualization of volume data with quadratic super splines. In *Proceedings of Visualization'03*, pages 393–400. IEEE, 2003.

[14] G.H. Weber, G. Scheuermann, and B. Hamann. Detecting critical regions in scalar fields. In *Proceedings of VisSym'03, Joint Eurographics and IEEE TCVG Symposium on Visualization*, Data Visualisation, pages 85–94 & 288. IEEE, 2003.

[15] J. Weickert. In *Anisotropic diffusion in image processing*, ECMI Series. Teubner Stuttgart, 1998.

[16] W. Welch and A. Witkin. Variational surface modeling. In *Proceedings of Siggraph'92*, Computer Graphics, pages 157–166. ACM, 1992.

[17] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. In *The Visual Computer*, volume 15, pages 100–111, 1999.