The Visual Computer

Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces

R. Westermann, L. Kobbelt, T. Ertl

Computer Graphics Group,
Computer Sciences Department,
University Erlangen-Nürnberg, Am Weichselgarten 9,
91058 Erlangen, Germany
E-mail:
{wester|kobbelt|ertl}@informatik.uni-erlangen.de

We propose an adaptive approach for the fast reconstruction of isosurfaces from regular volume data at arbitrary levels of detail. The algorithm has been designed to enable real-time navigation through complex structures while providing user-adjustable resolution levels. Since adaptive on-the-fly reconstruction and rendering is performed from a hierarchical octree representation of the volume data, the method does not depend on preprocessing with respect to a specific isovalue, thus the user can browse interactively through the set of all possible isosurfaces. Special attention is paid to the fixing of cracks in the surface where the adaptive reconstruction level changes and to the efficient estimation of the isosurface's curvature.

Key words: Volume Rendering – Isosurface reconstruction – Crack fixing – Refinement oracles – Level of detail

Correspondence to: R. Westermann

1 Introduction

The extraction of isosurfaces is a widely used visualization method for scalar-valued volume data sets. It is especially appropriate for volume data containing objects with clearly determined boundaries (like bones in CT), where the lighting and shading of the surface greatly enhances their 3D structure. Furthermore, the generation of polygonal isosurfaces seems to be the preferred visualization technique for workstations with 3D graphics accelerators.

The standard marching cubes algorithm [11] traverses all cells of the volume and determines the triangulation within each cell by means of trilinear interpolation of the values at the cell vertices. Special treatment of ambiguities is required to avoid inconsistencies visible as holes [13]. While this method leads to satisfactory results for small to medium sized data sets, it turns out that it is not appropriate for data set sizes typically found in medical applications for example. Here, the surface extraction with sequential standard algorithms takes on the order of minutes and generates up to a million triangles or more, both of which severely restrict interactive manipulation.

Since volumetric data sets are intrinsically huge, much effort has been made during the last years to come up with optimized visualization algorithms. The goal is to develop algorithms that react to changes of mapping parameters (e.g., varying the isovalue) by almost immediately regenerating the corresponding geometrical representation, which then ought be rendered with several frames per second. Only with this type of real-time interaction and navigation is it possible to effectively analyze an unknown data set and to compensate for the information lost during the projection of the 3D scene onto the screen. Various methods to deal with these problems include discretized algorithms [12], improved sorting, and incremental update techniques [1, 17], efficient cell search with interval data structures [10], and polygon reduction [6, 7, 15].

However, despite all the sophistication incorporated into these methods, it seems that the data sets are growing faster than algorithmic progress is made. For example, data volumes from 3D medical imaging like CT are approaching sizes of 512^3, which amounts to more than 100 million voxel cells. It is obvious that visualization methods that essentially have to access each cell of a data set in order to derive a visual mapping might not

The Visual Computer (1999) 15:100–111

catch up to the goal of interactive processing. Thus, we have to reduce the number of cells that have to be visually mapped. This means that we have to adaptively switch to coarser representations of the data whenever this is acceptable within tolerances prescribed by the user.

The first of these hierarchical approaches applied in volume visualization were based on octrees. The basic idea is to recursively combine eight cubed subvolumes to a coarser cell, working from the bottom up from the original data set. By storing additional information at each node, one can detect and skip uninteresting parts of the volume during the traversal. Using such hierarchies for the mapping itself, i.e., for the isosurface extraction, is difficult because neighboring octree leaves at different levels of resolution exhibit hanging nodes that lead to interpolation discontinuities visible as cracks [14, 16].

In this paper we describe a new method that solves the problems associated with adaptive isosurface extraction from octrees and that provides the basis for the real-time exploration of large regular volume data sets. After discussing other octree approaches in Sect. 2, we describe in detail how continuity can be established across hierarchy levels (Sect. 3). View dependency as the basic requirement for real-time performance (Sect. 4) is achieved by two complementary refinement oracles presented in Sect. 5. Implementation issues are treated in Sect. 6 before we discuss our results (Sect. 7), and conclude with ideas for future work.

2 Octree-based isosurface reconstruction

The benefits of octrees for faster reconstruction of isosurfaces from regular volume data were first recognized in [20]. By storing, at each inner node of the tree, the minimum and maximum material value that appears in any of the branches below that node, the search for all relevant cells through which the surface passes can be speeded up considerably.

Nevertheless, although the number of cells that have to be visited is reduced when the hierarchy is recursively traversed, the surface is still reconstructed from the original data. As a consequence, the size of the details captured is determined by the resolution of the original cells. This prevents an adaptive reconstruction with adjustable approximation precision. For high-resolution data sets, however, the complexity of the generated meshes makes interactive surface extraction impossible, since the number of generated triangles can hardly be displayed in real time.

In order to circumvent these drawbacks, algorithms have been designed to enable *adaptive* surface reconstruction from hierarchically decomposed volume data [14, 16]. Usually the hierarchy is traversed in a top-down order and the marching cubes *extraction procedure* [11] is applied to those nodes that meet a certain criterion. Once a node has been selected for extraction, the traversal is pruned to avoid the processing of child nodes below the current one.

To generate an octree hierarchy for a given volume data set, there are different strategies for obtaining the coarser representations. In volume rendering applications [3, 8, 9], average pyramids are commonly used. These are computed by successively applying a low-pass filter to the voxel data starting at the finest level. Every other sample is pushed up to the next level in the hierarchy, thus reducing the resolution in each dimension by a factor of two (*down sampling*). The small memory overhead of 15% to store the lower resolutions can be avoided if coarser levels are generated by merely subsampling the original data [14], since the access to coarser levels can be implemented by index scaling. Wavelet techniques [19] combine in-place storage of the octree hierarchy with low-pass filtering for coarser levels, but require more involved methods to randomly access a specific voxel value.

3 Continuous isosurfaces

Despite the apparent advantages of octree representations that provide increasingly smoother approximations of the data at coarser levels, problems occur if an isosurface is to be reconstructed adaptively from different levels. Since data samples are averaged, the isosurface may shift or it may completely disappear at one of the coarser levels. Cracks and holes will be the consequence, even if the gradient of the volume data is sufficiently smooth.

The reason for these difficulties is that during adaptive traversal of the octree structure, the un-

derlying scalar field is in fact no longer continuous. In [14] an approach is proposed where coarser approximations are obtained by subsampling the original data. To maintain a *continuous* scalar field even if the extraction level changes, the material values at cell faces where a level transition occurs are properly adjusted: Whenever a cell is adjacent to a coarser level cell, the corresponding data values are resampled by interpolating between the voxel values at the coarser level (Fig. 1). A similar approach is proposed in [16], where the intersection points of the cell edges with the surface of interest are computed in advance at the finest level, and each coarser level subsamples among these points in order to maintain the surface continuity. Our experiments have shown that, even for moderately smooth data sets, this strategy leads to unsatisfactory results as soon as the depth of the hierarchy exceeds 2 or 3. After a few subsampling steps, the topology of the extracted isosurface is already destroyed, even though a continuous representation is guaranteed (Fig. 2).

Due to this observation, we chose an average pyramid in our approach, but we did slightly change the treatment of level transitions to meet the continuity requirements. In the pyramid octree with low-pass filtered coarser levels, the continuity at level transitions can be established by letting the coarser cell sample the scalar field from the finer level at the even-indexed voxels (cf. Fig. 1). The odd-indexed voxels on the finer level have to be recomputed by linear interpolation in turn. As in the other approaches, a continuous transition between different levels is achieved, but a much smoother surface is reconstructed at the coarser levels in the hierarchy.

With the combination of average pyramid representation and appropriate resampling of the values at level transitions we guarantee the continuity of the 3D scalar field where the isosurface is to be extracted. As a consequence, the marching cubes procedure computes the same approximate intersection point for all cells adjacent to a common edge (*edge compatibility*).

However, the continuity of the scalar field does not guarantee the continuity of the isosurface if extraction is performed on different levels. This can clearly be seen from the fact that the isocurve on the common face is approximated by a straight line from the coarser side while it is a broken line with several segments on the finer side (cf. Fig. 3).

Several authors have proposed other techniques to solve this *cracking problem*. In finite element analysis, a related problem arises for adaptively refined volume elements. A standard solution there is to perform a conforming split that eliminates T-vertices [2]. In the case of hexahedral elements, the splitting is done by inserting a cube's center and decomposing the cube into six pyramidal elements. This somewhat decouples the necessary fixing operations on each side. According to the pattern of hanging nodes from finer neighboring cells that have an edge in common with the current cell, we further split the pyramids (cf. Fig. 4).

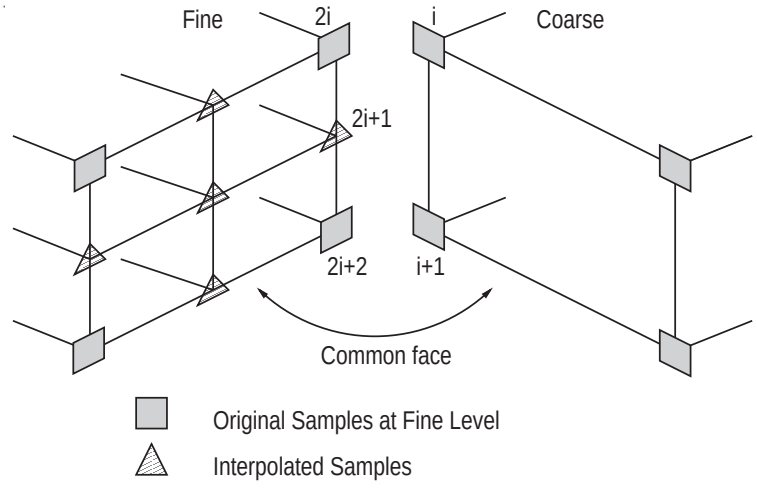
Although the conforming split technique is appealing, it turns out not to be useful for adaptive isosurface extraction. The many tetrahedra and pyramid elements resulting from the split sometimes cause the total number of generated triangles to actually increase, even above the number of triangles that would have been generated for uniform reconstruction on the finest level (cf. Fig. 5).

Another fixing technique has been proposed [16] where the additional intersection points from the finer level are projected onto the coarser level's isoline to geometrically mend the cracks. However, this technique produces T-vertices that can lead to visual artifacts if shading techniques based on normal interpolation are applied.

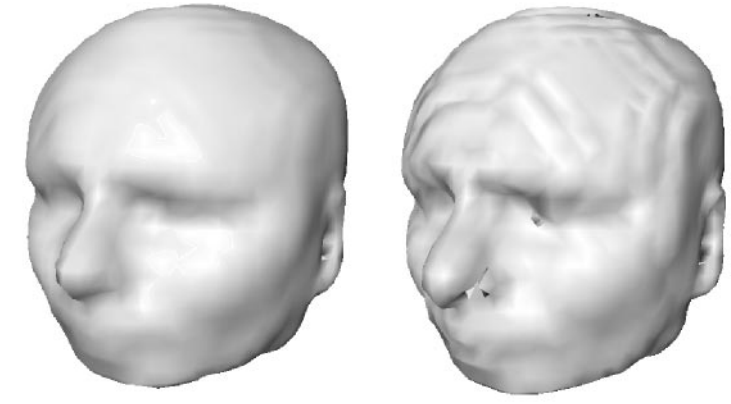
For our approach, we assume that the local refinement oracle guarantees that leaf cells will not differ by more than one generation. In this case the generic constellation to be solved can be depicted as in Fig. 3. A cell from generation $i+1$ is adjacent to four cells from the next finer generation i. In order to minimize the information that has to be gathered from neighboring cells, we decided to leave the marching cubes algorithm unchanged for the finer cells and adapt the extraction in the coarser one in order to close the crack.

Consider an outer boundary edge of a triangle that the marching cubes algorithm generates on the coarser level. Since the corresponding isocurve extracted for the same face, but from the finer side, is a broken line, we split the coarse triangle by inserting its center of gravity and represent it as a *fan of triangles*. Face compatibility is then achieved by simply including the additional intersection points found on the finer level into the sequence of points defining the fan (cf. Fig. 6).

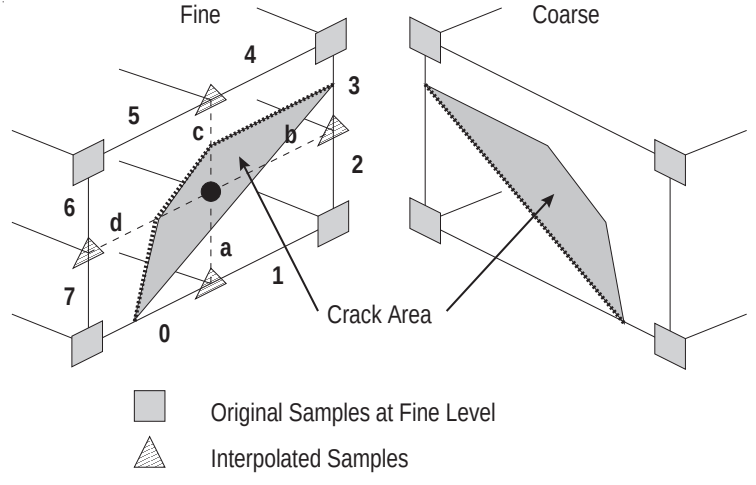
These additional intersection points have to lie on the interior edges emanating from the center vertex



1



2

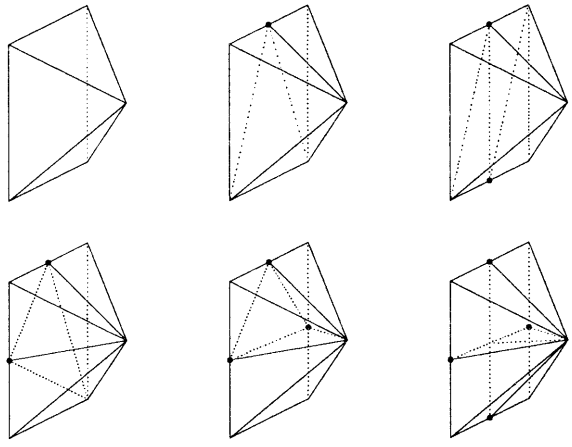


3

Fig. 1. Resampling at level transitions

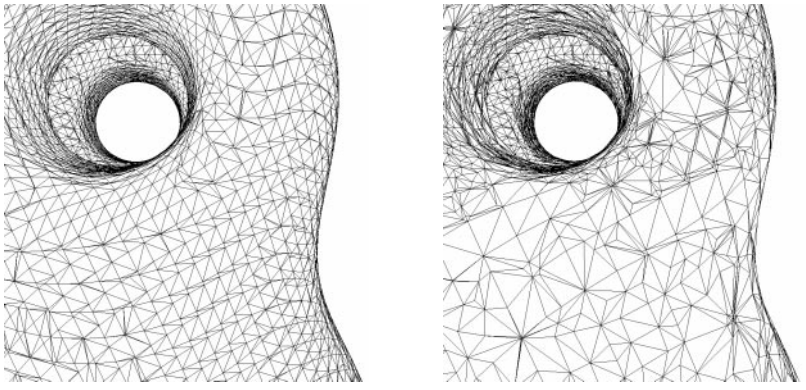
Fig. 2. Isosurface reconstruction from averaged (left) and subsampled (right) data

Fig. 3. Cracks in the piecewise linear approximation to the isosurface occur at common cell faces where cells from different octree levels meet – even if edge compatibility is guaranteed



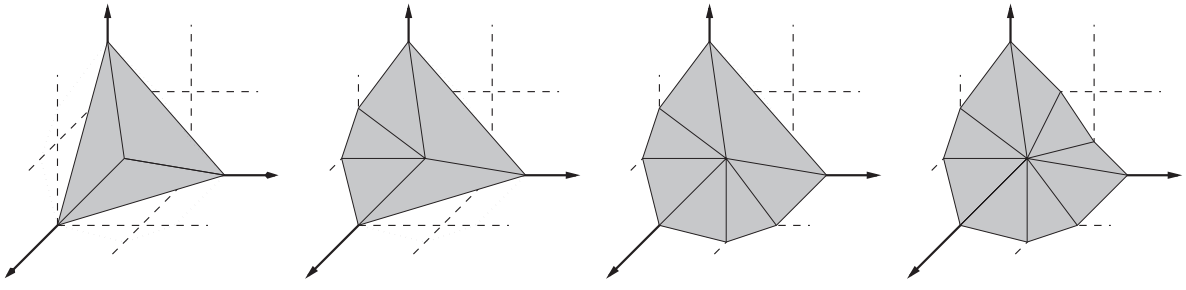
4

Fig. 4. Possible configurations for the conforming split



5

Fig. 5. The triangulation of an isosurface by the uniform marching cubes algorithm on the finest level contains 38608 triangles (left), while the adaptive extraction with conforming splits at the level transitions generates 42343 triangles (right). Although larger triangles are visible, many small triangles occur due to the splitting of some cubes into pyramidal cells



6

Fig. 6. The cracks in the isosurface at level transitions are fixed by replacing coarse triangles by fans of triangles

on the finer level (cf. edges a, b, c, d in Fig. 3). The particular configurations can be indexed by an eight-digit binary number, with each digit being set by the binary predicates indicating whether one of the edges numbered 0 through 7 in Fig. 3 intersects the isosurface or not. This amounts to $2^8 = 256$ different configurations that can be solved off-line and stored in a look-up table. Notice that some cases have ambiguous configurations that have to be resolved by checking the scalar value at the center vertex. In fact, the sign of the center value decides whether the isosurface passes above or below.

4 Real-time exploration of volume data

In the last section, we have seen that hierarchical and adaptive reconstruction of isosurfaces from an octree data structure is the key to coping with the surface's exponential growth of complexity for increasing resolution. However, for real-time applications, the requirements are even harder and the mesh quality apparently has to be traded for CPU cycles. In order to allow the user an effective exploration of large volume data sets in real time, the system has to be able not only to generate different views of the same isosurface at several frames per second, but also to let the user browse through the whole pencil of isosurfaces. The latter feature turns out to be particularly important if the target isovalue has to be found by trial and error or if multiple relevant isosurfaces are present in the data.

Although adaptive isosurface extraction significantly reduces the number of triangles, we still waste graphics performance for rendering unimportant, uninteresting, or even invisible parts of the surface. The standard answer to this observation from the computer graphics point of view is *view dependency*. The decision on which level of the octree a particular region of the isosurface is to be extracted is based not only on intrinsic properties of the surface itself (e.g., curvature), but also on "environmental" aspects like the distance from the viewing camera, the angle to the viewing direction (e.g., back-face culling), or the distance from the center of the view port.

However, since the algorithm for isosurface extraction cannot predict the user's interaction, i.e., predict whether the viewing perspective or the current isovalue will change, we have to run the complete extraction algorithm for each frame. As a consequence, there is no point in caching any geometric information about the current isosurface. We therefore advocate a one-pass scheme for the extraction algorithm. When traversing the octree, a local oracle decides at every node whether the surface can be extracted on this level according to the prescribed error tolerances. If the answer is affirmative, we prune the octree below the current node, compute a local piecewise linear approximation, and send the triangles immediately to the graphics pipeline without further maintaining a global data structure for the isosurface.

5 Refinement oracles

The crucial ingredient for an octree-based adaptive reconstruction algorithm is the oracle, which decides whether the traversal proceeds or the local reconstruction starts. Typically, such oracles are based on some estimate of the local curvature [14] or on an estimate of the potential approximation error caused by the octree not descending further [16]. In our isosurface extraction tool, we pursue a twofold strategy where we combine view-dependent refinement with a local flatness estimator.

During the browsing phase, the user wants to find a specific feature in the data. Hence, the emphasis is on providing real-time visual feedback to interactions like moving/rotating the volume or changing the isovalue. In correspondence to the human visual cognition interface, it is usually sufficient to render the surface in high detail only in a rather small region of interest, while the rest can be displayed rather coarsely.

5.1 Focus point oracle

We implemented this strategy by controlling the local refinement through a *focus point* (center of interest), which serves as a pointing device to indicate the region where the user expects to find important detail. The focus point can be moved freely in space, e.g., by using a space mouse. The size of the region of interest can be modified by adjusting the *radius of interest*.

The oracle now simply computes the euclidean distance of a cell's center from the focus point and evaluates a profile function, which determines down to which level the cells have to be refined. Since we want to keep the crack fixing as simple as possible, we have to construct a radial function that guarantees that the oracle does not allow neighboring cells to differ by more than one generation.

Consider the steepest legal level transition, which is a cell from level 0 (the cell in which the focus point lies) neighboring a cell from level 1, neighboring a cell from level 2 and so on. The maximum

distance from the focus point that the center of the kth cell in this sequence can have is

$$D_k = \left(1 + 3\sum_{i=0}^{k-1} 2^i\right)\frac{d}{2} = \left(1 + 3\left(2^k - 1\right)\right)\frac{d}{2},$$

with $d = \sqrt{3}$ being the diagonal of a level 0 cell. Since we want to bound the coarsening when moving away from the focus point, we have to base the oracle on a monotonic function that maps the D_k to k. The function

$$f : s \mapsto 1 + \log_2\left(\frac{s+d}{3\,d}\right) \tag{1}$$

satisfies this requirement. The radius of interest r is introduced by simply evaluating $f(s-r)$ instead of $f(s)$ and clamping the argument of the logarithm appropriately. For efficiency, the function of Eq. 1 can be precomputed and stored in a table.

5.2 Curvature-dependent oracle

When the radius of interest is set to a rather large value, we end up extracting a considerable portion of the isosurface on the finest level. To have more control over the complexity of the generated surface, we introduce an additional curvature-dependent oracle, which is applied to all cells in the *interior* of the sphere of interest (sr). Obviously, this oracle has to be defined only for the finest-but-one level.

We want to construct an easy-to-evaluate function $K_F(v)$ that gives an estimate for the maximum curvature of the isosurface $F(x, y, z) = v$ within the unit cube $[0, 1]^3$. For the sake of simplicity, we restrict the function F to be a trilinear interpolant.

For a regular point $[x, y, z]$ on the isosurface, we have $\nabla F \neq 0$, and we can (without loss of generality) assume the existence of a function $\phi(x, y)$ such that

$$F(x, y, \phi(x, y)) = v$$

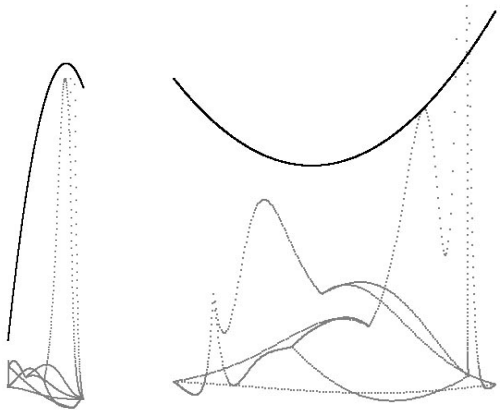
in the vicinity of $[x, y, z]$. It is now straightforward to derive the coefficients of the first and second fundamental forms for the surface $(x, y) \mapsto [x, y, \phi(x, y)]$ and to compute any curvature measure from this. With the standard notation $F_x, F_y, \ldots$ for the partial derivatives of F, we obtain the total curvature

$$\begin{aligned}
&\frac{1}{4}\left(\kappa_1^2 + \kappa_2^2\right) \\
&= \frac{\left(F_x F_y F_{xy} + F_x F_z F_{xz} + F_y F_z F_{yz}\right)^2}{\left(F_x^2 + F_y^2 + F_z^2\right)^3} \\
&\quad + \frac{F_x^2 F_{yz}^2 + F_y^2 F_{xz}^2 + F_z^2 F_{xy}^2}{2\left(F_x^2 + F_y^2 + F_z^2\right)^2} \\
&\quad - \frac{F_x F_y F_{yz} F_{xz} + F_x F_z F_{yz} F_{xy} + F_y F_z F_{xz} F_{xy}}{\left(F_x^2 + F_y^2 + F_z^2\right)^2}
\end{aligned} \tag{2}$$

of the surface $F(x, y, z) = v$, where all partial derivative are evaluated at $[x, y, z]$.

Of course, this functional is rather complicated to evaluate and we have to derive a simpler estimate. Our goal is to get the coefficients a_i of a low-degree polynomial p such that $p(v) \geq K_F(v)$ for all $v_{\min} \leq v \leq v_{\max}$. Such a polynomial is precomputed for each cell and stored. When extracting the isosurface for some value v, the curvature-dependent oracle just has to evaluate this polynomial. In the case of a constant (degree 0) polynomial, we simply estimate the maximum curvature. This kind of error estimate has been used by other authors [5, 14], but taking the total maximum of the curvature for all possible isosurfaces within the range of one octree cell usually overestimates the true curvature significantly. The additional degrees of freedom in higher-order polynomials can be used to find tighter upper bounds. We found that quadratic polynomials yield good results in most cases. Since this requires three coefficient for each cell on the first down-sampled level, we end up with a memory overhead of $\frac{3}{8} = 37.5\%$.

For a random sample point $[x_i, y_i, z_i] \in [0, 1]^3$, we get the scalar value $v_i = F(x_i, y_i, z_i)$ and the corresponding curvature k_i by evaluating Eq. 2. Distributing many samples within the unit cube, we obtain a cloud of points (v_i, k_i) in the $(v \times k)$-plane that characterizes the potential curvature ranges for all possible isosurfaces (cf. Fig. 7). The task is then to find a polynomial $p(v)$ that satisfies $p(v_i) \geq k_i$ for all i as tightly as possible. Since the marching cubes algorithm computes isosurface points on the edges of the finest grid only and linearly interpolates between them anyway, we can also restrict the random samples to these edges



7

Fig. 7. Typical curvature data for isosurfaces from trilinear scalar fields. We sample along the edges of a unit cube. Then we construct a quadratic polynomial $p(v)$ with $p(v_i) \geq k_i$, which has to be evaluated in order to estimate the curvature for a certain iso-value. Notice that some samples are not considered. These are samples that lie close to a singular point where the gradient falls below a prescribed threshold ε

without significantly affecting the curvature estimates.

We apply a simple heuristic for the computation of $p(v)$, which leads to reasonable results. Therefore, we try to find coefficients $b_0, \ldots, b_n$ (Bézier coefficients) in order to represent $p(v)$ on the basis of Bernstein polynomials. This allows us to exploit the convex hull property of Bézier curves [4]. We start by setting all b_j to zero and then do an update for each i with $p(v_i)k_i$. The update should have minimum impact on the b_j in the sense that

$$\sum_j \left(b'_j - b_j\right)^2 \rightarrow \min .$$

Simple least squares fitting shows that this is achieved if

$$b'_j = b_j + \frac{\delta B_j^n(v_i)}{\sum_k B_k^n(v_i)^2},$$

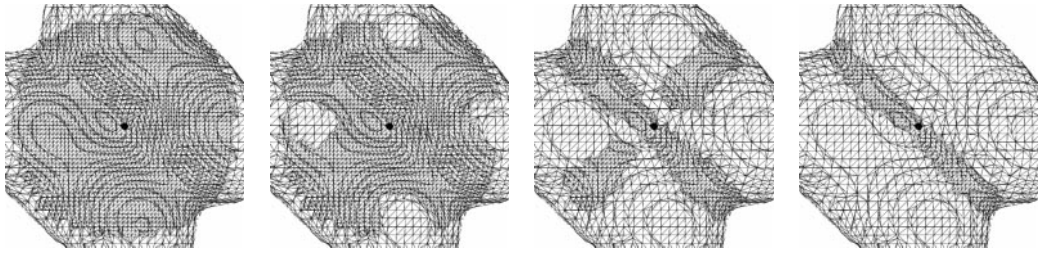
where $B_j^n(v_i)$ are the Bernstein polynomials evaluated at v_i and δ is the positive residuum $k_i - p(v_i)$ *before* the update.

6 Implementing the application interface

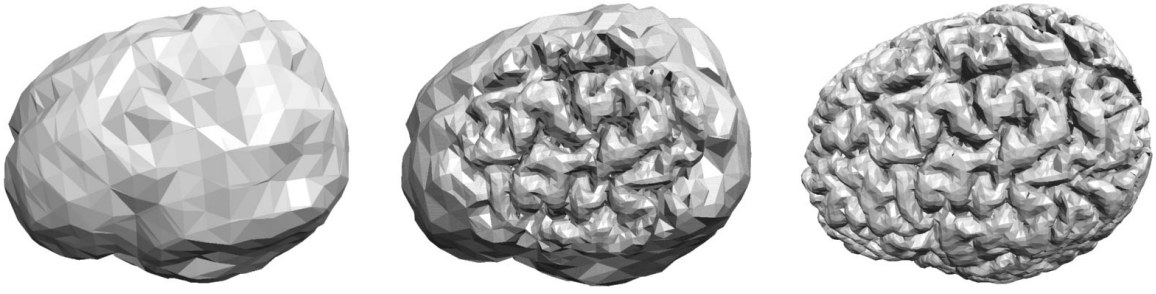
The proposed visualization tool is embedded into the OpenInventor rendering toolkit, thus offering the highest flexibility in terms of user manipulation and navigation.

OpenInventor is an object-oriented graphics toolkit built on top of OpenGL, which has become a de facto standard for interactive modeling, rendering, and manipulation of 3D scenes [18]. For our method to perform efficiently, a new class has been designed that manages the hierarchical volume data representation and provides the core methods to adaptively reconstruct arbitrary isosurfaces. The newly created volume node is a separate object within the hierarchical structure of the scene graph. This allows convenient application of built-in manipulators, sensors, editors, and other predefined classes, methods, and features (light sources, antialiasing, stereo mode, perspective/parallel rendering, fly, walk, trackball). For the real-time exploration of microscopic structures various viewers enabling intuitive navigation through complex environments are of major relevance.

Within the OpenInventor scene graph mechanism, the volume node is organized as a separately managed subgraph. The new `SoVolumeKit` is derived from the class `SoBaseKit`, a container node providing system-defined routines and actions. Clipping planes with a geometric representation are added and can be accessed from the OpenInventor standard manipulators and the core volume element can bei mplemented as a separate shape node derived from SoShape. Even for highly complex structures, additional clipping planes can be used effectively to cut portions of the data, thus removing less important details.



8



9a 9b 9c

Fig. 8. Curvature and focus adaptive refinement. On the left the curvature tolerance is set to zero (46×10^3 triangels). With increasing tolerance more and more cells within the focus area need not be refined. The resulting meshes have 43×10^3 triangels, 32×10^3 triangels, and 25×10^3 triangels

Fig. 9a–c. Examples for the focus point-dependent adaptive reconstruction. Due to the low-pass pyramid, the isosurfaces on the coarser levels are smooth (9a). In the focus area, we can zoom in on the surface down to the finest resolution (9b). The triangle count is 32×10^3 triangels (9a) and 88×10^3 triangels (9c). The brain is extracted from a 256×256×128 CT-scan data set. Extracting the corresponding isosurface by plain marching cubes on the finest level yields 1.6×10^6 triangles (9c)

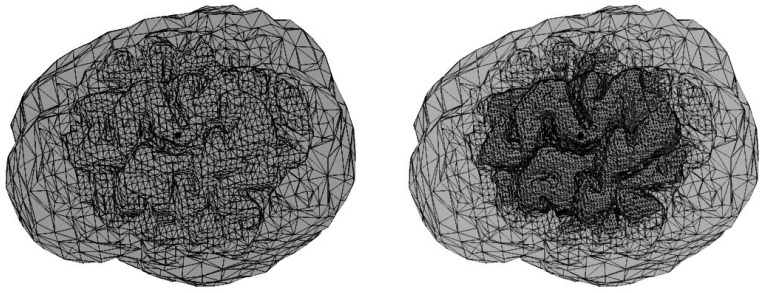
During the rendering phase, an object of type `SoGLRenderAction` traverses the scene graph and asks all objects to render themselves by calling their local `GLRender` method. Within this method of `SoVolume`, all object-specific OpenGL calls that are necessary to prepare the final rendering of the isosurface are performed. Particularly, material properties, the shading mode, culling parameters, and the display mode of the triangles are chosen. All triangles are sent to the geometry engine immediately after they have been reconstructed. In this way we completely avoid the use of intermediate data structures to hold the triangle lists.

The focus point is implemented as a separate node including an displayable object of `SoSphere` and a separate transform node of `SoTransform`. It is linked to the scene graph and connected to the input device that triggers the user navigation. The volume node requests the position of the focus point before the render action takes place and updates the look-up table from which the degree of refinement is derived.

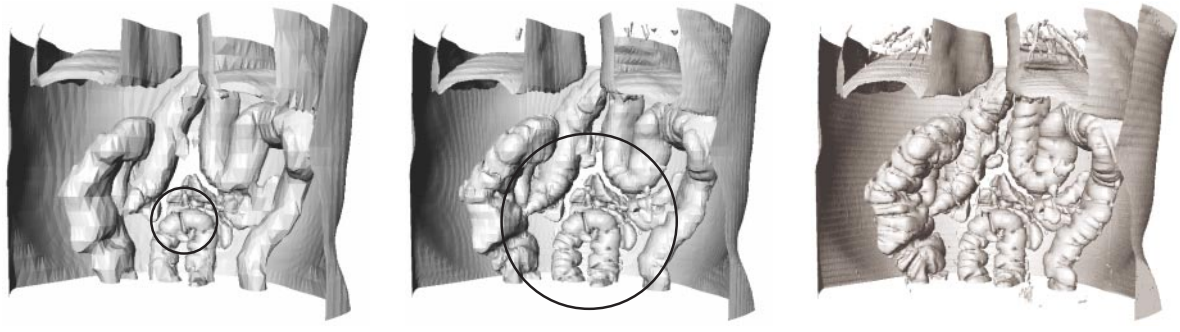
7 Results

Figure 8 shows a mesh representing an isosurface of a synthetic 64^3 volume data set. The focus point is located in the center of the viewport and the focus area is clearly visible. Since we generate a highly detailed mesh only in the vicinity of the focus point, most of the surface can be extracted on a rather coarse level. The averaging filter that was used to compute the voxel values on coarser levels causes the surface to remain smooth. Thus yields an intuitive visual appearance.

If we also take the local curvature of the isosurface into account, then only the nonflat regions are ac-



10



11a 11b 11c

Fig. 10. Wire-frame representations of two adaptively reconstructed isosurfaces

Fig. 11a–c. An example for adaptive and real-time reconstruction of isosurfaces from large-scale volume data (512×512×128). The images show an increasing focus area (black circle) from left to right. Inner structures of the abdomen were reconstructed with 123×10^3 triangels in 1.1 s (**a**). 2.2×10^6triangles were generated in 15.8 s (**b**). The plain marching cubes on the finest level yields 6×10^6triangles in 45.0 s (**c**). Notice that merely rendering a static mesh with 6×10^6triangles (without extraction) on a high-end graphics workstation takes about 5 s

tually subdivided. This allows us to further reduce the number of triangles in the output.

Figure 9 shows an isosurface extracted from a 256×256×128 CT scan. The size of the original data set makes it impossible to effectively explore the raw data on a standard graphics workstation. Adaptive reconstruction allows the user to adjust the complexity of the output to the available hardware resources by enlarging or narrowing the radius of interest. Wire-frame representations of adaptively reconstructed isosurfaces are shown in Fig. 10.

An even larger data set is shown in Fig. 11 (512×512×128). Placing the focus sphere allows the user to explore the details of any inner organ, while the rest of the abdomen is displayed on a rather coarse level. Since we generate the coarser level data by averaging instead of plain subsampling, the global shape of the intestine remains intact. Our fixing technique at level transitions prevents cracks in the isosurfaces.

On a SGI Onyx (R10000, 195 Mhz) with REII graphics hardware our implementation generates isosurfaces with a triangle count of about 50 K triangles at several frames per second. Since we do not exploit any frame-to-frame coherence, the performance is not affected by changes of the isovalue.

8 Conclusion

In this paper we have presented general ideas to exploit a multi-resolution hierarchy of regular volume data for the real-time reconstruction of isosurfaces at an arbitrary level of detail. In order

to account for even the highest resolution data sets, an adaptive strategy has been proposed, enabling the user to focus arbitrarily on any detail of interest. A solution for the crack-fixing problem in adaptive isosurface extraction algorithms has been proposed, and a new error criterion based on the local curvature of the selected isosurface has been introduced, which offers the option of further controlling the approximation precision. Real-time exploration of high-resolution data sets and selection of the desired isosurface is achieved in this way.

Instead of a curvature-based refinement oracle, we can also use a criterion that is based on the actual approximation error caused by not descending to the finest level. For that, however, we have to explicitly compute the surfaces for all possible isovalues in advance in order to estimate the difference between the exact surface and its approximation for each node of the octree. Although the extraction process can be done locally for each node, it is in general too expensive for realistically sized data sets. As a consequence, this strategy has not been considered in our approach.

By integrating these algorithms into the OpenInventor toolkit we exploit the advanced features of modern high-end graphics workstations through standard APIs like OpenGL. The integration of sophisticated user manipulators allows intuitive and easy ways to extract the structures of interest.

Our results have shown that the adaptive and hierarchical nature of our method allows for the processing of even the highest resolution data sets, which can hardly be managed by traditional approaches.

We expect some of our ideas to be of major relevance, especially for applications integrating the internet and benefiting from progressive transmission and rendering. Since we do not build polygon lists explicitly, we could (instead of sending the generated triangles to the geometry processor) establish a communication protocol with a client interface, e.g., a VRML viewer. The surface of interest is then generated on the server side and the primitives are transmitted across the communication channel.

References

1. Bajaj CL, Pascucci V, Schikore DR (1996) Fast isocontouring for improved interactivity. ACM Symposium on Volume Visualization (Proceedings), 39–46
2. Bey J (1995) Tetrahedral grid refinement. Computing 55:355–378
3. Danskin J, Hanrahan P (1992) Fast algorithms for volume ray tracing. ACM Symposium on Volume Visualization (Proceedings), pp 91–98
4. Farin G (1993) Curves and surfaces for Computer aided geometric design: a practical guide, 3rd edn, Academic Press, San Diego
5. Hamann B, Trotts J, Farin G (1997) On approximating contours of the piecewise trilinear interpolant using triangular rational quadratic Bézier patches. IEEE Trans VCG (is Transactions on Visualization and Computer Graphics) Vol 3, No 3
6. Hoppe H (1996) Progressive Meshes. Computer Graphics (SIGGRAPH 96 Conference Proceedings), pp 99–108
7. Kobbelt L, Campagna S, Seidel H-P (1998) A general framework for mesh decimation. Proceedings of the Graphics Interface (Conference Proceedings), pp 43–50
8. Laur D, Hanrahan P (1991) Hierarchical splatting: a progressive refinement algorithm for volume rendering. SIGGRAPH'91 Computer Graphics (SIGGRAPH '96 Conference Proceedings), pp 285–288
9. Levoy M (1990) Efficient ray tracing of volume data. IEEE Trans VCG (is: Transactions on Visualization and Computer Graphics)
10. Livnat Y, Shen H-W, Johnson CR (1996) A near optimal isosurface extraction algorithm using span space. IEEE Trans Visualization Computer Graphics (is: Transactions on Visualization and Computer Graphics)
11. Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3D surface construction algorithm. SIGGRAPH'87 Computer Graphics (SIGGRAPH '87 Conference Proceedings), pp 163–169
12. Montani C, Scateni R, Scopigno R (1994) Discretized marching cubes. In: Bergeron D, Kaufman A (eds) Visualization'94, IEEE Visualization (Conference Proceedings) pp 281–287
13. Nielson G, Hamann B (1991) The asymptodic decider: removing the ambiguity in marching cubes. In: Nielson G, Rosenblum L (eds) Visualization'91, IEEE Visualization (Conference Proceedings) pp 83–91
14. Ohlberger M, Rumpf M (1997) Hierarchical and adaptive visualization on nested grids. Computing 59:269–285
15. Schroeder W, Zarge JA, Lorensen WE (1995) Decimation of triangle meshes. Computer Graphics (SIGGRAPH '92 Conference Proceedings), pp 65–70
16. Shekhar R, Fayyad E, Yagel R, Cornhill J (1996) Octree-based decimation of marching cubes surfaces. Visualization'96, IEEE Visualization (Conference Proceedings) pp 335–342
17. Shen H, Johnson C (1995) Sweeping simplices: a fast isosurface extraction algorithm for unstructgured grids. Visualization'95, IEEE Visualization (Conference Proceedings) pp 143–150
18. Werneke J (1994) The inventor mentor, programming object-oriented 3D graphics with OpenInventor, 2nd edn. Addison-Wesley, Reading, Massachusetts

19. Westermann R (1994) A multiresolution framework for volume rendering. In: Kaufman A, Krüger W (eds) ACM Symposium on Volume Visualization (Proceedings), pp 51–58
20. Wilhelms J, Van Gelder A (1992) Octrees for faster isosurface generation. ACM Trans Graph (is: Transactions on Graphics)

RÜDIGER WESTERMANN is a research fellow in the computer graphics group at the University of Erlangen. His research interests include hierarchical methods in computer graphics, volume rendering of structured and unstructured grids, hardware features, flow visualization and parallel graphics algorithms. He received a Diploma in Computer Science from the Technical University Darmstadt in Germany. He was the first member of the computer graphics group founded by Dr. Wolfgang Krüger at the German Institute for Mathematics and Computer Science in Bonn. Simultaneously he persued his Doctoral thesis on multiresolution techniques in volume rendering, and he received a PhD in computer science from the University of Dortmund in Germany.s

LEIF KOBBELT currently holds a position as a research fellow at the University of Erlangen, Germany. He received his master's (1992) and Ph.D. (1994) degrees from the University of Karlsruhe, Germany. He then spent one year at the University of Wisconsin, Madison as a visiting researcher. Since 1996 he is working in the geometric modeling unit of the Computer Graphics Group at Erlangen. His current major research interest is geometric modeling based on polygonal meshes.

THOMAS ERTL is a professor of computer graphics and visualization in the computer science department of the University of Erlangen where he leads the scientific visualization group. His research interests include volume rendering, flow visualization, multi-resolution analysis, parallel and hardware accelerated graphics, large datasets and interactive steering. He received an MS in computer science from the University of Colorado at Boulder, and a PhD in theoretical astrophysics from the University of Tübingen, Germany.