Real-time Exploration of Regular Volume Data by Adaptive Reconstruction of Iso-Surfaces

R. Westermann, L. Kobbelt, T. Ertl

Computer Graphics Group
University Erlangen-Nürnberg, Germany *

Abstract

Recent advances in the technology of 3D sensors and in the performance of numerical simulations result in the generation of volume data at ever growing size. In order to allow real-time exploration of even the highest resolution data sets, adaptive techniques benefiting from the hierarchical nature of multi-resolution representations have gained special attention. In this paper we propose an adaptive approach to the fast reconstruction of iso-surfaces 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 pre-processing with respect to a specific iso-value thus allowing the user to interactively browse through the set of all possible iso-surfaces. 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 iso-surface's curvature.

1 Introduction

The extraction of iso-surfaces 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 iso-surfaces 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 based on 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, e.g., in medical applications. Here, the surface extraction with sequential standard algorithms takes on the order of minutes and generates up to a million triangles and more which both severely restrict interactive manipulation.

Since volumetric data sets are intrinsically huge, a lot of efforts have been undertaken during the last years to come up with optimized visualization algorithms. The goal is to develop algorithms which react to changes of mapping parameters (e.g. varying the iso-value) 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 it is 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 [17, 1], efficient cell search with interval data structures [10] and polygon reduction [15, 6, 7].

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 which essentially have to access each cell of a data set in order to derive a visual mapping might not catch up to the goal of interactive processing. Thus, we have to reduce the number of cells that have to be visually mapped, which 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 sub-volumes to a coarser cell working 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 iso-surface extraction, is difficult because neighboring octree leaves at different levels of resolution exhibit hanging nodes which lead to interpolation discontinuities visible as cracks [14, 16].

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

2 Octree based iso-surface reconstruction

The benefits of octrees for faster reconstruction of iso-surfaces 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 where the surface passes through can be speeded up considerably.

Nevertheless, although the number of cells which have to be visited is reduced when recursively traversing the hierarchy, 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, thus preventing an adaptive reconstruction with adjustable approximation precision. For high resolution data sets, however, the complexity of the generated meshes makes interactive

* Computer Sciences Department (IMMD9) University of Erlangen-Nürnberg, Am Weichselgarten 9, 91058 Erlangen, Germany, Email: {wester|kobbelt|ertl}@informatik.uni-erlangen.de

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 thereby applying the marching cubes extraction procedure [11] to those nodes which 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.

When generating an octree hierarchy for a given volume data set there are different strategies how to obtain the coarser representations. In volume rendering applications [9, 8, 3] 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 level are generated by merely sub-sampling 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 iso-surfaces

Despite the apparent advantages of octree representations which provide increasingly smoother approximations of the data at coarser levels, problems occur if an iso-surface is to be reconstructed adaptively from different levels. Since data samples are averaged, the iso-surface may shift or it completely disappears 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 when adaptively traversing the octree structure, the underlying scalar field in fact is no longer continuous. In [14] an approach is proposed where coarser approximations are obtained by sub-sampling 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 (see Figure 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 sub-samples among these points in order to maintain the surface continuity.

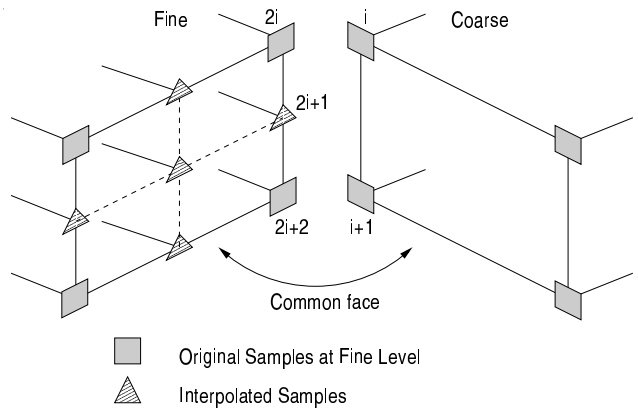

Figure 1: Resampling at level transitions.

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. Already after a few sub-sampling steps the topology of the extracted iso-surface is destroyed even though a continuous representation is guaranteed (see Figure 2).

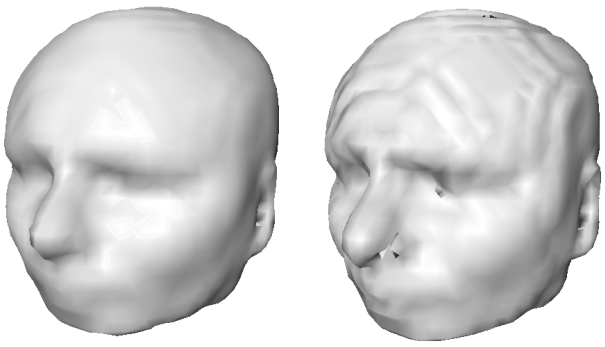

Figure 2: Iso-surface reconstruction from averaged (left) and sub-sampled (right) data.

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

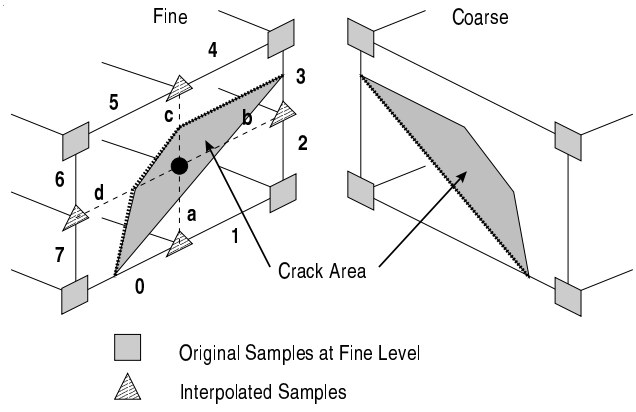

Figure 3: Cracks in the piecewise linear approximation to the iso-surface occur at common cell faces where cells from different octree levels meet — even if edge-compatibility is guaranteed.

However the continuity of the scalar field does not guarantee the continuity of the iso-surface if extraction is performed on different levels. This can be clearly seen from the fact that the iso-curve 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 different 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 which 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 which have an edge in common with the current cell, we further split the pyramids (cf. Fig. 4).

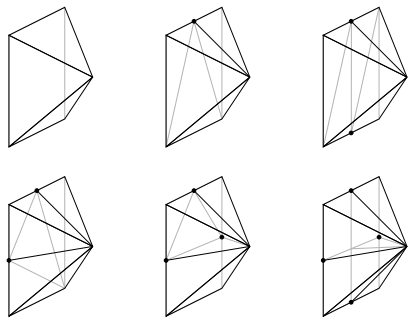

Figure 4: Possible configurations for the conforming split.

Although the conforming split technique is appealing, it turns out not to be useful for adaptive iso-surface extraction since 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 by [16] where the additional intersection points from the finer level are projected onto the coarser level's iso-line to geometrically mend the cracks. However, this technique produces T-vertices which 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 leaf-cells to not differ by more than one generation. In this case the generic constellation to solve 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 inquired 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 which the marching cubes algorithm generates on the coarser level. Since the corresponding iso-curve 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 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 which can be solved off-line and stored in a look-up table. Notice that some cases have ambiguous configurations which have to be resolved by checking the scalar value at the center vertex. In fact, the sign of the center value decides whether the iso-surface passes above or below.

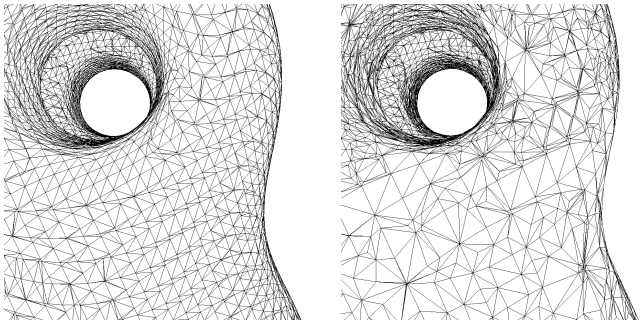

Figure 5: The triangulation of an iso-surface by the uniform marching cubes algorithm on the finest level contains 38608 triangles (left) while the adaptive extraction with conforming splits at the level transitions generates 42343 triangles (right). Although larger triangles are visible, many small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

4 Real-time exploration of volume data

In the last section we saw that hierarchical and adaptive reconstruction of iso-surfaces from an octree data structure is the key to cope 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 to not only generate different views of the same iso-surface with several frames per second but also to let the user browse through the whole pencil of iso-surfaces. The latter feature turns out to be particularly important if the target iso-value has to be found by trial and error or if multiple relevant iso-surfaces are present in the data.

Although adaptive iso-surface extraction significantly reduces the amount 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 iso-surface is to be extracted is not only based on intrinsic properties of the surface itself (e.g. curvature) but also "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 have to be taken into account.

However, since the algorithm for iso-surface extraction cannot predict the user's interaction, i.e. predict whether the viewing perspective or the current iso-value 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 iso-surface. 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 iso-surface.

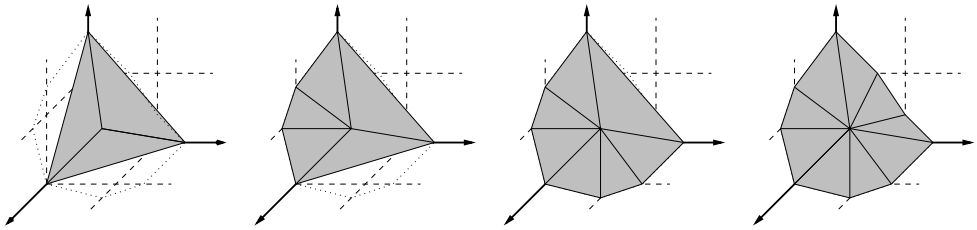

Figure 6: The cracks in the iso-surface at level transitions are fixed by replacing coarse triangles by fans of triangles.

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 not further descending the octree [16]. In our iso-surface extraction tool we pursue a two fold 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 iso-value. In correspondence to the human visual cognition interface it is usually sufficient the 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 which 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 the center of the kth cell in this sequence can have from the focus point is

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

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 which maps the D_k to k. The function

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

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 (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 iso-surface on the finest level. To have more control over the complexity of the generated surface we therefore introduce an additional curvature dependent oracle which is applied to all cells in the *interior* of the sphere of interest ($s < r$). Obviously this oracle has to be defined only for the finest but one level.

We want to construct an easy-to-evaluate function $K_F(v)$ which gives an estimate for the maximum curvature of the iso-surface $F(x,y,z) = v$ 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 iso-surface, 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 straight forward 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

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

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 are 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 iso-surface 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 [14, 5] but taking the total maximum of the curvature for all possible iso-surfaces within the range of one octree cell usually over-estimates 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 (2). Distributing many samples within the unit cube, we obtain a cloud of points (v_i, k_i) in the $(v \times k)$–plane which characterizes the potential curvature ranges for all possible iso-surfaces (cf. Fig 7). The task is then to find a polynomial $p(v)$ which satisfies $p(v_i) \geq k_i$ for all i as tight as possible. Since the marching cubes algorithm computes iso-surface 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 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 $p(v)$ in 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

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

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.

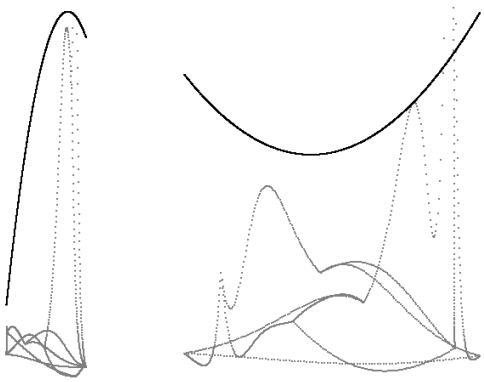

Figure 7: Typical curvature data for iso-surfaces from trilinear scalar fields. We sample along the edges of a unit cube. Then we construct a quadratic polynomial $p(v)$ with $p(v_i) \geq k_i$ which has to be evaluated in order to estimate the curvature for a certain iso-value. Notice that some samples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

6 Implementing the application interface

The proposed visualization tool is embedded into the OpenInventor rendering toolkit thus offering 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 defacto standard for interactive modeling, rendering and manipulation of 3D scenes [18]. For our method to perform efficiently, a new class has been designed which manages the hierarchical volume data representation and provides the core methods to adaptively reconstruct arbitrary iso-surfaces. 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, anti-aliasing, stereo mode, perspective/parallel rendering, fly, walk, trackball). For the real-time exploration of microscopic structures different 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. The core volume element is implemented as a separate shape node derived from `SoShape`. Clipping planes with a geometric representation are added which can be accessed from the OpenInventor standard manipulators. Even for highly complex structures additional clipping planes can be used effectively to cut portions of the data thus removing less important details.

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 are performed which are necessary to prepare the final rendering of the iso-surface. Particularly, material properties, the shading mode, culling parameters, and the display mode of the triangles are chosen. All triangles are send 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 which triggers the user navigation. The volume node requests the position of the focus point before the render action takes place and updates the lookup 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, the majority 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 yielding an intuitive visual appearance.

If we additionally take the local curvature of the iso-surface into account then only the non-flat regions are actually subdivided. This allows us to further reduce the number of triangles in the output.

Figure 9 shows an iso-surface extracted from a $256 \times 256 \times 128$ 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.

An even larger data set is shown in Fig.11 ($512 \times 512 \times 128$). 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 iso-surfaces.

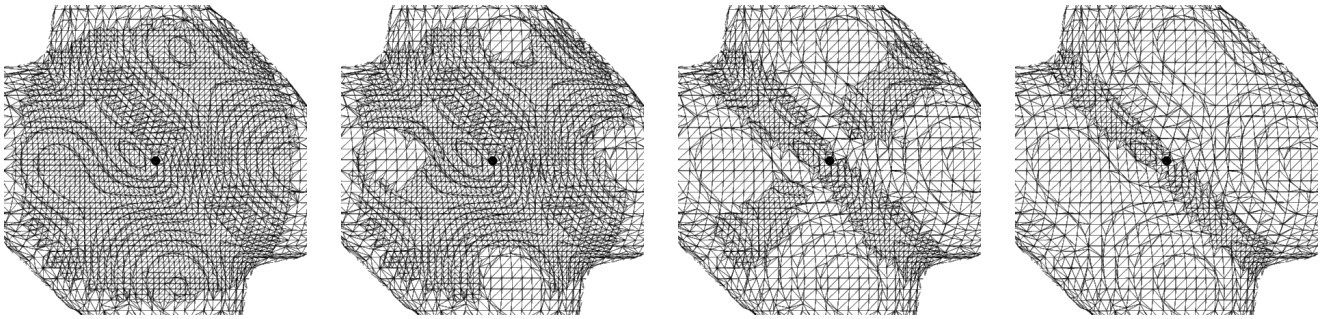

Figure 8: Curvature and focus adaptive refinement. On the left the curvature tolerance is set to zero ($46K\triangle$). With increasing tolerance more and more cells within the focus area need not be refined. The resulting meshes have $43K\triangle$, $32K\triangle$, and $25K\triangle$ respectively.

On a SGI Onyx (R10000,195Mhz) with REII graphics hardware our implementation generates iso-surfaces with a triangle count of about 50K 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 iso-value.

8 Conclusion

In this paper we have presented general ideas to exploit a multiresolution hierarchy of regular volume data for the real-time reconstruction of iso-surfaces at 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 iso-surface extraction algorithms has been proposed and a new error criterion based on the local curvature of the selected iso-surface has been introduced which offers the possibility to further control the approximation precision. Real-time exploration of high resolution data sets and selection of the desired iso-surface 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 iso-values 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 to be performed on 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 allow 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] C.L. Bajaj, V. Pascucci, and D.R. Schikore. Fast Isocontouring for Improved Interactivity. In *ACM Symposium on Volume Visualization*, pages 39–46, 1996.

[2] J. Bey. Tetrahedral Grid Refinement. *Computing*, 55(4):355–378, 1995.

[3] J. Danskin and P. Hanrahan. Fast Algorithms for Volume Ray Tracing. In *1992 Symposium on Volume Visualization*, pages 91–98. ACM SIGGRAPH, 1992.

[4] G. Farin. *Curves and Surfaces.* Academic Press, 1993.

[5] B. Hamann, I. Trotts, and G. Farin. On approximating contours of the piecewise trilinear interpolant using triangular rational quadratic bezier patches. *IEEE Trans. VCG*, pages 215 –227, 1997.

[6] Hugues Hoppe. Progressive Meshes. In *Computer Graphics (SIGGRAPH 96 Proceedings)*, pages 99–108, 1996.

[7] Leif Kobbelt, Swen Campagna, and H-P. Seidel. A general framework for mesh decimation. In *Proceedings of the Graphics Interface conference '98*, pages 43 – 50, 1998.

[8] D. Laur and P. Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. *Computer Graphics (SIGGRAPH '91)*, 25(4):285–288, 1991.

[9] M. Levoy. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.

[10] Y. Livnat, H.-W. Shen, and C.R. Johnson. A Near Optimal Isosurface Extraction Algorithm using Span Space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[11] W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (SIGGRAPH '87)*, 21(4):163–169, 1987.

[12] C. Montani, R. Scateni, and R. Scopigno. Discretized Marching Cubes. In D. Bergeron and A. Kaufman, editors, *Visualization'94*, pages 281–287. IEEE Computer Society Press, 1994.

[13] G. Nielson and B. Hamann. The Asymptodic Decider: Removing the Ambiguity in Marching Cubes. In G. Nielson and Rosenblum. L., editors, *Visualization '91*, pages 83–91. IEEE Computer Society Press, 1991.

[14] M. Ohlberger and M. Rumpf. Hierarchical and Adaptive Visualization on Nested Grids. *Computing*, 59 (4):269–285, 1997.

[15] W. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of Triangle Meshes. *Computer Graphics (SIGGRAPH '92)*, 26(4):65–70, 1992.

[16] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-based decimation of marching cubes surfaces. In *Visualization '96*, pages 335–342, 1996.

[17] H. Shen and C. Johnson. Sweeping Simplices: A Fast Iso-Surface Axtraction Algorithm for Unstructured Grids. In *IEEE Visualization '95*, pages 143–150, 1995.

[18] J. Werneke. *The Inventor Mentor, Programming Object-Oriented 3D Graphics with OpenInventor*. Addison-Wesley, release 2 edition, 1994.

[19] R. Westermann. A Multiresolution Framework for Volume Rendering. In A. Kaufman and W. Krüger, editors, *1994 Symposium on Volume Visualization*, pages 51–58. ACM SIGGRAPH, 1994.

[20] J. Wilhelms and A. Van Geldern. Octrees for faster Iso-Surface Generation. *ACM Transactions on Graphics*, 11(3):201–297, July 1992.

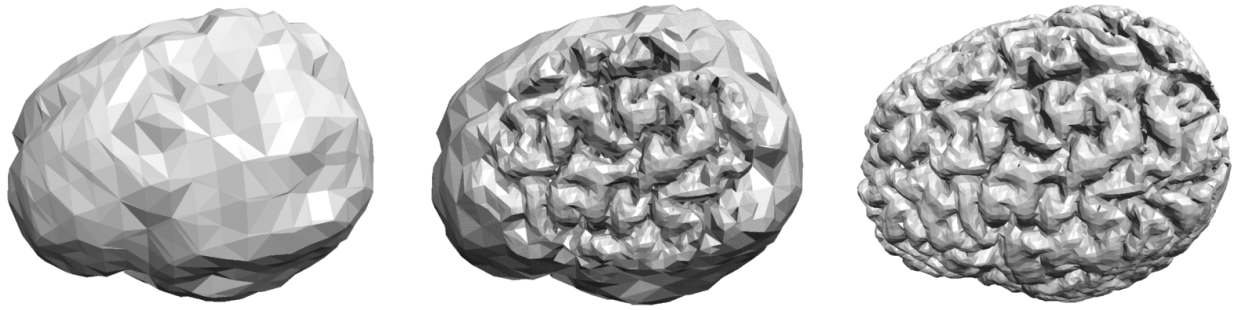

Figure 9: Examples for the focus point dependent adaptive reconstruction. Due to the low-pass pyramid, the iso-surfaces on the coarser levels are smooth (upper left). In the focus area, we can zoom in on the surface down to the finest resolution (upper center). The triangle count is $32K\triangle$ (lower left) and $88K\triangle$ (lower right). The brain is extracted from a $256 \times 256 \times 128$ CT-scan data set. Extracting the corresponding iso-surface by plain marching cubes on the finest level yields $1.6M\triangle$ triangles (upper right).

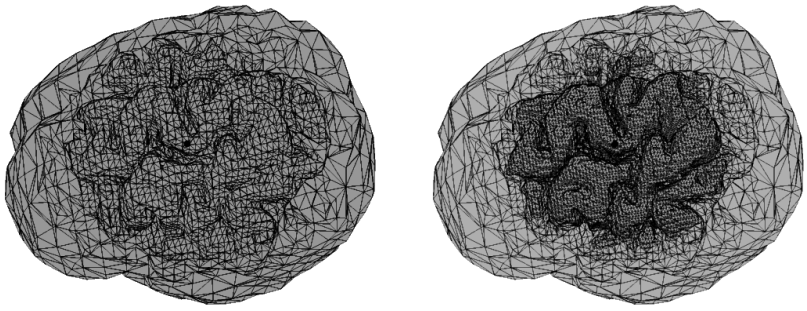

Figure 10: Wire frame representations are illustrated for two adaptively reconstructed iso-surfaces.

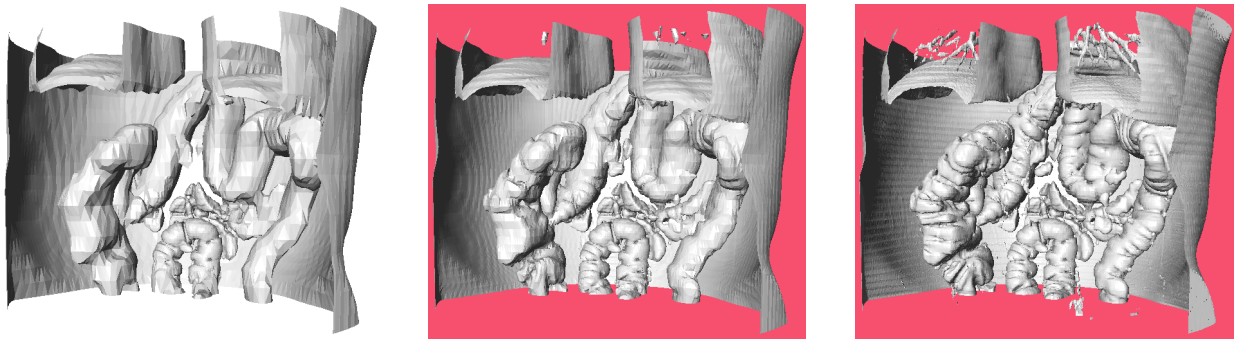

Figure 11: An example for adaptive and real-time reconstruction of iso-surfaces from large-scale volume data ($512 \times 512 \times 128$). The images show an increasing focus area (black circle) from left to right. Inner structures of the abdomen were reconstructed with $123K\triangle$ in 1.1 seconds (left). $2.2M\triangle$ were generated in 15.8 seconds (center). The plain marching cubes on the finest level yields $6M\triangle$ in 45.0 seconds (right). Notice that merely rendering a static mesh with $6M\triangle$ (without extraction) on a high-end graphics workstation takes about 5 seconds!