Adaptive Generation of Surfaces in Volume Data

Heinrich Müller, Michael Stark *

December 1991

Abstract

The generation of polygonal approximations of iso-surfaces or separating surfaces in volume data is a major method in scientific visualization. A widespread approach for this task is the so-called marching cubes algorithm. The marching cubes algorithm, however, has the disadvantage that the number of polygonal chains generated is considerable. The splitting box algorithm presented in this paper reduces the number of polygonal chains by adapting their size to the shape of the surface. The resulting polygonal chains offer a wide spectrum for representing the separating surface. An exact representation is achieved by a new type of generic patches calculated from the polygonal chains. Approximations of different quality may be obtained by combining the algorithm generating the patches with simple triangulations.

1 Introduction

The generation of polygonal approximations of separating surfaces in volume data is a widespread approach of scientific visualization in quite different areas of application. In this paper we assume the volume data to be given by a discrete regular 3D grid of vertices attributed with values sampled from a scalar function over a finite continuous interval in space.

The separating surfaces are obtained by partitioning the vertices in two classes. One approach is to distinguish vertices whose values lie above a given threshold from vertices with values below the threshold. For continuous volume data, i.e. for a continuous scalar function of the interval observed, the separating surface is just the set of points with values equal to the threshold, i.e. an iso-surface. However, in the case of non-continuous volumes like they may occur in medical applications the threshold never needs to be achieved between two separated vertices. For this reason we will prefer the notion of a *separating surface* against the more usual notion of an iso-surface.

One approach for calculating separating surfaces in volume data is the *marching cubes algorithm* [Lorensen, Cline, 1987]. For the following description the separation is assumed to be given by a threshold value. The marching cubes algorithm examines the cubes

*Institut für Informatik, Universität Freiburg, Rheinstr. 10–12, W-7800 Freiburg, Germany, e-mail: [mueller|stark]@informatik.uni-freiburg.de

defined by eight neighboring vertices of the grid. A vertex of a cube is called *black* if its value is greater than or equal to the threshold, and *white* otherwise. The marching cubes algorithm is based on the assumption that the separating surface has only intersections with those edges of a cube which have vertices of different color, and that there is exactly one intersection of the separating surface with an edge of this sort. The intersection point between the edge and the separating surface is estimated by interpolating the values of the vertices, for example linearly. Let us call this point a *contour point*. Then the contour points are connected by oriented line segments which induce polygonal chains separating the black and white vertices of the cube. We denote the line segments as *contour segments* and the polygonal chains as *contour chains*. The contour chains together with an inner triangulation define the separating surface resulting from the marching cubes algorithm.

Since each vertex may be colored either black or white, there are 256 ways of coloring a cube. Using symmetries, [Lorensen, Cline, 1987] reduced the number of really different configurations to 15. With these basic configurations a look-up table can be established yielding the contour chain and its triangulation for each cube of the input grid immediately. The look-up table is accessed with an index calculated from the colors of the vertices of an inspected cube. The marching cubes algorithm scans all cubes of the input grid and reports their triangulated contour chains found in the look-up table.

The 15 basic configurations considered by [Lorensen, Cline, 1987] are incomplete in the sense that they are generating surfaces which occasionally may have holes. By adding some more configurations the continuity of the surface can be guaranteed, cf. eg. [Heiden, Goetze, Brickmann, 1991]. Fig 1 shows a complete set of configurations. Alternatively there is a method by [Wyvill, Mc Pheeters, Wyvill, 1986] for immediately calculating the contour chains without a look-up table.

Another problem of the marching cubes algorithm is that for some configurations there are different natural alternatives of defining the contour chains. Dependent on the choice the resulting surface may show a completely different topology. A comparison of different heuristics of choice is given in [Wilhelms, Van Gelder, 1990b].

Still another problem is that the time of computation spent for empty cubes is considerable. An empty cube is one without surface transition. [Wilhelms, Van Gelder, 1990a] treat this problem with octrees. [Zahlten, Jürgens, 1991] propose a variant named *chain of cubes algorithm* which tracks along the separating surface starting with a cube intersecting the surface.

A problem not treated yet is the large number of contour chains generated for grids of high resolution. In practice, the goal of reducing the number of contour chains is usually achieved by averaging several neighboring vertices of the given grid into vertices of a new, more coarse grid. Since the new grid has less cubes, the number of contour chains will certainly be less. However, the surface thus obtained may show severe differences from the surface generated from the original data. A simple example is shown in Fig. 2. In this example, a separating surface for a threshold value of 0.5 is identified in the original data. In the reduced grid, the surface shown in the original one got lost.

In section 2 of this paper an alternative method called *splitting box algorithm* is presented. The splitting box algorithm adapts the size of the contour chains and hence their number to the shape of the separating surface. The topology of the surface coincides with the marching cubes solution, and it is not more than the sampling distance apart

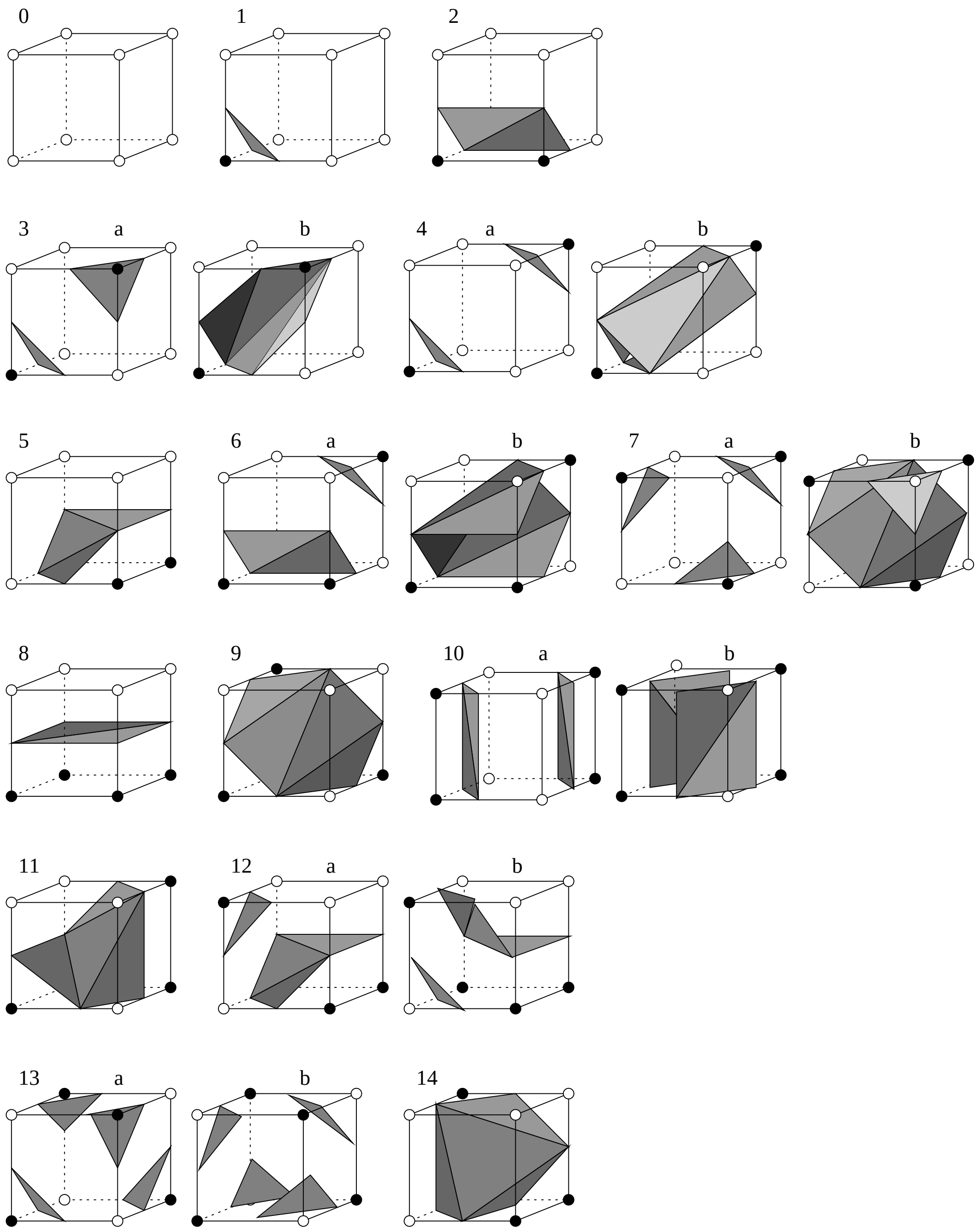

Figure 1: Complete set of marching cubes configurations

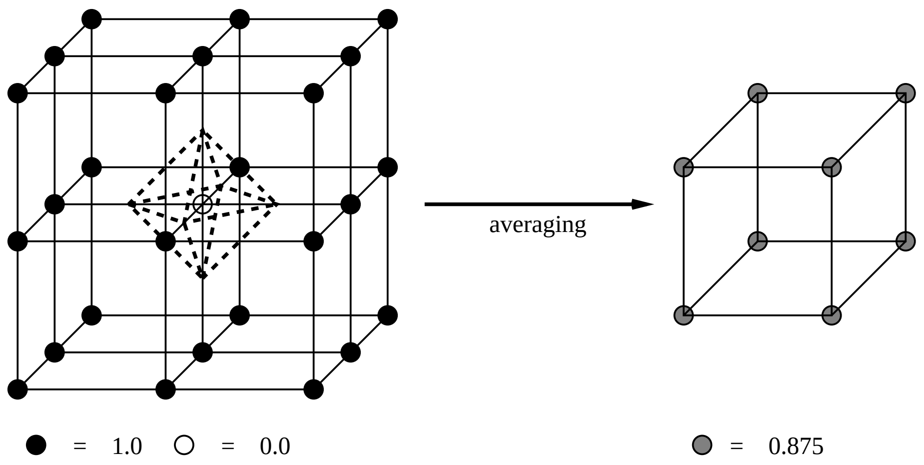

Figure 2: Loss of a separating surface by averaging.

from the marching cubes solution. More precisely, the splitting box algorithm satisfies the following boundary conditions:

1. It preserves the exact separation of black and white vertices. Only the contour point interpolated between two vertices may differ from that of the marching cubes algorithm.

2. It preserves the topology of the marching cubes surface. This means that the surfaces are also closed (except possibly at the boundary of the grid) and that all connected components are preserved.

The output of the splitting box algorithm is a set of contour chains of different size. They offer a wide spectrum for calculating separating surfaces or approximations ranging from the quasi exact reconstruction of the marching cubes surface to more generous triangular approximations with the number of triangles reduced in comparison with the marching cubes algorithm.

The exact reconstruction is achieved by algorithmically defined surface patches induced by the contour chains. The interior of a patch is calculated by iterated linear interpolation from its contour chain. The approximations are defined by a combination of these patches with some approach of triangulation also in use with the conventional marching cubes algorithm. The details are described in section 3.

An estimation of time and space requirements is given in section 4. The time of computation is linear in the size of the input grid, and thus of the same order as for the conventional marching cubes algorithm. Besides the space for the input grid, the algorithm requires additional space, which however is at most logarithmic in the size of the input data. Section 5 contains the results of experiments with the conventional marching cubes algorithm and the splitting box algorithm on data sets of materials science and medicine. The running times confirm the theoretical analysis. For the more smooth medical data set a reduction to approximately 58% of the contour chains in comparison with the conventional marching cubes algorithm was observed. For the more fractal example of a separating surface in a block of aluminium oxide a reduction to 72% was achieved.

2 The splitting box algorithm

The input of the splitting box algorithm is the same as for the conventional marching cubes algorithm. It consists of a finite regular 3D grid of scalar values and a threshold. The grid is assumed to cover geometrically a rectangular parallelepiped. The output is a set of closed polygonal chains, the contour chains, describing a separating surface between the vertices of the grid with values below and above the threshold.

The main geometric objects treated by the splitting box algorithm are rectangular parallelepipeds, called *boxes* for short in the following. The edges of a box are induced by linear sequences of vertices of the grid, cf. Fig. 3. The number of vertices of an edge is called its *length*. Different from the marching cubes algorithm which is restricted to cubes with edge length 2, the splitting box algorithm deals also with boxes having longer edges.

For the explanation of the splitting box algorithm some more terminology is required. An edge of a box is *MC* if it possesses at most one transition of the separating surface. Analogous to the conventional marching cubes algorithm a surface transition is assumed if the color changes at most once along the edge, i.e. there is at most one pair of consecutive vertices with different colors on the edge. A face of a box is denoted as *MC* if its four edges are MC edges. Finally, a box is *MC* if its twelve edges are MC. A simple conclusion is that the six faces of a MC box are all MC faces, and vice versa.

The splitting box algorithm starts with the box given by the input grid. This box is bisected perpendicular to its longest edge into two sub-boxes. If the length of the edge is even, the subdivision is performed so that the lengths of the resulting edges differ by 1 along the subdivided direction (Fig. 3), and is equal sized otherwise. The resulting boxes are recursively bisected in the same manner. The process of bisection ends if a $2 \times 2 \times 2$ box is reached. If there is no unique direction in which the edges are longest, the direction of bisection is chosen according to a fixed order of the possible directions.

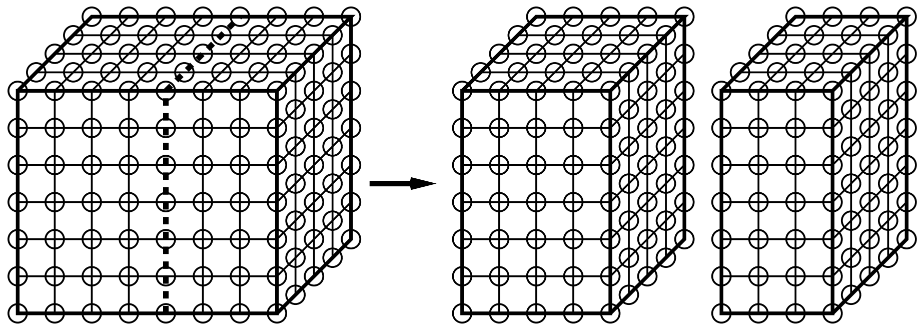

Figure 3: Bisecting a box perpendicular to the longest edge

The boxes arising during the process of bisection are checked whether they are MC. If a box is recognized MC its bisection is postponed. Instead, contour chains are generated for the box, according to the rules of the marching cubes configurations. The contour point on an MC edge is interpolated between the pair of consecutive vertices of different color on the edge, with the same approach as used with the marching cubes algorithm, for example by linear interpolation. Thus the contour point coincides exactly with the one the conventional marching cubes algorithm would generate.

Usually the contour chains obtained in that way will not describe the behavior of a separating surface in the MC box exactly. There might be for example a connected component of the surface in the interior of the box which does not intersect its edges and thus is ignored. Furthermore the contour segments connect consecutive contour points linearly and thus will usually not be a good approximation of the real contour. On the other hand, it may of course happen that the approximation is indeed satisfactory. For these reasons, the bisecting process is continued now, but with the new goal of checking the quality of approximation of the contour chains with respect to the true separating surface in the MC box. A necessary condition for accepting the approximation is that only MC boxes are found in the remainder of the bisecting process. However there are further conditions required which will be explained in the next section. If a non-MC box is found, the bisecting process continues analogous to the first phase, with the goal to find new MC boxes. At last the $2 \times 2 \times 2$ boxes will satisfy the criterion of a satisfactory approximation.

2.1 Checking the quality of approximation

The quality of approximation is checked using a *bisection tree* which is built up implicitly during the bisecting process, cf. Fig. 4. The nodes of the bisection tree contain contour chains. Each contour chain of an inner node has two references onto its parts in the two sub-boxes which define the sons of the node in the tree. Additionally a contour chain owns an attribute `valid` indicating whether the contour chain describes a feasible approximation of a separating surface. The attribute `valid` of a contour chain C is `true` if all contour chains in the subtree following down the references from C have the attribute `valid` equal to `true`. A contour chain is reported by splitting box algorithm if its attribute `valid` is true and if there is no predecessor in the bisection tree with this property.

With the bisection of a box B two cases can be distinguished. They depend on properties of the resulting boxes B_1 and B_2:

1. If one of B_1 and B_2 is no MC box the eventually existing contour chains of B are marked with `valid=false`. If one of B_1 and B_2 is MC, its contour chains are calculated according to the marching cubes configurations and checked recursively. For non-MC boxes the search for MC sub-boxes is continued.

2. If both B_1 and B_2 are MC, contour chains are calculated for both of them according to the marching cubes configurations, based on the edges of B_1 and B_2 respectively. This means that the contour chains already existing for B have to be ignored at first. If there exist contour chains for B they are compared afterwards with the polygonal chains of B_1 and B_2 with respect to similarity of approximation.

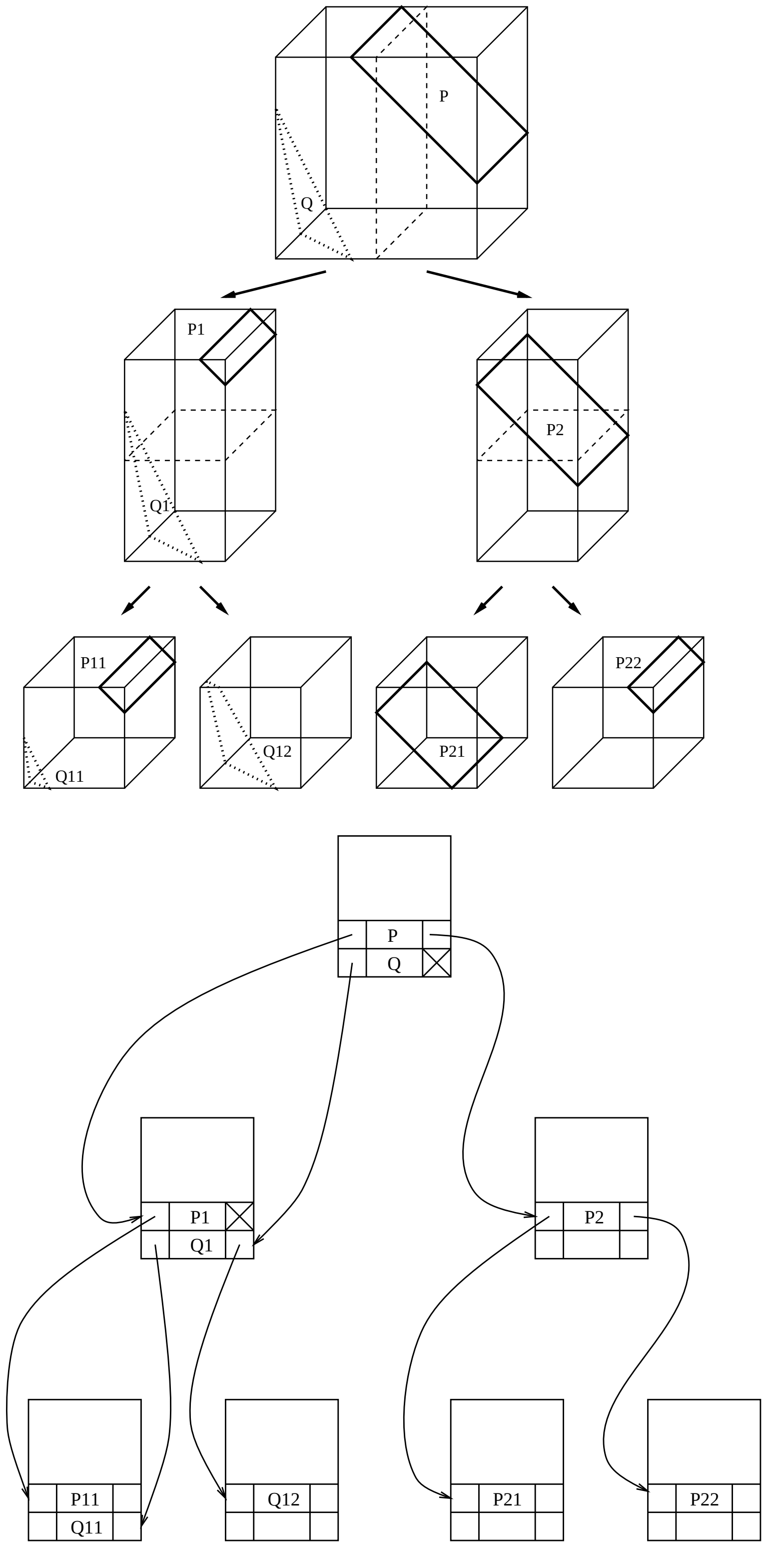

Figure 4: A hierarchy of bisections and the induced bisection tree.

Checking the similarity of approximation for the contour chains in B with respect to B_1 and B_2 required in the second case consists of two steps. First the intersection points between the segments of the contour chains in B and the four new edges defining B_1 and B_2 are considered. If an intersection point lies between a pair of consecutive vertices having different colors it is accepted as a feasible approximation of the contour point on this edge. Fig. 5 shows examples of a feasible and a non-feasible intersection point of a contour chain with a newly generated edge. If an intersection is feasible the coordinates of the respective contour point are replaced by its coordinates, i.e. the respective contour points of B_1 and B_2 are replaced by this intersection point.

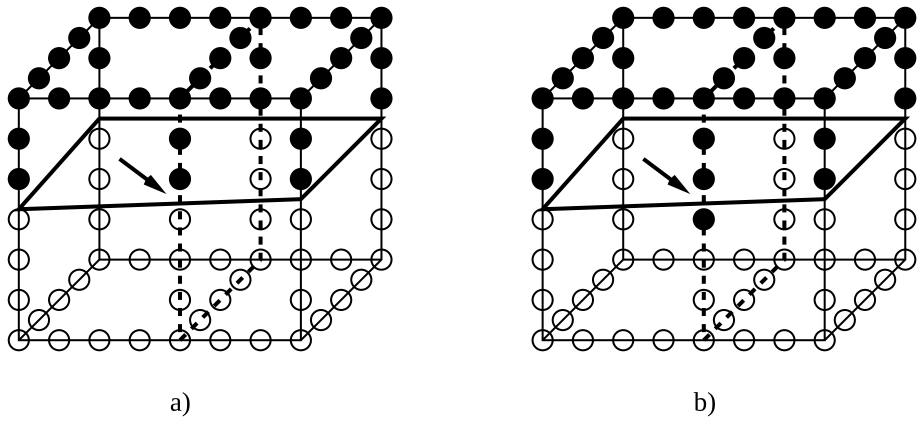

Figure 5: a) a feasible point of intersection, b) a non-feasible point of intersection

The second step is the comparison of the contour chains of B with the contour chains of B_1 and B_2. Fig. 1 shows that for some configurations the separating surface in a box consists of more than one contour chain. The different contour chains are investigated independently by the splitting box algorithm. The reason is that contour chains which have already a satisfactory approximation or which are not influenced by the bisection of the box can be accepted while those of bad approximation will be marked non-feasible and thus will be subdivided further.

Since the contour chains are closed polygonal chains four cases can be distinguished:

1. A contour chain of B lies completely in B_1. Here the contour chain of B must be checked with respect to coincidence with those of B_1.

2. A contour chain of B lies completely in B_2. This is analogous to the previous case.

3. A contour chain of B has exactly two intersection points with the splitting plane between B_1 and B_2. In this case, the contour chain of B is clipped at the splitting plane into its contributions to B_1 and B_2. Both sub-chains are checked for coincidence with the contour chains of B_1 and B_2, respectively.

4. A contour chain of B has four intersection points with the splitting plane, cf. Fig. 6. There are several possibilities to define the sub-chains of the contour chains of B in

B_1 and B_2. Therefore and since the contour chain of B is strongly winded, `valid` is set to `false` in this case.

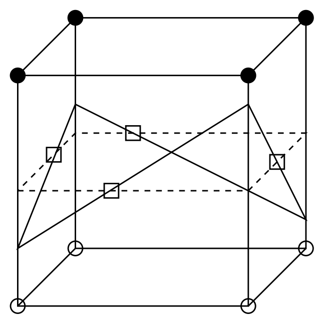

Figure 6: A contour chain with four intersection points on the splitting plane.

The clipping required in the third case is done by successively processing consecutive segments of the contour chain. If a segment has an intersection point with the splitting plane it is calculated and stored. Furthermore, a change between the first and the second sub-chain is registered. If later on the second intersection point is found, the new common segment of the clipped contour chains is calculated from the intersection points. This segment closes the contour chains of the second part. By processing the remaining segments the first sub-chain is completed.

The coincidence of contour chains is tested by checking whether for each sub-chain C_i of a clipped contour chain C of B, there exists a contour chain in B_i which coincides geometrically with C_i. If so, the references from C on its two corresponding components in B_1 and B_2 are set. Otherwise its attribute `valid` is set to `false`.

2.2 The problem of cracks and its solution

A more detailed look on the description of the algorithm shows that contour chains for neighboring boxes may be generated on different levels in the hierarchy of bisection. This may lead to the problem of cracks which is well known for straightforward approaches of adaptive approximation of curved surfaces [Clay, Moreton, 1988]. In the case of the splitting box algorithm this means that the curves induced by the separating surface on a splitting plane may be different for two neighboring boxes. Fig 7a shows two neighboring boxes with a common face. For the box on the right side a feasible contour chain can be generated since it is MC and the black and white vertices are correctly separated. However, the box on the left side has to be bisected further since the left front edge is not MC. Thus the final contour segments are generated in sub-boxes. This causes two cracks along the splitting plane.

A crucial property of the following solution concerning the problem of cracks is that neighboring boxes can be processed independently, without storing additional context information. This favorable property is true for the conventional marching cubes algorithm as well as for the splitting box algorithm described so far. The idea is to process the two

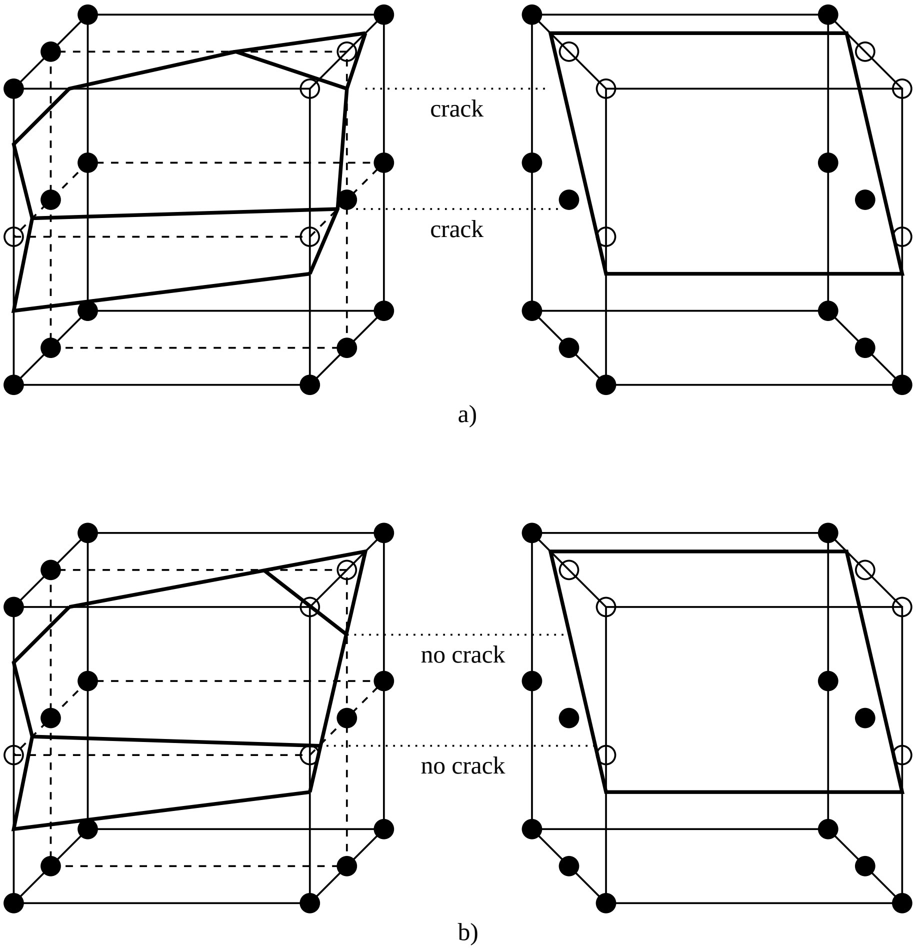

Figure 7: a) Rise of two cracks induced by contour chains generated on different levels. b) Example of a solution.

common faces of neighboring boxes in the same way. This requires a modification of the approach mentioned in the previous section. The difference is that the MC property and the approximation is checked not only for a whole box but also for its six faces. The way of clipping the contour segments on a face depends only on the vertices of the face. It follows that two common faces are treated the same way.

During the bisecting process each newly generated face, i.e. the two common faces of the resulting boxes and faces resulting by bisecting faces of the old box, are processed as follows:

1. If the new face is a MC face, the contour segments are generated on the face according to the configurations of the marching cubes algorithm. This is done in both cases whether the face's box is MC or not.

2. If a face of a box has contour segments intersecting the new edge before bisection, the intersection point is compared with the contour points of the new faces on that edge. If the intersection point lies between the same vertices as the contour point, the latter is replaced by the intersection point in both sub-boxes.

If all faces of a box are MC, the box is also MC. Its contour chains are constructed from the contour segments of the faces. Afterwards the box is processed as described in the previous section.

The first step guarantees that contour segments on common faces of neighboring boxes are no longer generated on different levels in the hierarchy of bisection. In contrary, the contour segments are generated on the same, namely the highest possible level. The geometric location of the contour segments depends only on the grid vertices belonging to the four edges of the face. These vertices however coincide for common faces so that the same contour segments are generated on common faces.

The second step assures that this coincidence is preserved during the continuing process of bisecting. The reason is that common faces are processed independently, but following the same scheme. First this holds for the sequence of bisections which is directed by the longest edges and hence is the same on both faces. Furthermore, if contour segments are subdivided, they are checked on both faces according to the same scheme and either stored or removed on both faces consistently . In particular, a contour point is either replaced by the intersection point on both sides or remains unchanged on both sides. Since afterwards the selected point is inherited to both sub-boxes, all four boxes sharing that edge have the same contour point on the edge.

Fig. 7b displays the correction of the two cracks of Fig. 7a by this approach. Although the left box is not MC, contour segments can be generated on two of its faces. They separate the black and white vertices on both faces and hence are preserved during refinement. The contour points of the contour segments are the intersection points of the two long contour segments with the grid edges. Hence no cracks result on the common face.

A summary of the algorithm can be found in the appendix.

3 Filling the contour chains

The conventional marching cubes algorithm generates a set of non-planar closed polygonal chains as output. The final goal however is a surface. This arises the problem of fitting a surface patch into each polygonal chain. One solution is triangulation. Triangulation is usually done very arbitrary. [Lorensen, Cline, 1987] and [Heiden, Goetze, Brickmann, 1991] assign a chordal triangulation induced by chords of the contour chain. A contour chain with n points is triangulated into $n-2$ triangles. [Wyvill, Mc Pheeters, Wyvill, 1986] calculate the center of the contour chain in order to connect it with the contour points of the chain. A contour chain with n points thus induces n triangles.

The polygonal chains delivered by the splitting box algorithm are non-planar too. Those generated on the lowest level in the bisection hierarchy correspond directly to the polygonal chains generated by the marching cubes algorithm and can be filled with a surface in the same way. For those on a higher level a surface can be deduced immediately from the splitting box algorithm as follows. Starting with the polygonal chain and its surrounding box the surface is constructed by applying the bisection procedure of the splitting box algorithm without the operations for checking the quality of approximation. The box and thus the contour chains are successively bisected as described in section 2.1. The arrangement of the bisections performed is induced by the box according to the longest edge rule. The polygonal chains resulting on the lowest level are triangulated as for the marching cubes algorithm.

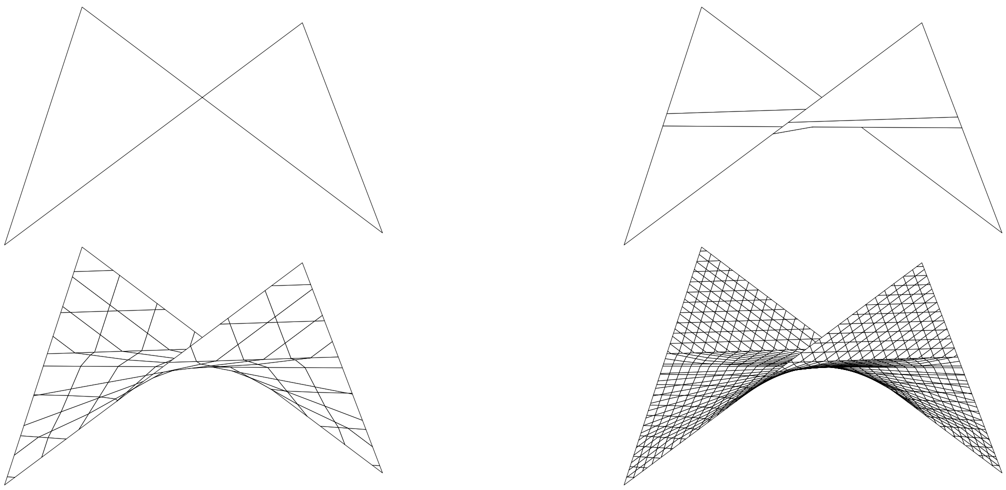

Figure 8: Refinements of a generic patch induced by a nonplanar quadrilateral.

Fig. 8 shows a quadrilateral and refinements with 3, 9, and 15 levels. Surfaces obtained by successive subdivision are sometimes called *generic surfaces*. The analysis of generic surfaces is one direction of research in computer aided geometric design. Further examples of generic surfaces can be found in [Dyn, Levin, Gregory, 1990]. The shape of the generic surface patches of the splitting box algorithm depends on the polygonal chain defining its boundary, and on the surrounding box. Since the points on the surface are calculated by iterated convex combinations of points on the boundary the patch lies in the convex hull of the given contour chain. A particular consequence is that for a planar contour chain

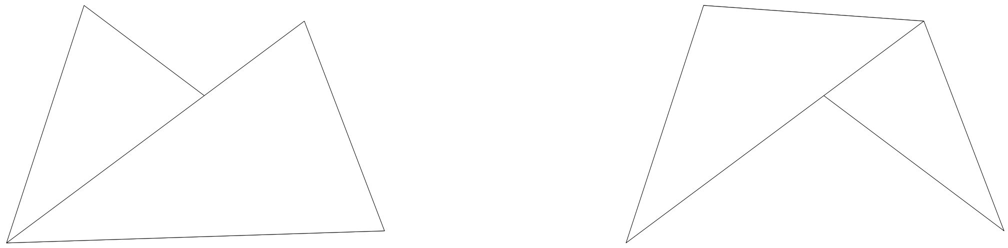

Figure 9: Two chordal triangulations of a nonplanar quadrilateral.

the resulting patch is flat.

For approximation of the separating surface the process of refinement of a generic patch is interrupted on grid level. Normally, the procedure of generation of a generic surface has to be iterated ad infinitum. The set of points this process is usually converging to is the generic surface generated by this process. In contrary it may happen with the splitting box procedure that it can not be continued to infinity. One case of termination is the situation in which the polygonal chain is divided into more than two parts. As noted in section 2.1 the process of partitioning is not continued then. However, according to the calculation of an approximating contour chain by the splitting box algorithm it is guaranteed that its refinement can be continued at least down to the grid level, as required.

Summarizing, the splitting box algorithm generates a separating surface which is composed by generic patches. The generic patches are given by contour chains. They are calculated by applying the reduced version of the splitting box algorithm described above down to a level of bisection determined by the given grid. The contour chains on grid levels are triangulated by one of the methods in use for the conventional marching cubes algorithm. The contour chain as well as the generic patch derived from it separate the black and white points of the grid and thus define a correct separating surface.

Dependent on the application it may not be necessary to perform the refinement down to the grid level in order to get a suitable reconstruction of the separating surface. For the use on a graphic display a refinement only down to the resolution of the display medium is sufficient. An approximate surface is then obtained by triangulating the contour chains on this level.

Another possibility of adaptive control can be achieved by providing a criterion of flatness for the contour chains. The process of refinement is stopped at those polygonal chains satisfying the criterion. Then these chains are triangulated too.

Of course, the surface calculated with one of these heuristics usually will lose the property of separating black and white points. Fig. 9 shows the two possible chordal triangulations of the non-planar quadrilateral of Fig. 8. A comparison with the latter figure shows that both triangulations deviate considerably from the true surface. Also self-penetrations of the surface or penetrations of different connected components might occur. Already the triangulation of a single polygonal chain may cause problems. There are for example polygonal chains for which each chordal triangulation shows these problems.

The problem of penetrating triangulations can be avoided by a slight modification of the splitting box algorithm. A contour chain C not on the elementary cube level is reported only if it is the only one in its box. This happens if the sub-boxes do only contain contour portions related to C. For triangulations with triangles totally in the convex hull, as it is the case for the chordal and the center triangulation, the triangulation is also contained in the surrounding box of the contour chain and thus cannot intersect the triangulation of any other contour chain not in the same box.

The question arises whether there are better triangular approximations of the generic surfaces of the splitting box algorithms than those just described. In its highest refinement, the surface of Fig. 8 shows large triangular sub-areas which could be components of a triangulation in quite natural manner. We leave this question as an open problem.

The discussion shows that the contour chains generated by the splitting box algorithm offer a wide spectrum for calculating surface approximations. It reaches from the exact reconstruction to sometimes also useful triangular approximations with a number of triangles reduced in comparison to the conventional marching cubes algorithm.

4 Time and space requirements

In order to simplify the analysis of the time and space requirements of the splitting box algorithm we assume the initial box to be a cube with edge length $n = 2^m + 1$. This means that an edge of the cube has $2^m + 1$ grid vertices. By bisection such an edge is split into two edges of length $2^{m-1} + 1$. Bounds of the same order as those derived here also hold for boxes of arbitrary shape.

The worst case time requirements of the splitting box algorithm are dominated by testing the MC property for edges of the boxes arising during the process of bisection. The test is performed by sequentially inspecting the grid vertices along the edge. Let $T_{init}(n)$ be the time required for testing the MC property of the initial cube. Since it has twelve edges of length $2^m + 1$ we have

$$T_{init}(n) = T_{init}(2^m + 1) = 12 \cdot (2^m + 1) = 12 \cdot n.$$

Applying three consecutive steps of bisection on a cube of edge length $2^k + 1$ the result is eight sub-cubes of edge length $2^{k-1} + 1$. In the first of these three steps, eight new edges of length $2^{k-1} + 1$ generated by splitting four old edges and four completely new edges of length $2^k + 1$ must be tested. In the second step, for each of the two sub-boxes determined in the first step, eight new edges of length $2^{k-1} + 1$ generated by splitting, two completely new edges of length $2^k + 1$, and two completely new edges of length $2^{k-1} + 1$ have to be processed. In the third step, for each of the four sub-boxes of the previous step, eight new edges of length $2^{k-1} + 1$ generated by splitting four older edges and four completely new edges of length $2^{k-1} + 1$ are tested. Thus we have

$$\begin{aligned} T_{split}(2^k + 1) &= 1 \cdot (8 \cdot (2^{k-1} + 1) + 4 \cdot (2^k + 1)) \\ &+ 2 \cdot (8 \cdot (2^{k-1} + 1) + 2 \cdot (2^k + 1) + 2 \cdot (2^{k-1} + 1)) \\ &+ 4 \cdot (8 \cdot (2^{k-1} + 1) + 4 \cdot (2^{k-1} + 1)) \\ &= 46 \cdot (2^k + 1) + 38. \end{aligned}$$

Hence the total time required by the splitting box algorithm for testing the MC property is

$$\begin{aligned} T(n) = T(2^m+1) &= T_{init}(2^m+1) + \sum_{k=1}^{m} 8^{m-k} \cdot T_{split}(2^k+1) \\ &= 12 \cdot (2^m+1) + \sum_{k=1}^{m} 8^{m-k} \cdot (46 \cdot (2^k+1) + 38) \\ &< 12 \cdot (2^m+1) + 28 \cdot 8^m \\ &< 28 \cdot (2^m+1)^3 \\ &= 28 \cdot n^3 \end{aligned}$$

Since the time necessary for the other operations of the splitting box algorithm is constant for each box, the time of computation is asymptotically of the same order as the time for testing the MC property, i.e. $O(n^3)$. Since n^3 is the number of vertices of the input cube, the running time is linear in the size of the input data. Thus the asymptotic time requirement of the splitting box algorithm is the same as of the conventional marching cubes algorithm.

The space required by the splitting box algorithm is composed of the space for the input data and the working space for the process of bisection. If the process of bisection is performed recursively in a depth-first manner, the working space required on each level is constant. The reason is that on each level only data structures for two boxes do exist. The data structure for a box requires space for its vertices with maximum and minimum coordinates, twelve boolean values for the MC property of edges, twelve contour points, and twelve contour segments. Each contour segment has two references to its two end points, two references to sub-segments and two boolean values. The boolean values indicate whether the segment is feasible and whether it has a predecessor in the bisection tree.

Since at most one path of the bisection tree is in the working space during the execution of the splitting box algorithm and the bisection tree has $O(\log n)$ levels, the working space required is of order $O(\log n)$.

Let us finally note that the space required for a box is constant even if the modification for solving the problem of cracks is considered. The reason is that no information about the partitioning of neighbored boxes is stored.

5 Experimental results

In order to compare the number of contour chains generated by the marching cubes algorithm and the splitting box algorithm, both algorithms were implemented in C and run on a Sun SLC workstation. The implementation of the marching cubes algorithm is a variant of the algorithm by [Wyvill, Mc Pheeters, Wyvill 1986]. The algorithms were applied on data sets from different areas of application for which the separating surfaces have quite different characteristics. The numbers of resulting contour chains are compiled in Tab. 1. Tab. 2 contains the corresponding times of computation.

The data sets of the first example were obtained by digitizing the function $f(x, y, z) = \sqrt{x^2 + y^2 + z^2}$. By variation of the sampling rate data sets of different sizes were synthesized. The sampled values were quantized to one byte. Evidently, the iso-surfaces of this example are spheres. Hence both algorithms deliver a polygonal approximation of a sphere. The reason for showing this example is to visualize the hierarchical partition of the splitting box algorithm. A hidden-line representation of the approximations is shown in Figs. 10 and 11. Tab. 1 shows that for the sphere a reduction to approximately one third of the number of contour chains generated by the marching cubes algorithm is achieved by the splitting box algorithm.

The second example is a medical data set. It consists of cross sectional images of the human hip region which were obtained by computer tomography. The original data set consists of 14 sections of a resolution of 256×256 at a quantization of 2 bytes per element. However, the data set contains only values between 0 and 935. In order to have data sets of different sizes of the same type for the analysis of running times, smaller data sets were generated by averaging neighbored vertices in the original one. Because of the low number of cross sections the average was taken only within the cross sections. A simple segmentation with one threshold value was performed. The threshold was chosen based on the histogram and describes roughly the boundary between tissue and bones. Figs. 12 and 13 show hidden-line representations of the results obtained with the marching cubes algorithm and the splitting box algorithm. Tab. 1 shows that the reductions lie between 53.8% and 58.1%. The range spanned by the different sizes is small since the shape of the surfaces is only little influenced by averaging. The results are worse than for the sphere since the data sets contain more detailed and less smooth structures.

The third example comes from materials science. The material investigated is Al_2O_3 contained in a block of plastic. The data set was captured by micro computer tomography. It consists of 30 slices of resolution 512×512 at a quantization of 1 byte per element. For the experiments a region of 250×100 was cut out which contains the relevant areas completely. The surface was again determined by a threshold value. The threshold was chosen from the histogram on the boundary between the Al_2O_3 and the surrounding plastic. Figs. 14 and 15 show hidden-line representations of the results. Tab. 1 shows that the savings are less than for the previous examples. The reason is the fractal shape of the separating surface. The reduction increases for the reduced data sets from 28% to 41%. This tendency can be explained with the averaging procedure. Successive averaging corresponds to low pass filtering of the data and causes an effect of smoothing. Hence the separating surfaces also become smoother which is favorable for the the splitting box algorithm.

The absolute values of the processing times in Tab. 2 are less noteworthy since they are dependent on the implementation. They rather should be regarded giving the tendency of running times for different sizes of input. The better times achieved by the splitting box algorithm may be caused by the fact that the same contour point is calculated up to four times, once for each adjacent cube, by our implementation of the marching cubes algorithm. The splitting box algorithm calculates a contour point at most twice and then inherits it to other boxes.

Further investigations carried out on the test data concern the size and torsion of the contour chains generated. The *size of a contour chain* is defined as the size of its surrounding box. The surrounding box is the smallest sub-box of the given grid completely enclosing the contour chain. The size is measured by the number of grid cubes the enclosing box consists of. A grid cube is defined by eight neighbored vertices of the grid. Hence the contour chains generated by the marching cubes algorithm all have size 1.

The *torsion of a contour chain* is defined as the maximal angle between two vertex normals of the contour chain. The normal of vertex V_i of a contour chain is calculated as the cross product of the vectors V_iV_{i-1} and V_iV_{i+1}. Equivalently, the vertex normal at V_i is the normal of the triangle defined by the three consecutive vertices $V_{i-1}V_iV_{i+1}$. Between each pair of these normals, the angle is calculated, and finally the maximum of these angles is taken.

The following results were calculated on the three data sets of highest resolution, respectively. Fig. 16 shows the relative frequencies of the sizes of contour chains. Since the splitting box algorithm calculates only boxes of special sizes the histogram differs from 0 only at particular values. The histograms show an evident tendency of decreasing frequency with increasing sizes. This means that the splitting box algorithm substitutes a few small contour chains by a bigger one. Striking with the sphere is that more contour chains of size 2 than of size 1 are generated. It is also evident that the maximum size of contour chains depends on the smoothness of the surface. The maximum size for the sphere is 729, for the medical data set 64, and for the Al_2O_3 data set only 32.

Figs. 17 and 18 show the relative frequencies of torsion values for the contour chains generated by the marching cubes algorithm and by the splitting box algorithm. A clear difference between the sphere and the other data sets can be recognized. For the sphere the maximum torsion is 45° because of its smoothness. Further, the histogram of the sphere shows several distinct local maxima because of the symmetry of the sphere. In contrary, the other data sets show torsion values up to 180°. The histograms are more smooth and the relative frequencies decrease with increasing torsion. Fig. 18 shows the tendency of the splitting box algorithm to generate slightly higher torsion values. This can be noticed particularly in the histogram of the sphere.

Fig. 19 displays the dependency of the sizes of polygonal chains dependent on their torsions. It turns out that the average size of polygonal chains with high torsion is small. As a general tendency it can be noticed that large contour chains are rather planar and that the maximum sizes of contour chains decrease for increasing torsion.

	Marching-Cubes	Splitting-Box	remaining chains
Sphere			
017×017×017	1064	184	17.3%
033×033×033	4304	1312	30.5%
065×065×065	17264	5748	33.3%
129×129×129	69080	22313	32.3%
Tissue/Bone			
032×032×014	3574	1922	53.8%
064×064×014	12815	7125	55.6%
128×128×014	42830	24793	57.9%
256×256×014	142004	82450	58.1%
Al_2O_3			
031×012×030	4203	2368	56.3%
062×025×030	18044	10629	58.9%
125×050×030	64270	42184	65.6%
250×100×030	220276	158603	72.0%

Table 1: Number of generated contour chains

	Marching-Cubes	Splitting-Box
Sphere		
017×017×017	5.7	2.2
033×033×033	25.7	13.9
065×065×065	129.1	95.4
129×129×129	732.9	584.3
Tissue/Bone		
032×032×014	14.6	10.4
064×064×014	53.5	38.1
128×128×014	187.1	138.8
256×256×014	809.4	501.9
Al_2O_3		
031×012×030	16.6	11.5
062×025×030	68.1	61.0
125×050×030	256.3	195.1
250×100×030	1116.4	733.9

Table 2: Processing times in CPU seconds

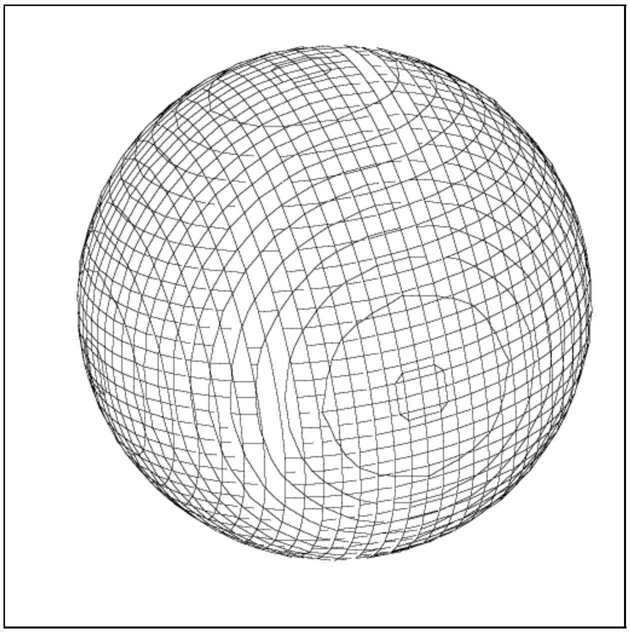

Figure 10: Sphere, 033×033×033 (Marching-Cubes)

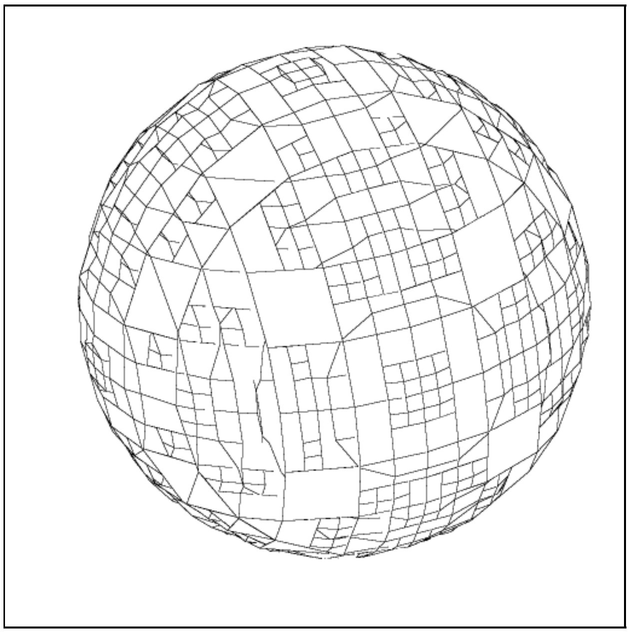

Figure 11: Sphere, 033×033×033 (Splitting-Box)

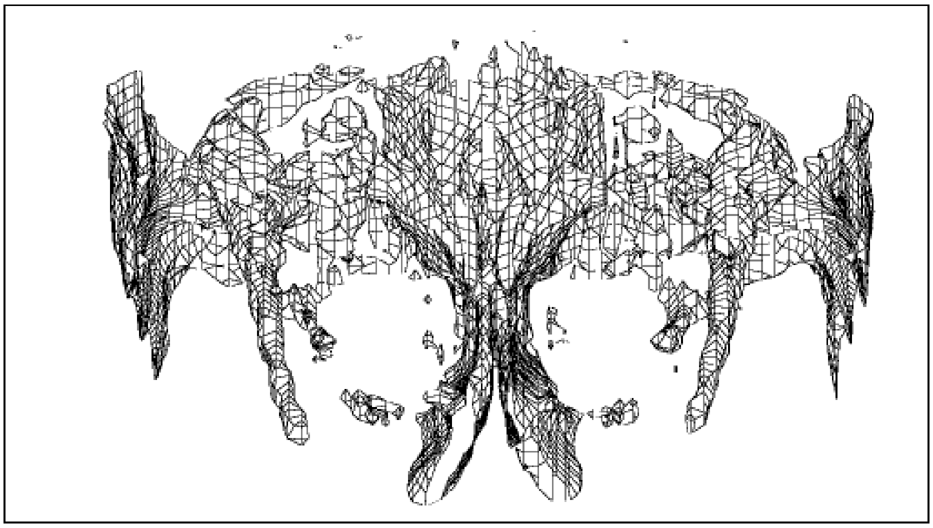

Figure 12: Tissue/Bone, 064×064×014 (Marching-Cubes)

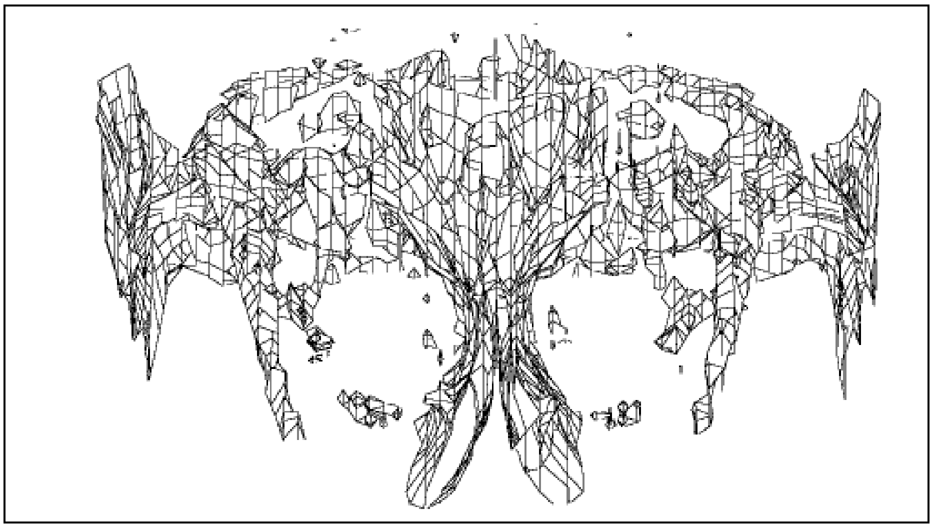

Figure 13: Tissue/Bone, 064×064×014 (Splitting-Box)

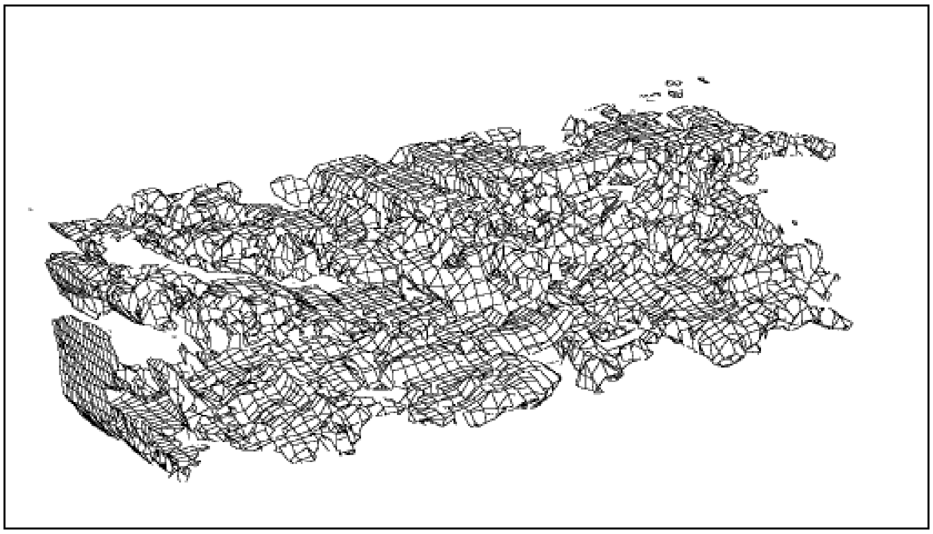

Figure 14: Aluminium Oxide, 062×025×030 (Marching-Cubes)

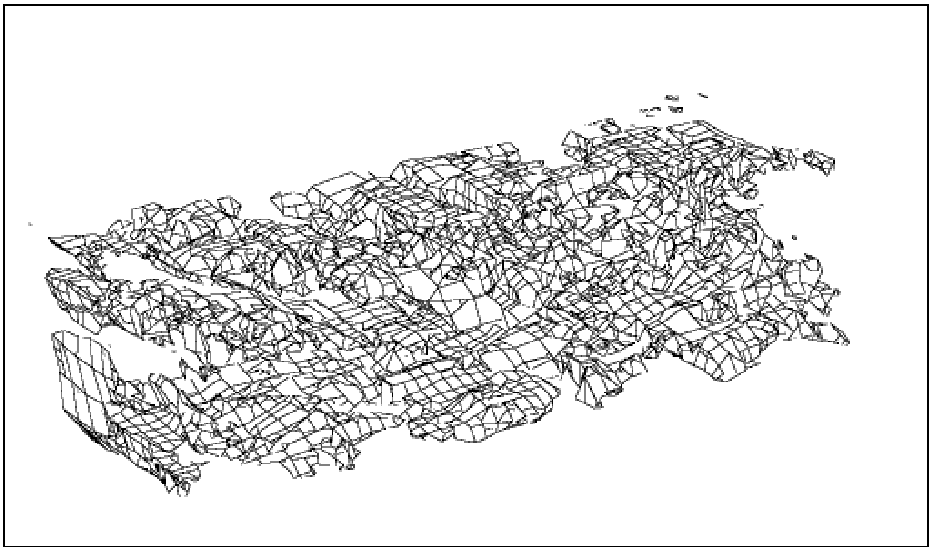

Figure 15: Aluminium Oxide, 062×025×030 (Splitting-Box)

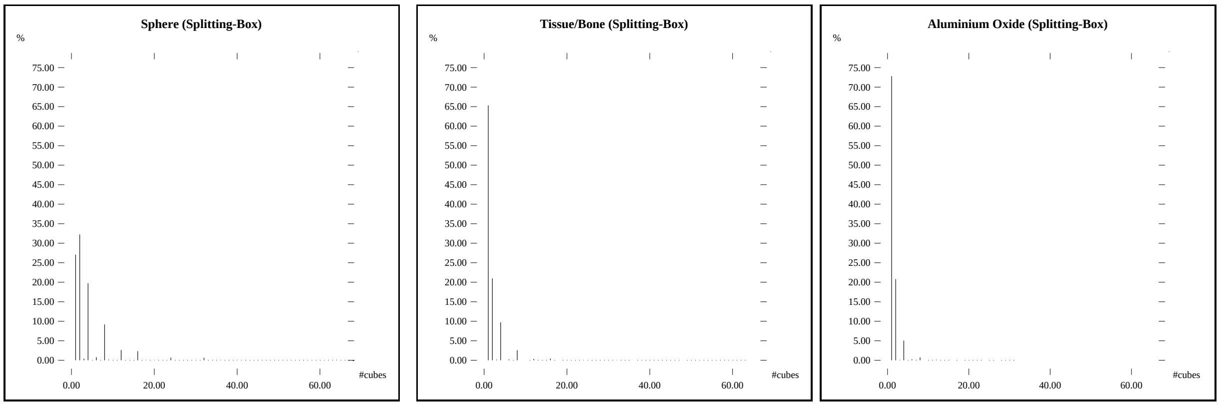

Figure 16: The relative frequency of sizes of contour chains.

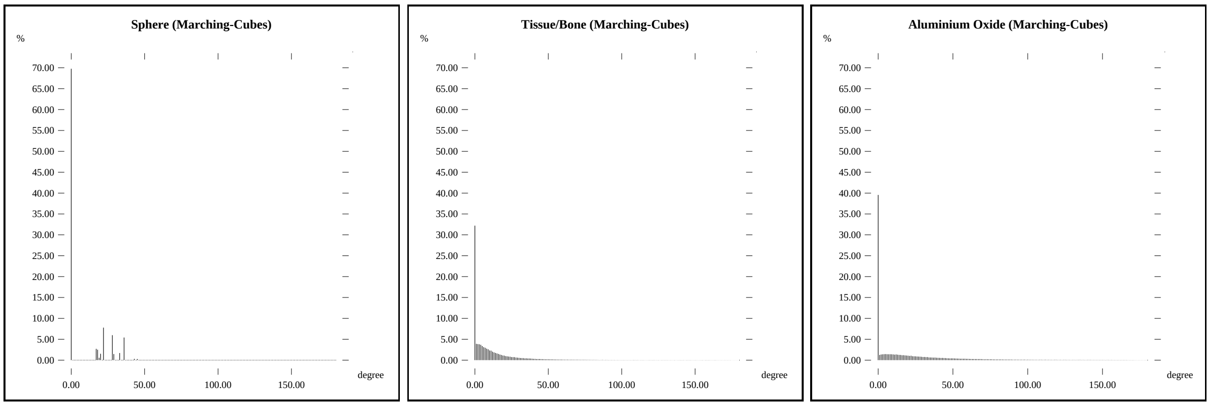

Figure 17: The relative frequency of deviations of normals (marching cubes).

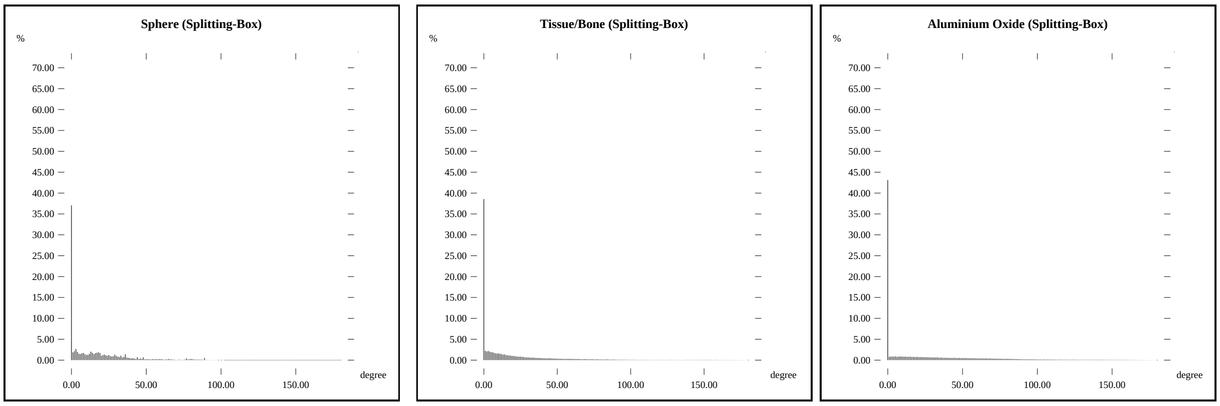

Figure 18: The relative frequency of deviations of normals (splitting box).

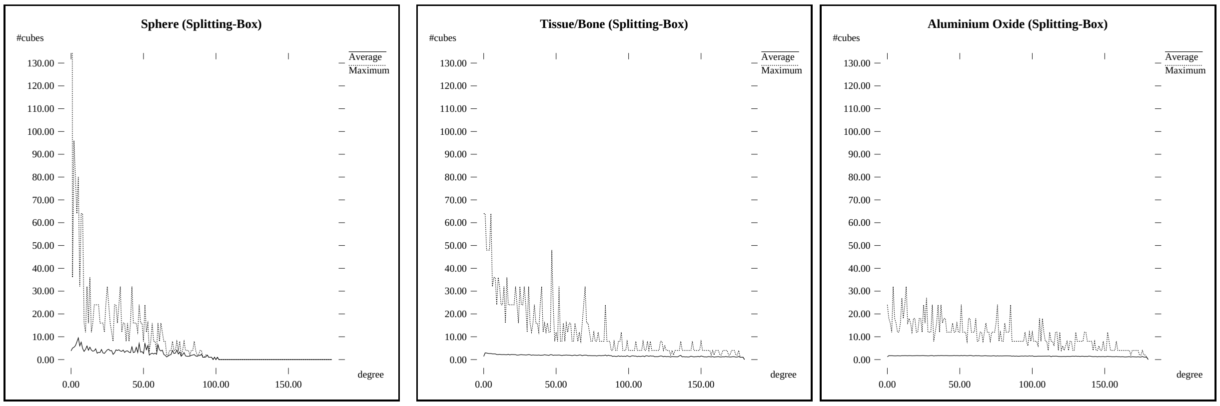

Figure 19: The size of contour chains dependent on the deviation of normals.

6 Discussion

A new hierarchical approach for the adaptive generation of separating surfaces in regularly sampled volume data was presented. The splitting box algorithm reduces the number of contour chains in comparison with the conventional marching cubes algorithm by choosing the size of the contour chains dependent on the shape of the separating surface. The reduction thus achieved is particularly high for smooth surfaces with a small number of connected components. Less favorable are surfaces with a fractal shape or surfaces with a large number of connected components.

The problem of cracks was solved by processing neighbored boxes independently but according to the same strategy on common faces. Since no context information has to be stored only a constant amount of space is required on each level of subdivision. The asymptotic running time corresponds to that of the marching cubes algorithm, i.e. is linear in the size of the input data.

A problem with the splitting box algorithm might be that volume data are usually stored slice by slice, and row by row in each slice. The splitting box algorithm, however, shows a rather scattered access behavior on these data. This may cause troubles if the input data does not fit into main memory. A solution for big data sets might be to divide the data into portions of a number of slices fitting into main memory, and applying the algorithm on each portion in separate.

The contour chains generated by the splitting box algorithm define generic patches. Their shape is induced by the bisecting strategy of the splitting box algorithm. The patches separate the black and white vertices and hence constitute indeed a separating surface. In some applications, for example that of graphical representation, the straightforward triangulation may be sufficient. However, the separation of black and white vertices cannot be guaranteed then, and the surface may show penetrations. An open problem is the triangulation of the generic patches minimizing the number of triangles and the deviation from the true surface simultaneously.

References

[1] R.D. Clay and H.P. Moreton. Efficient adaptive subdivision of bézier surfaces. In *Eurographics '88*, pages 357–372. North-Holland, Amsterdam, 1988.

[2] N. Dyn, D. Levin, and J.A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9:160–169, 1990.

[3] W. Heiden, T. Goetze, and J. Brickmann. "Marching Cube" Algorithmen zur schnellen Generierung von Isoflachen auf der Basis dreidimensionaler Datenfelder. In *Visualisierung von Volumendaten*, pages 112–117. Springer, Berlin, 1991. in german.

[4] W.E. Lorensen and H.E: Cline. Marching cubes: a high resolution 3d surface construction algorithm. *Computer Graphics*, 21(3):163–169, 1987.

[5] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface genereation, extended abstract. *Computer Graphics*, 24(5):57–62, 1990.

[6] J. Wilhelms and A. Van Gelder. Topological considerations in isosurface generation, extended abstract. *Computer Graphics*, 24(5):79–86, 1990.

[7] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2:227–234, 1986.

[8] C. Zahlten and H. Jürgens. Continuation methods for approximating isovalued complex surfaces. In *Eurographics '91*, pages 5–19. North-Holland, Amsterdam, 1991.

Appendix. The splitting box algorithm

In this appendix, a compact summary of the splitting box algorithm is given in a Pascal-like notation.

The input of the splitting box algorithm consists of a grid of vertices which has the shape of a box, and a threshold value. Each vertex is labeled with a number. The output is a set of closed polygonal chains describing a separating surface between the vertices of the grid with values above and below the given threshold.

The description of the data structure of a box is reduced to the information necessary for the following presentation. It corresponds to a Pascal record with the following components:

`mc`: a boolean variable having the value `true` if the box has the MC property;

C^*: a list of at most four contour chains.

The data structure of a contour chain corresponds to a record too. Besides a list of contour points it contains the following components:

C_1,C_2: two references to contour chains resulting from splitting the polygonal chain;

`valid`: a boolean variable having the value `true` if the polygonal chain is a feasible approximation of the separating surface in the box;

`pred`: a boolean variable having the value `true` if the polygonal chain was generated by splitting a polygonal chain on the previous level.

The splitting box algorithm uses three sub-algorithms. The boolean function `splitbox` tests whether the box still can be split. If the box is a cube with edge length 2, the box is not split and `splitbox` returns the value `false`. Otherwise `splitbox` splits the box into two sub-boxes, generates and checks contour segments and returns the value `true`. The sub-algorithm `marchcube` generates contour chains from contour segments. The sub-algorithm *splitchain* splits a contour chain into its parts in the two sub-boxes, as described in section 2.1.

PROCEDURE Splitting-Box(B: Box)
BEGIN
  IF splitbox(B, B_1, B_2) **THEN**
    IF not ($B_1.mc$ **and** $B_2.mc$) **THEN**
    BEGIN
      FOR $C \in B.C^*$ **DO**
        $C.valid :=$ **false**;
      IF (B_1.mc) **THEN** marchcube(B_1);
      IF (B_2.mc) **THEN** marchcube(B_2);
      Splitting-Box(B_1);
      Splitting-Box(B_2);
      FOR $C \in B_1.C^*$ **DO**
        IF (C.valid) **THEN** output C;
      FOR $C \in B_2.C^*$ **DO**
        IF (C.valid) **THEN** output C;
    END
    ELSE
    BEGIN
      marchcube(B_1);
      marchcube(B_2);
      FOR $C \in B.C^*$ **DO**
        BEGIN
          C.valid := **false**;
          splitchain(C, C_1, C_2);
          IF ($C_1 \neq$ **nil**) **THEN**
            IF ($C_2 \neq$ **nil**) **THEN**
            BEGIN
              IF ($C_1 = H_1 \in B_1.C^*$ **and** $C_2 = H_2 \in B_2.C^*$) **THEN**
              BEGIN
                C.valid := **true**;
                $C.C_1 := H_1$; $C.C_2 := H_2$;
                H_1.pred := **true**; H_2.pred := **true**;
              END
            END

```
            ELSE
            BEGIN
              IF (C_1 = H_1 ∈ B_1.C*) THEN
              BEGIN
                C.valid := true;
                C.C_1 := H_1;
                H_1.pred := true;
              END
            END
          ELSE
            IF (C_2 ≠ nil) THEN
            BEGIN
              IF (C_2 = H_2 ∈ B_2.C*) THEN
              BEGIN
                C.valid := true;
                C.C_2 := H_2;
                H_2.pred := true;
              END
            END
        END;
      Splitting-Box(B_1);
      Splitting-Box(B_2);
      FOR C ∈ B.C* DO
        IF C.valid THEN
        BEGIN
          IF ((C.C_1 ≠ nil) and not C.C_1.valid) THEN
            BEGIN
              IF ((C.C_2 ≠ nil) and C.C_2.valid) THEN output C.C_2;
              C.valid := false;
            END
          ELSE
            IF ((C.C_2 ≠ nil) and not C.C_2.valid) THEN
            BEGIN
              IF ((C.C_1 ≠ nil) and C.C_1.valid) THEN output C.C_1;
              C.valid := false;
            END
        END;
      FOR C ∈ B_1.C* DO
        IF (C.valid and not C.pred) THEN output C;
      FOR C ∈ B_2.C* DO
        IF (C.valid and not C.pred) THEN output C;
    END
  ELSE
    FOR C ∈ B.C* DO
      C.valid := true;
END.
```

The central operation of the splitting box algorithm, the boolean function `splitbox`, can be sketched as follows.

```
FUNCTION splitbox(B: Box; VAR B1, B2: Box): Boolean
BEGIN
  IF (B is a cube with edge length 2) THEN
    splitbox := false;
  ELSE
  BEGIN
    FOR ALL undivided edges E of B DO
      inherit contour points to B1, B2;
    FOR ALL remaining edges E of B DO
      BEGIN
        split(E, E1, E2);
        IF (E.mc) THEN
          inherit contour point to B1, B2;
        ELSE
        BEGIN
          IF (E1.mc) THEN
            compute contour point;
          IF (E2.mc) THEN
            compute contour point;
        END;
    FOR ALL new edges E of B1,B2 DO
      IF (E.mc) THEN
      BEGIN
        compute contour point CP;
        IF (E intersects a contour segment of B) THEN
        BEGIN
          compute intersection point IP;
          IF (IP is a valid approximation of CP)
            replace CP by IP;
        END
      END;
    FOR ALL faces F of B1,B2 DO
      IF (F.mc) THEN
        compute contour segments;
    splitbox := true;
  END
END.
```