Noname manuscript No.
(will be inserted by the editor)

Adaptive surface extraction from anisotropic volumetric data: contouring on generalized octrees

Ricardo Uribe Lobello, Florence Denis and Florent Dupont

the date of receipt and acceptance should be inserted later

Abstract In this article, we present an algorithm to extract adaptive surfaces from anisotropic volumetric data. For example, this kind of data can be obtained from a set of segmented images, from the sampling of an implicit function or it can be built by using depth images produced by time-of-flight cameras. However, for many applications as geometry modelling, rendering or finite elements, it is better to use an explicit surface representation. This surface must fit to the geometrical and topological features of the object in order to obtain a good approximation and to avoid topological artefacts. Our algorithm is able to extract adaptive surfaces that accurately approximate the geometry of the original object while minimizing aliasing effects. In addition, our solution is suitable to handle the anisotropy of volumetric representations. In comparison with relevant methods in the state of the art, ours offers a good compromise between mesh quality and precision in the geometrical approximation.

Keywords Surface extraction, Multi-resolution models, Volumetric representation, Image processing.

1 Introduction

Volumetric data are already being used as a valuable source of information in multiple domains. It can be obtained from the segmentation of a set of images obtained from a MRI or from 3D rasterization in order to produce a voxel representation of the object. Similarly, the recent developments of time-of-flight camera technology makes it possible to create 3D models from objects by using depth information in a scene.

Volumetric datasets are often characterized by a strong anisotropy caused by the increasing resolution of images (millions of pixels) produced compared with the small quantity of provided images (usually some hundreds). This will produce highly stretched voxels affecting the ability of conventional octree based algorithms to capture high frequency details of the surface.

Voxel data can be represented as a scalar function F so that $\left\{F(x,y,z) \in \mathbb{R} \mid (x,y,z) \in \mathbb{Z}^3\right\}$ defined at the nodes of a regular grid Ω. Let $v_{x,y,z}$ be the voxel centred at the index (x,y,z). A labelled volumetric object is defined as the set of voxels $V = \{\forall v_{x,y,z} \mid F(x,y,z) = l\}$ where l is a value usually established in a segmentation process.

However, for rendering and geometrical modelling, it is more suitable to handle an explicit surface representation. This has increased the development of algorithms to extract a piecewise surface approximation from the discrete surface of V, noted ∂V.

Generating accurate surfaces from voxels is challenging because many datasets can include degenerated configurations that will induce non-manifold surfaces. Moreover, ∂V can contain sharp features that will force to increase arbitrarily the sampling. In these cases, uniform sampling is not efficient because most of the samples are far from the surface. Therefore, we need a method that adaptively samples ∂V and generates an accurate surface.

In order to be used by post processing algorithms as re-meshing or rendering, the extracted surface must be a topological 2-manifold, which means that every edge on the surface has to be shared by at most two polygons while each vertex in the surface must be surrounded by

The authors are at the
LIRIS laboratory of the University of Lyon
Villeurbanne, 69100, France.
Tel.: +33 (0)4.72.43.26.10
Fax: +33 (0)4.72.43.15.36
E-mail: ricardo.uribe-lobello@liris.cnrs.fr,
florence.denis@liris.cnrs.fr, florent.dupont@liris.cnrs.fr

a 1-ring of edges. Additionally, for numerical applications, the quality of individual polygons is an important factor that can strongly affect the performance and convergence of numerical algorithms.
In this article, we present a surface extraction method well adapted to anisotropic volumetric datasets. It produces visually appealing meshes while preserving the quality of most of the triangular elements.

2 Related Work

There is a vast bibliography concerning surface generation methods. Most methods used to extract surfaces from labelled volumetric data are based on the well known Marching Cubes algorithm [8]. Dual Contouring methods have been mainly used with implicit functions but some of them have been partially adapted to volumetric data. We briefly summarize relevant methods hereafter.
The Marching Cubes (MC) algorithm is based on a regular division of the volumetric data in cubical cells that can be processed separately. All possible intersection patterns have been reduced to 14 pre-calculated cases resumed in a look-up table. However, MC extracts very dense surfaces with many bad-shaped triangles.
Several algorithms have made improvements to MC [20, 15,10]. Kobbelt *et al.* [6] replace the regular grid by an octree and only process the cells which intersect the surface. Varadhan *et al.* [18] control the octree subdivision using a topology preserving criterion in order to make every cell MC-compatible and obtain consistent adaptive meshes. Kazhdan *et al.* [5] propose an algorithm to extract closed manifold surfaces with MC from unrestricted octrees. However, all these algorithms apply MC triangularion inside cells and generate a lot of bad quality triangles. In addition, as surface nodes are located on the edges of octree cells, sharp features are lost during reconstruction.

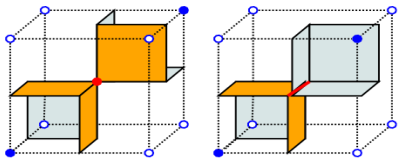


Fig. 1 (Left) Non-manifold vertex configuration and (right) non-manifold edge configuration in original Dual Contouring algorithm.

Dual contouring (DC) methods (see Ju *et al.* [4,13]) are able to build adaptive meshes and to capture sharp features in the presence of Hermite Data (surface normals). In these methods, surface nodes are not created on cell's edges but one node is generated inside every cell. However, this approach can produce topological artefacts and non-manifold configurations as illustrated in figure 2.
By limiting DC octree adaptivity, Varadhan *et al.* [17] have proposed an improved DC that avoids non-manifold configurations. Then, in Dual Marching Cubes (DMC), Nielson [11] used a look-up table to produce surfaces that are dual to those created by Marching cubes by allowing cells to contain more than one dual node in order to solve the non-manifold vertex configurations. However, we observe that by simply applying the DMC algorithm, non-manifold edge configurations can still appear between cells sharing an ambiguous face (see figure 2).

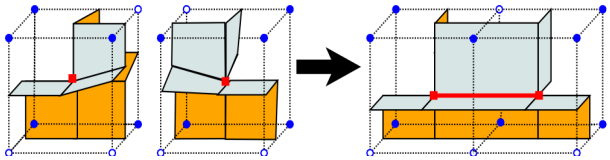


Fig. 2 (Left) Two DMC look-up table cases 17 connected by an ambiguous face. (Right) Non-manifold edge configuration (line in red).

Concerning non-manifold configurations, several approaches propose to create dual grids extracted from an octree data structure. These grids are aligned to features of the surface by using Hermite data on the octree cells. Schaefer *et al.* [14] have proposed to use a dual hexahedral grid combined with marching cubes (Dual Marching Cubes DMC2) to extract the surface. However, as dual cells are not necessarily convex, this algorithm can produce self-intersected surfaces (see figure 3). In addition, aligning the grid to shape features can make many degenerated triangles appear.

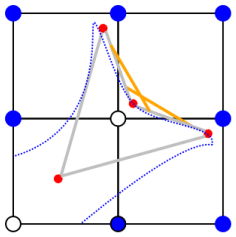


Fig. 3 Schaefer *et al.* DMC2 self intersection problem. Vertices in blue are inside ∂V (white are outside), the surface is the blue dotted line, red circles are dual nodes and grey lines are dual cell edges. Yellow lines are (self-intersected) surface edges that are generated connecting surface nodes located on dual cell edges.

In order to avoid self intersections, Manson and Schaefer [9] built a dual tetrahedral grid that do not contain inverted tetrahedra. Then, they applied a marching

tetrahedra (MT) to extract the surface. This solution produces intersection-free surfaces, unfortunately, using MT, combined with a feature aligned grid produces meshes with many bad-shaped triangles.
Finally, Manifold Dual Contouring (MDC) [12] produces adaptive manifold meshes by using DMC over a regular octree and a topology preserving criterion that maintains the topology of the original object. Nevertheless, this approach can still generate self-intersected surfaces and its bottom-up strategy for mesh simplification is not suitable to handle strongly anisotropic datasets.

In this article, we are mainly interested in adaptive surface extraction from the voxel representation of an object. We want to obtain manifold meshes at several resolution levels without creating too many degenerated triangles. Therefore, we have considered a dual approach as the most suitable option. We present a dual algorithm to produce multi-resolution and adaptive surfaces that is appropriate to volumetric data. It produces meshes with good-shaped triangles and is able to approximate sharp features. The dual approach is inspired by Nielson's Dual Marching Cubes (DMC)[11]. This algorithm allows us to place several surface nodes inside each octree cell and gives us more freedom in nodes localization in order to better approximate the underlying surface ∂V. Our solution extends DMC over a generalized octree data structure to extract surfaces adapted to the geometry of ∂V.
We start by defining some preliminary concepts and the notation that is going to be used. Let be V the volumetric object as defined in the introduction and ∂V its boundary. In this paper, a *cell* is defined as an axis-aligned hexahedral block containing voxels and it is equivalent to the cell of an octree. Hereafter, our octree cells are going to be noted as cells. *Cell volume* will indicate the voxels contained inside a cell and we intend to check only the values of the eight voxels in the corners of every cell. The vertices of every cell are labelled to indicate whether they are inside the object.

Definition 1 *An ambiguous face exists when the face contains two vertices inside ∂V interleaved with two vertices outside ∂V.*

Our algorithm is able to consider at most four surface components inside a single cell. Every component must have a single connected boundary and be well approximated by a plane.

3 Octree data structure

In order to adapt our space subdivision approach to the anisotropy of volumetric data, our octree data structure is not the usual one: the octree cells do not have to be always subdivided in the tree main axial directions. In consequence, we can choose to subdivide a cell in only one or two axial directions as illustrated in the figure 4 for a quadtree.

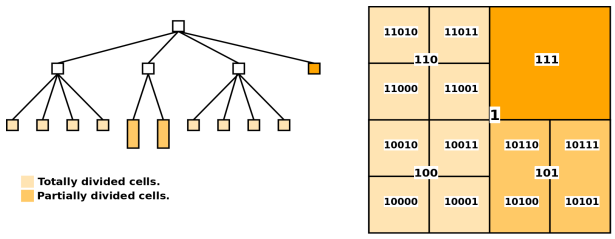


Fig. 4 Generalized quadtree data structure (left) and its representation as Morton codes (right).

For the multi-resolution data structure, we have implemented a hashed octree that uses a Morton space filling code [7] as index. A Morton code can compactly encode hierarchical and spatial information about cells. However, this code have been conceived to work in a regular grid, so, to be usable in a generalized and adaptive data structure, it must be completed with some spatial coordinates information. Our approach is based on the idea that, even if an octree cell c is not divided in all directions, the sub-cells S contained inside it must fulfil, or tile, all the space inside c. This will also mean that all Morton codes indexing sub-cells inside c must be pointing to cells in S. This distribution has to be done with respect to the space occupied for every subcell as illustrated in figure 4 for cell indexed by 101.
This characteristic of our octree construction algorithm combined with the spatial coordinates information contained inside cells allows us to access any neighbouring cell even if the octree has not been divided regularly. Our algorithm is shown in 1.

Algorithm 1: Get Neighbour algorithm

```
input  : Cell c and a face f.
output: A face adjacent cell a through the face f.
level ← GetLevel (c);
coincidentFace ← GetCoincidentFace (f);
offset ← 1;
while offset ≤ level do
    a ← GetAdjacentCellByFace (c,f,offset);
    if Exists (a) then
        cellLimitC ← GetCellLimitByFace (c,f);
        cellLimitA ← GetCellLimitByFace
        (a,coincidentFace);
        if cellLimitC == cellLimitA then
            return adjacentCell;
        else
            offset = offset +1;
        end
    end
end
```

Our function *GetAdjacentCellByFace* returns a cell face adjacent to the provided cell. This is done by obtaining the spatial coordinates of the current cell, adding an offset in the corresponding direction and converting the spatial coordinates into a Morton code. As anisotropic subdivision is usually done at the deepest levels of the octree, our offset will not be larger than the current depth of the octree. If no neighbouring cell exists, our method looks her parent cell by shifting the three least significant bits of its Morton code. The method *GetCellLimitByFace* returns the geometrical limit of a cell in a direction deduced from the face provided. This ensures that two cells are really face adjacent because the irregular subdivision of the octree can produce cells that are close in terms of Morton encoding but far geometrically.
Finally, as our algorithm is strongly dependent on the access to neighbouring cells, we use a hash table to index the octree cells and make almost constant access times.

4 Octree construction

The octree construction uses an iterative top-bottom algorithm that builds an octree based on ∂V topological and geometrical characteristics. First, to capture ∂V general features, we regularly divide the octree until some user defined minimal depth (usually 2 or 3) is reached. Then, we apply several criteria in order to decide if a cell must be subdivided: we check if the octree cell is inhomogeneous, then, we apply a complex cell criteria as explained below.

Definition 2 *Let C be a cell. C is* complex *if at least one of its faces is complex. A cell face is complex if:*

- *It intersects ∂V and all its vertices are either inside or outside V (see figure 5 left).*
- *At least one of its edges intersects ∂V more than once (see figure 5 right).*

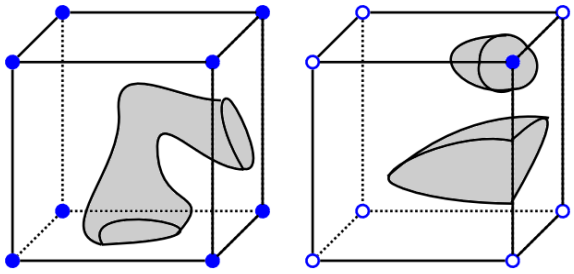


Fig. 5 (Left) a complex cell with a tunnel and (right) a complex cell with a complex edge.

A Complex face criterion allows us to detect cells that contain a piece of the surface that is smaller than the current level of subdivision. In the octree construction context, this criterion is implemented by extracting the connected components of the voxels contained in every homogeneous cell face (all vertices are inside or outside ∂V). In addition, a complex edge criterion detects configurations that can induce non-manifold surfaces when a given cell edge intersects ∂V more than once. This strategy allows us to only check voxels that belongs to cell faces to ensure that all cells intersected by ∂V are going to be processed.

Rule 1 Subdivision Rule: *If a cell C is complex, it has to be subdivided. Otherwise, it is no longer subdivided and it is marked as an octree leaf.*

The application of the subdivision rule 1 will build an octree where no leaf cell that intersects ∂V is complex. From this octree, we can extract closed manifold surfaces at any level of resolution.

Adaptive surfaces and manifold verification: in order to adapt our surfaces to the curvature of ∂V, we propose to use a second rule based on the normals at the intersection points of cell edges with ∂V. To calculate the normals, we have used a surface approach based on the measure of the average of unitary normals in the neighbourhood of an edge intersection with ∂V [2].

Rule 2 *Let C be a cell and $I_{i=0\ldots n}$ its intersection points with ∂V. Let be $n_{i=0\ldots n}$ the normals on $I_{i=0\ldots n}$. If $Min(n_i \bullet n_j | i \neq j\} \leq \delta$, C must be subdivided. δ is a user provided parameter with values in the closed interval* $[0, 1]$.

The previous rule gives valuable information about the shape of ∂V inside the cell. As consequence, δ close to 1 will produce an almost regular mesh. On the contrary, δ close to 0 will generate a highly simplified surface. The curvature rule 2 is applied once a cell has succeed the complex cell test. In this way, a normal can be calculated at the unique intersection point of every edge that traverses ∂V and the curvature rule 2 can be checked. Cells that have not been subdivided because the curvature of ∂V exceeds δ inside them are marked as *compact*.

Our subdivision rule 1 ensures that our octree will be subdivided until there are no more complex cells. The limit case will be a regular divided octree, without any non-manifold vertex configuration because of the dual equivalence between the Marching Cubes and the Dual Marching Cubes look-up tables. However, the

Algorithm 2: Octree construction

```
input  : Root cell c. Minimal minLevel and maximal
         maxLevel level subdivision.
output: A regular octree until level minLevel and an
         adaptive octree subdivision between minLevel and
         maxLevel.
Add c to the list toProcess;
while toProcess is not empty do
    currentCell ← First(toProcess)
    n ← Level (currentCell)
    if (n≥ minLevel) and (n ≤ maxLevel) then
        divide ← no
        if Complex (currentCell) then
            divide ← yes
        else
            if (Inhomogeneous (currentCell) and
            (HighCurvature (currentCell)) then
                divide ← yes
            else
                Mark currentCell as a compact leaf.
            end
        end
        if divide then
            Comment: currentCell has to be divided;
            S = {q1, q2, q3, ..., q8} ← Divide(currentCell);
            foreach cell qi in S do
                Add qi in toProcess
            end
        end
    end
end
```

addition of the rule 2 for the curvature can lead to non-manifold configurations. This is due to the local application of subdivision rules 1 and 2. A cell c is not subdivided if it is not complex and its curvature is less than a user defined threshold. That means that the topology of ∂V inside c can be well approximated by using one of the cases of the DMC look-up table. However, c neighbouring cells can keep subdividing until revealing a more complex topology that was not visible from c. This can include ∂V multiple traverses through c edges (see figure 6 top).

In order to solve this, we propose to extract information about ∂V shape in face adjacent cells over the faces of *compact* cells. Even if *Westermann et al.* [19] have proposed a similar approach to fix surface cracks by using an extension of the marching cubes triangulation, their approach is limited to neighbouring cells with at most one level of difference in the octree. Our approach does not have this limitation and it is just guided by the topological complexity of ∂V on the faces of compact cells.

First, for every compact cell c, we get the adjacent cells at the same octree level. Then, for every adjacent cell through a face f, we extract all its leaf cells S that are face adjacent with c. By using the cells in S we build a quadtree with all the faces of cells in S that are adjacent to c. This quadtree contains the values of $F(x, y, z)$ over its vertices and can represent the intersection between V and f produced by the octree subdivision (see figure 6 middle). So, we can apply a complex face and complex edge test and determine if the face is complex. If so, we divide the adjacent compact cell and we add its sub-cells to the compact cells list (see figure 6 middle).

Our algorithm is entirely based on the information provided by the evaluation of $F(x, y, z)$ in the octree vertices. We have implemented this procedure recursively and its stop guarantee is provided by the fact that

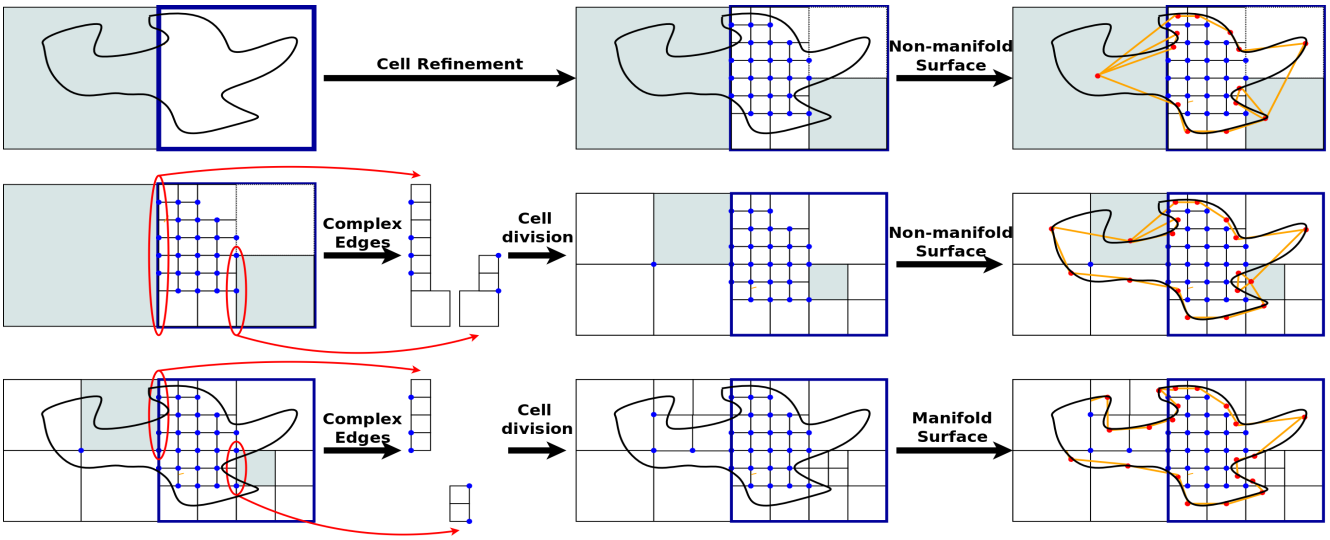


Fig. 6 2D illustration of our local refinement and manifold verification algorithm. First row shows an object in black contained inside two cells (left). Then, we apply our refinement algorithm over the right cell (center). This octree cells configuration will generate a non-manifold configuration (right) with dual nodes (in red) contained in not refined (compact) cells (in light blue). Middle row, we extract the value of octree vertices that are contained inside grey cells edges (left), if those edges are complex, we refine the compact cells (center) and we update the list of compact cells (in grey) to check (right). Bottom row, we repeat this test recursively (left) until verified all compact cells (center). Then, we can extract the surface (right).

neighbouring cells that have the same size will be covered by the DMC look-up-table. In practice, our algorithm stops when all compact cells having complex faces or edges have been eliminated by subdivision (see figure 6 bottom).

Proof of correctness: Let state that in Dual Marching Cubes, all the patches have to pass through intersected edges. In the case of a complex edge, let be C a compact cell at depth n and S a set of cells at level $\{n + x \mid x > 0\}$. Let be E an edge of S and e_i the edges of cells in S so that $\{\forall e_i \mid e_i \subset E\}$ and $\{\forall j \neq i \mid e_j \cap e_i = \emptyset\}$. Without lost of generality, we can say that if we force every edge in C to traverse ∂V only once, we also assure that only one cell in S is traversed by ∂V. This comes from the fact that all edges $e_i \in S$ are disjoint. Then, ∂V can traverse just one cell edge in S.

In the case of a complex face, let F be a face of a compact cell. Then, if we assure that no isolated connected component exits inside F, there cannot be intersected edges to connect with the dual nodes created for the simple connected components around the edge of the compact cell. □

Our local refinement algorithm can be used to produce surfaces that are not only adapted to the curvature but guided by external criteria. For some applications, a model has to be refined locally in some areas corresponding to the interest of the user. An example of this feature is illustrated in figure 7, where our algorithm is applied to the "Lucy" surface model that is progressively refined to obtain a more detailed surface on the top of the model. Our algorithm guarantees that the refined model is still a closed valid manifold.

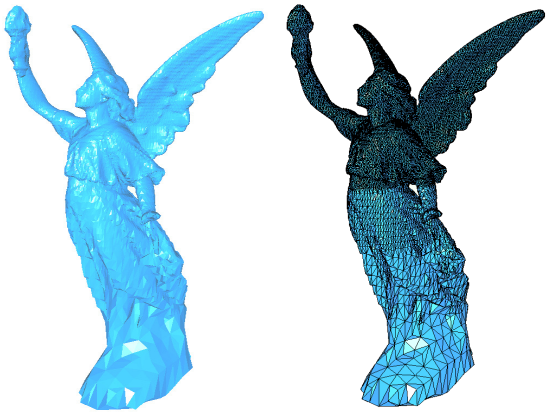


Fig. 7 Progressive "Lucy" (mesh and surface) model extracted from a $512 \times 256 \times 128$ voxels volume. A more refined mesh is produced adaptively on the top of the model.

Finally, in order to improve the general quality of triangles in our surfaces, we can keep the level difference between octree leaf cells in 1, even though, our algorithm is not limited by this.

5 Dual nodes generation

Once we have applied the manifold verification algorithm, we use the Dual Marching Cubes look-up-table (LUT) (see figure 8) to create the surface nodes inside our octree cells.

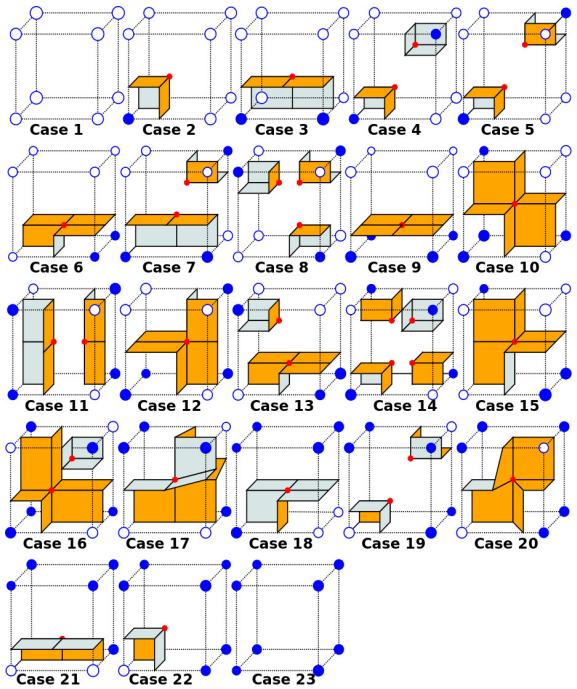


Fig. 8 DMC Look-up table of surface patches generated by using connected components on the graph defined by the cell vertices. Blue vertices are inside ∂V, external faces are orange, internal faces are grey. Dual nodes are noted in red.

However, the application of the DMC LUT does not guarantee that all edges connected by two dual nodes are shared by at most two polygons. If we see a cell as a graph with eight vertices and twelve edges, DMC inserts dual nodes based on the number of connected components that are inside ∂V and generates non-manifold edge configurations in cells sharing an ambiguous face. In figure 9 left, there is only one connected component inside ∂V (in blue) and, in consequence, only one dual node is created.

We have solved this problem by considering the connectivity of the graph of vertices that conforms a cell (eight vertices and twelve edges). As showed in the figure 9, blue vertices are inside the surface ∂V and white ones are outside. Then, a blue connected component is composed by one or more blue vertices that are connected by edges which two endpoints are blue. Respec-

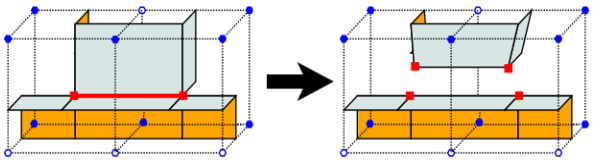


Fig. 9 (Left) Non-manifold edge generated through an ambiguous face since two portions of the surface share a common edge (in red). (Right) Connectivity inversion, two dual nodes are created in each cell and a tunnel in ∂V is recovered. Dual nodes are noted in red.

tively, white connected components are defined in the same way for white vertices. For each cell c with a single blue component and one or multiple ambiguous faces, we enumerate the adjacent cells A_c through the ambiguous faces that also have a single blue component. Then, for each cell in A_c, we extract the white connected components. In these cells, the existence of an ambiguous face implies the existence of at least two such white connected components. Therefore, we use the white components to generate the dual nodes in c and in all adjacent cells in A_c. This strategy produces two vertices instead of one and solves the problem recovering the tunnel in ∂V (see figure 9). Some examples of patches generated by our algorithm are shown in figure 10.

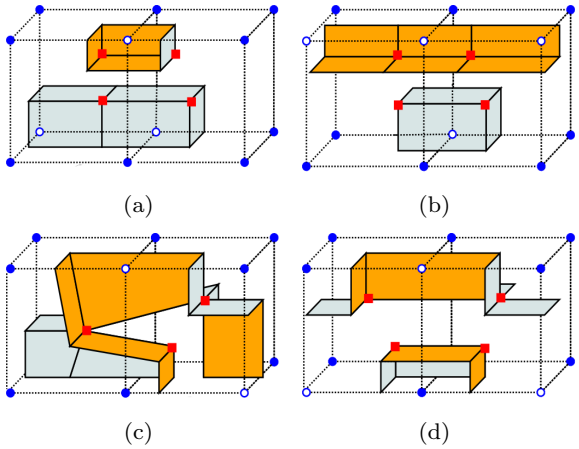
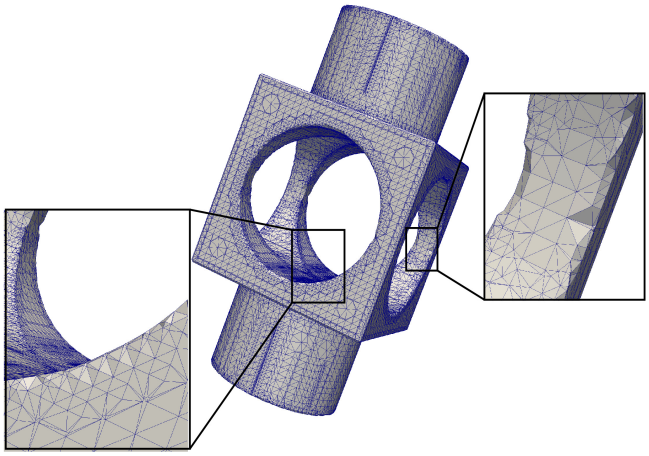
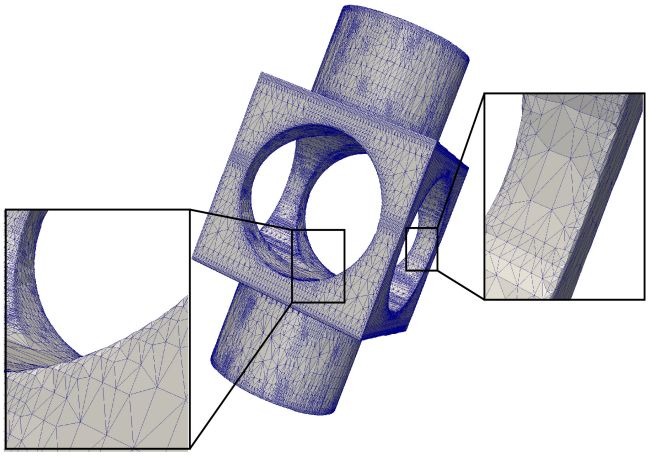


Fig. 10 Four cases where two cells share an ambiguous face. In the cases (a) and (b), there is only one ambiguous face per cell. The use of blue connected components will produce a non-manifold edge. Therefore, we use the white connected components to extract the surface patches. In (c), there is a cell with three ambiguous faces (right) and more that one blue connected component, then, it is possible to use the blue components to produce the patches. In (d), two cells with three ambiguous faces containing more that one blue component that can be used to produce the dual nodes and extract the surface patches.

Locating dual nodes inside cells instead of cell edges produces surfaces having less waving and reconstruction artefacts. In addition, dual nodes inside cells provide more freedom in order to capture sharp edges. To illustrate this, in figure 11, we compare the result of our algorithm with DMC2 [14] on a mechanical "Block" volumetric object of $512 \times 512 \times 128$ voxels. As it can be seen, our algorithm produces smoother surfaces while avoiding reconstruction artefacts especially visible in sharp edges. This is mainly explained because, even if DMC2 aligns the dual grid to the features of the volume, as it uses MC to extract the surface, it localizes the surface nodes on the edges of topological cubes reducing its ability to capture sharp edges. On the contrary, even if our algorithm uses an octree, as it localizes nodes inside cells, it has more freedom to place surface nodes close to the features of the V.



(a) Dual marching cubes (DMC2) with 54.888 triangles.



(b) Our algorithm with 46.888 triangles.

Fig. 11 Comparison of our algorithm with DMC2 [14] and MC [8] on a volumetric "Block" dataset of $512 \times 512 \times 128$ voxels with a curvature parameter λ of 0.9 and a maximal octree depth of 7. DMC2 has problems to capture some of the sharp edges and produces more degenerated triangles.

6 Connectivity generation

A *minimal edge* is one that does not contain any other edge in a neighbouring octree cell (see figure 12). In dual contouring algorithms, there is a one to one relationship between the surface patches and the minimal edges that are intersected by the surface.

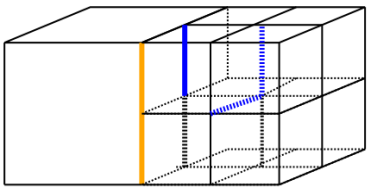


Fig. 12 Examples of minimal edges. The minimal edges does not contain any other smaller edge. Some examples are showed in blue. Non minimal edges contain more that one edge in edge adjacent cells of smaller size (in orange). Not all minimal (or not minimal) edges are highlighted in this figure.

Then, in order to extract the connectivity of the surface, we used an algorithm proposed in Dual Contouring [4] that traverses the octree to enumerate the minimal edges intersected by the surface. This method is based on three recursive methods: CellProc, FaceProc and EdgeProc.
CellProc receives a cell c as parameter, FaceProc receives two face adjacent cells and EdgeProc receives four edge adjacent cells (in the regular case, three in the adaptive case). The algorithm works as follows, CellProc receives a cell c and calls itself recursively in every sub-cell of c. Then, it makes twelve calls to FaceProc with every pair of face adjacent sub-cells. Finally, it makes six calls to EdgeProc with all four sub-cells sharing an edge (see figure 13 top).

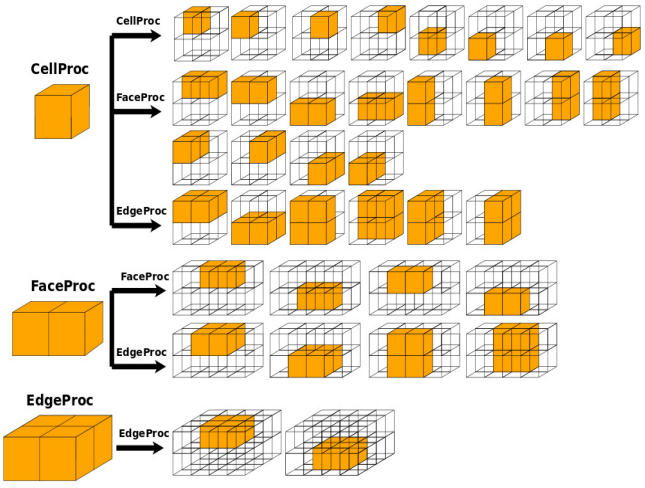


Fig. 13 CellProc, FaceProc and EdgeProc methods as they are used to extract the topology.

FaceProc receives two cells sharing a common face f and calls itself four times with every pair of sub-cells sharing a face contained in f. Then, it makes four calls to EdgeProc with every four sub-cells that share an edge contained in f (see figure 13 middle). Finally, EdgeProc receives four edge adjacent cells and makes two recursive calls to EdgeProc with all four sub-cells sharing a half-edge contained in the edge (see figure 13 bottom).
Only EdgeProc generates surface patches traversing *minimal edges*. When EdgeProc receives four cells that share a minimal edge, it connects the dual nodes in those cells that are related to the minimal edge to form a quad in the uniform case (see figure 14 left) and a triangle in the adaptive one as illustrated in figure 14 right.

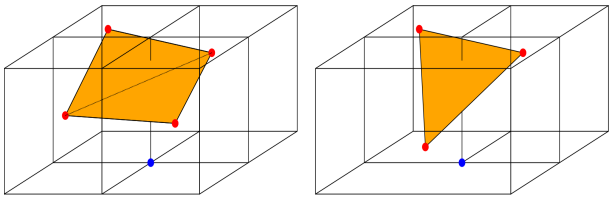


Fig. 14 EdgeProc patch creation. In the regular case, EdgeProc receives four cells (left) and creates a patch with the four minimizers (in red) of edge adjacent cells. (Right) In the adaptive case, EdgeProc receives one of the cells repeated, so, only three cells and it creates a triangular patch. Blue vertex is inside ∂V, all others are outside.

7 Dual nodes localization

Dual nodes localization is important to improve the geometrical approximation of the surface. A good localization algorithm will strongly avoid, or at least reduce, aliasing on the surface. In addition, our method must be robust enough to be applied over the usually noisy volumetric data surfaces.
We propose a dual node localization algorithm based on the centroid b_s of every connected component of V inside a cell. Let be C a leaf cell in the octree where $V \cap C \neq \emptyset$. As we know the dimensions of C and its mass M, we can obtain its centroid b_c. Then, we calculate the centroid of $V \cap C$ named b_s and its mass m_s. Finally, the mass and barycentre of the complementary b_e of $(V \cap C)$ can be calculated using equation $Mb_c = m_s b_s + m_e b_e$. Then, we use b_s, b_e and their respective masses to estimate an initial position v_p for the dual node that will lie on the segment between b_s and b_e.
However, in some configurations, this initial position v_p can lie far from the surface (either inside or outside ∂V). To approach the dual node to the surface, we use the segment defined by b_s and b_e and noted as S. First,

we move the dual node from its initial position v_p to the intermediate point of the segment connecting v_p with the endpoint of S that traverses ∂V, noted v_e. Then, we test if this intermediate point is over the surface, if it is the case, we stop, if not, we apply recursively the same operation to the half-segments of the segment v_p to v_e. In order to validate if the node v is over the surface, we use the voxels spacing as an interval and we check if the current node position falls into an interval that makes part of the discrete surface of ∂V. Dual node localization process is illustrated in figure 15.

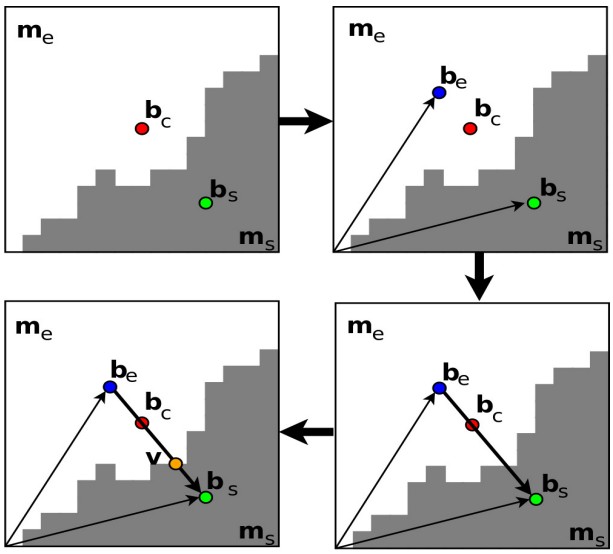
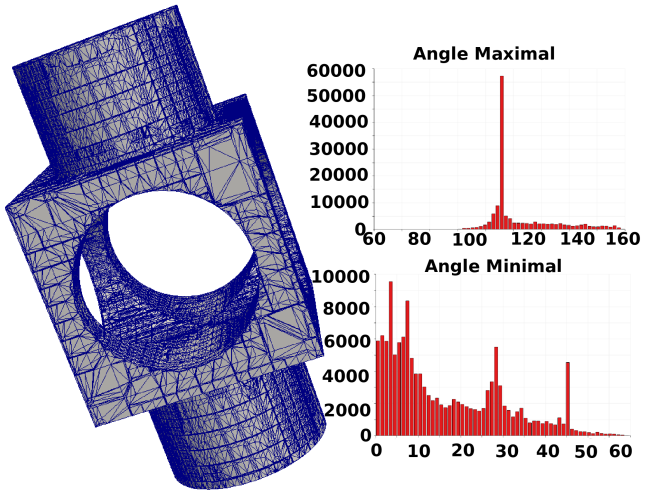


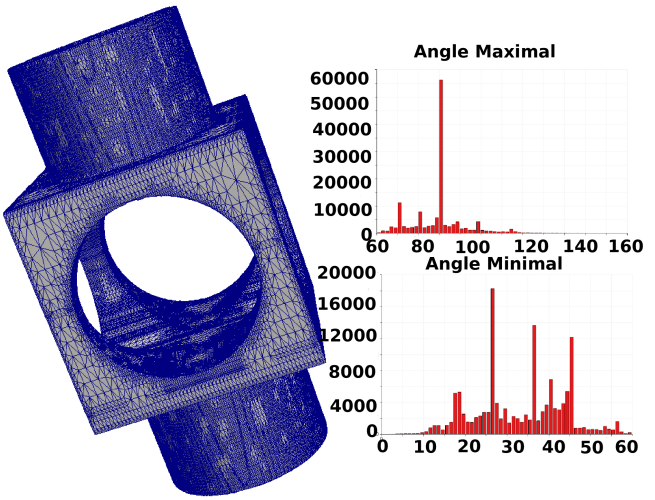
Fig. 15 2D illustration of the algorithm to estimate a good dual node localization by using connected components. Cell centroid (red), $V \cap C$ centroid (green), $(V \cap C)^c$ centroid (blue), dual node (yellow).

Our dual node localization method has two main characteristics: it sticks dual nodes to the volumetric surface of the component and, as each surface component inside a cell must have a simple connected boundary, it tends to localize the node inside the kernel or center area of the surface which points can be connected to any other point of the surface by using a line [18]. These properties are useful to avoid self intersections and to improve the general quality of triangles.

In figure 16, we compare our approach with the Dual Marching Tetrahedra (DMT) algorithm [9]. In figure 16a, DMT generates a good approximation but the mesh and the histograms of angles show a lot of triangles degenerated having very small (under 5°) minimal angles or huge maximal angles (over 100°). This is because DMT uses a dual tetrahedral grid aligned to the features of ∂V combined with Marching Tetrahedra triangulation. On the other side, as it can be seen in figure 16b, our approach produces a more homogeneous minimal and maximal angle distributions around 30° and 85° respectively, producing far less degenerated triangles.



(a) Dual marching Tetrahedra with 133.512 triangles and average min/max angles of 16.5/103.4.



(b) Ours with 127.188 triangles and average min/max angles of 33.1/89.6.

Fig. 16 Comparison between the DMT algorithm of Manson and Schaefer and our algorithm on a volumetric "Block" dataset of $512 \times 512 \times 256$ voxels with a curvature parameter λ of 0.9 and a maximal octree depth of 8. Histograms show that our algorithm obtains much better min/max angle distributions.

We considered other localization methods such as Quadric Error Functions [3], however, these methods can easily localize surface nodes outside the original volumetric object (generating self-intersections) and be strongly affected by noisy segmentation processes that will not produce a smooth discrete surface. Our experiments have shown that our method works well on smooth or noisy segmented surfaces.

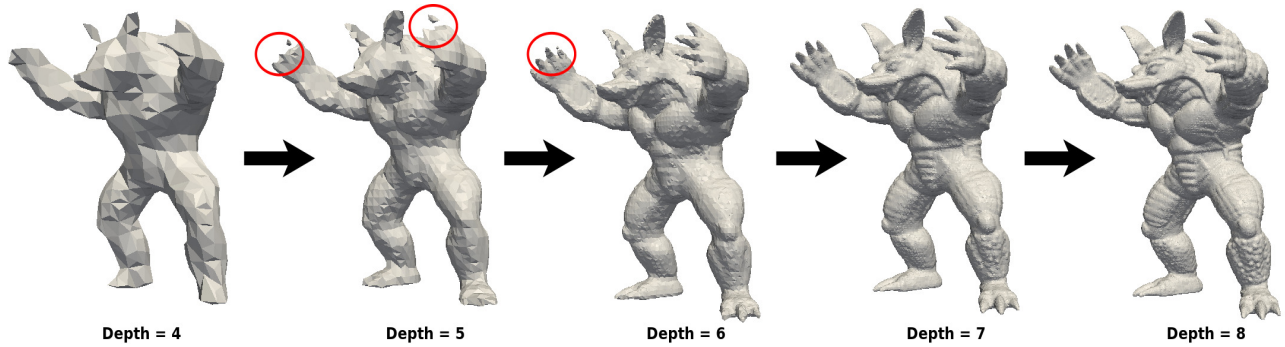


Fig. 17 Multi-resolution "Armadillo" model ($512 \times 512 \times 256$ voxels) with a curvature threshold of 0.9 and five different resolutions based on a maximal octree depth from 4 (1.458 triangles) to 8 (347.922 triangles). The topology of the surface changes through different resolutions (see red circles).

8 Results

One of the main characteristics of our solution is its ability to generate closed manifold surfaces at any resolution level. It can be used to produce multi-resolution models based on a user defined resolution. In figure 17, several "Armadillo" models are built based on a maximal octree level from 4 to 8. These models do not necessary have the same topology (see red circles around fingers) because thin features cannot be captured at lower resolutions and some sections of surface can be separated in the final mesh. However, this feature can be useful in visualization where the speed of transmission is more important than the topology of the model. On the contrary, when the topology of the model is an important factor, the curvature threshold can be used to obtain simplified models that will keep the same topology of the original volumetric object. This is shown in figure 18 where several topological equivalent surfaces are extracted from an "Armadillo" model of $512 \times 512 \times 256$ voxels. Observe that all surfaces keep the same topology and no thin components are separated.

Table 1 resumes sizes, approximation and timing statistics for the surfaces of figure 18. Hausdorff and RMS distance (measured with Metro [1]) in the different models are not strongly affected by the curvature simplification and a good approximation is still reached with a high level of simplification. Times in the table include the octree construction. The most expensive step is the normals calculation and our dual node localization algorithm because they are calculated directly on volumetric data.

Curvature	#Triangles	Hausdorff	RMS	Time(secs)
0.9	89094	0.0043	0.15	9.1
0.7	42822	0.0091	0.21	8.3
0.5	27274	0.012	0.33	7.3
0.3	20740	0.014	0.41	6.5
0.1	16922	0.014	0.45	6.3

Table 1 Statistics obtained for the surface extraction of a volume "Armadillo" (512^3 voxels) with different curvature values. Surfaces are always a closed manifold.

In order to compare our solution with other relevant methods in the literature, we used the Adaptive Marching Cubes (AMC) of Kazhdan *et al.* [5] and the Dual Marching Cubes (DMC2) of Schaefer *et al.* [14]. We applied these algorithms over a set of volumetric datasets extracted from classic polygonal models (see our recon-

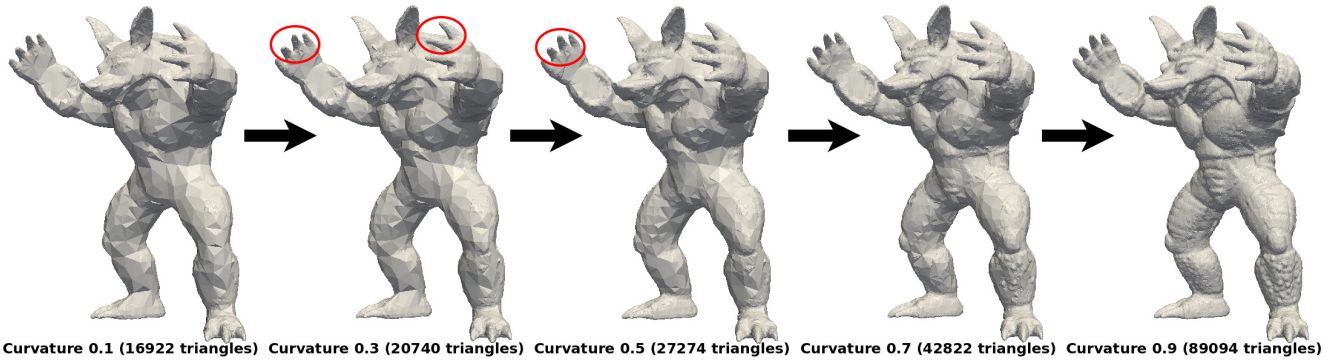


Fig. 18 Simplified surfaces with respect to the curvature threshold. Surfaces are extracted from an "Armadillo" volume of $512 \times 512 \times 256$ voxels using an octree of maximal level 7. Observe how thin features as the fingers are connected in all the surfaces (see red circles). Statistics of the extraction process for these surfaces are provided in table 1.

Criterion	# Triangles			# Degenerated* Triangles			Minimal Average Angles		
Models/Methods	AMC	DMC2	Ours	AMC	DMC2	Ours	AMC	DMC2	Ours
Dragon	122074	**107342**	133586	2360	11942	**140**	31.4	28.2	**33.1**
Horse	**36408**	42316	61262	850	5020	**162**	31.23	27.6	**32.4**
Buddha	**119124**	134342	165921	1411	12459	**271**	31.9	28.8	**32.84**
Armadillo	137098	156820	**90773**	3160	15741	**173**	31	27.88	**36.8**
Block	**55250**	71144	57796	1136	8659	**58**	**33.94**	29.86	31.4
FanDisk	**43094**	52344	46896	1252	7040	**63**	32.53	27.6	**33.4**

Table 2 Comparison of our method with Adaptive Marching Cubes (AMC) [Kazhdan *et al.* 2007] and Dual Marching Cubes (DMC2) [Schaefer et Warren 2004]. All surfaces are extracted with an octree of maximal depth of 8 in order to capture the finer components of the surface and a curvature λ parameter of 0.9 to reduce the number of triangles in the flat areas of the volume. * A degenerated Triangle is defined as having a minimal angle of at most 2 degrees.

structions in figure 19). We set the curvature parameter λ in 0.9 to force a cell subdivision if there is a slight curvature in the surface ∂V, and the maximal depth of the octree in 8 so to guarantee that we capture the main features of the models. All these results are summarized in table 2.

As it can be seen, even if all three algorithms are almost equivalent in mesh size and minimal angles, our algorithm generates consistently far less bad-shaped triangles defined as triangles containing minimal angles smaller that 2 degrees. In the case of AMC, MC triangulation is used inside cells and its technique to close the surface where the resolution changes usually produces a lot of bad-shaped triangles. On the other side, DMC2 builds a dual hexahedral grid from an octree using a dual node inside every octree cell (see [14]) but it still uses the MC triangulation. In addition, dual cells are not necessarily convex and their shapes can be very twisted. On the contrary, our method relies on the octree cells, uses multiple dual nodes inside every cell and provides more freedom in surface nodes localization.

Our experiments over multiple curvature and octree depth parameters showed that our method keeps generating surfaces with far less degenerated triangles. This is understandable because the main reasons for creating bad shaped triangles in AMC and DMC are their restrictions in the nodes localization which is not fundamentally affected by a higher resolution (higher octree depth), neither a more regular space subdivision (bigger curvature threshold).

9 Conclusions and perspectives

We have presented a robust surface extraction algorithm that can be used to obtain adaptive surfaces from highly anisotropic volumetric data. Our method operates directly on volumetric data and does not need any pre-calculated information. We have seen that almost all methods tackle one or two particular issues but they do not solve all the iso-surface extraction problems at once. It is clear that is not easy to find a way to address all limitations but our method provides a good trade-off between surface quality and geometrical accuracy. As our solution can obtain manifold meshes at any level of resolution, it can be used for data exploration, visualization and transmission applications.

In future work, we would like to improve the surface approximation by using more precise curvature estimators inside octree cells. Another interesting possibility is to use curvature information to move the division planes to the more curved zones of the surface instead of dividing octree cells at middle points to better capture the surface features. In addition, we are implementing an out-of-core version of our solution combined with parallel strategies in order to process huge datasets.

Acknowledgements

The authors would like to thank the Stanford 3D Scanning Repository and Georgia Tech Large Geometric Models Archive for the models used in this article. We would also like to thank the reviewers for their

Fig. 19 From left to right: Dragon, Horse, Buddha, Armadillo, Block and FanDisk models. All models are extracted from a $512 \times 512 \times 128$ voxels representation.

valuable commentaries. This article is an extended version of an article [16] presented in the french conference CORESA 2012.

References

1. Cignoni, P., Rocchini, C., and Scopigno, R. Metro : measuring error on simplified surfaces.
2. Flin, F. Adaptative estimation of normals and surface area for discrete 3d objects. *IEEE Transactions on Image Processing 14*, 5 (2005), 585–596.
3. Garland, M., and Heckbert, P. Surface simplification using quadric error metrics. SIGGRAPH, pp. 209–216.
4. Ju, T., Losasso, F., Schaefer, S., and Warren, J. Dual contouring of hermite data. In *Proceedings of SIGGRAPH* (2002), pp. 339–346.
5. Kazhdan, M., Klein, A., Dalal, K., and Hoppe, H. Unconstrained isosurface extraction on arbitrary octrees. In *Proceedings of Symposium on Geometry Processing, SGP* (2007), pp. 125–133.
6. Kobbelt, L. P., Botsch, M., Schwanecke, U., and Seidel, H.-P. Feature sensitive surface extraction from volume data. In *Proceedings of SIGGRAPH* (2001), pp. 57–66.
7. Lewiner, T., Mello, V., Peixoto, A., Pesco, S., and Lopes, H. Fast generation of pointerless octree duals. In *Symposium on Geometry Processing (Computer Graphics Forum)* (july 2010), vol. 29, pp. 1661–1669.
8. Lorensen, W., and Cline, H. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of SIGGRAPH* (1987), vol. 87, pp. 163–169.
9. Manson, J., and Schaefer, S. Isosurfaces over simplicial partitions of multiresolution grids. *Computer Graphics Forum 29*, 2 (2010), 377–385.
10. Nielson, G., and Hamann, B. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proceedings of the 2nd conference on Visualization'91* (1991), IEEE Computer Society Press, p. 91.
11. Nielson, G. M. Dual marching cubes. In *Proceedings of the conference on Visualization* (2004), IEEE Computer Society, pp. 489–496.
12. Schaefer, S., Ju, T., and Warren, J. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics 13* (May 2007), 610–619.
13. Schaefer, S., and Warren, J. Dual contouring: The secret sauce. Tech. Rep. TR02-408, Department of Computer Science, Rice University, 2002.
14. Schaefer, S., and Warren, J. Dual marching cubes: Primal contouring of dual grids. In *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference* (2004), IEEE Computer Society, pp. 70–76.
15. Shu, R., Zhou, C., and Kankanhalli, M. Adaptive marching cubes. *The Visual Computer 11*, 4 (1995), 202–217.
16. Uribe Lobello, R., Denis, F., and Dupont, F. Génération de surfaces adaptatives à partir de données volumiques binaires. In *CORESA* (May 2012). Prix Jeune chercheur.
17. Varadhan, G., Krishnan, S., Kim, Y., and Manocha, D. Feature-sensitive subdivision and isosurface reconstruction. In *Visualization, 2003* (2003), IEEE, pp. 99–106.
18. Varadhan, G., Krishnan, S., Sriram, T., and Manocha, D. Topology preserving surface extraction using adaptive subdivision. *Symposium on Geometry Processing - SGP* (2004), 235–244.
19. Westermann, R., Kobbelt, L., and Ertl, T. Real-time Exploration of Regular Volume Data by Adaptive Reconstruction of Iso-Surfaces. In *The Visual Computer 1999* (1999).
20. Wilhelms, J., and Van Gelder, A. Octrees for faster isosurface generation. *ACM Trans. Graph. 11* (July 1992), 201–227.