A parallel dual marching cubes approach to quad only surface reconstruction

Roberto Grosso¹ · Daniel Zint¹

Accepted: 12 April 2021 / Published online: 6 June 2021
© The Author(s) 2021

Abstract

We present a novel method that reconstructs surfaces from volume data using a dual marching cubes approach without lookup tables. The method generates quad only meshes which are consistent across cell borders, i.e., they are manifold and watertight. Vertices are positioned exactly on the reconstructed surface almost everywhere, leading to higher accuracy than other reconstruction methods. A halfedge data structure is used for storing the meshes which is convenient for further processing. The method processes elements in parallel and therefore runs efficiently on GPU. Due to the transition between layers in volume data, meshes have numerous vertices with valence three. We use simplification patterns for eliminating quads containing these vertices wherever possible which reduces the number of elements and increases quality. We briefly describe a CUDA implementation of our method, which allows processing huge amounts of data on GPU at almost interactive time rates. Finally, we present runtime and quality results of our method on medical and synthetic data sets.

Keywords Dual marching cubes · Quad meshing · Parallel mesh reconstruction · GPU programming

1 Introduction

Implicitly defined surfaces from real-valued three-dimensional scalar functions are usually computed with the marching cubes (MC) algorithm [25]. This method requires as input the iso-value defining the surface and processes data cell-wise. In this work, *cell* refers to voxels in uniform grids and hexahedra in more general structured grids. The algorithm can be easily and efficiently parallelized on a GPU which makes this technique versatile and attractive for applications in different areas of science and engineering. Nevertheless, a MC surface shows two major drawbacks, triangles are poorly shaped and the mesh is not topologically correct, i.e., it might be inconsistent across cell borders and not homeomorphic to the underlying surface as defined in Eq. (1) [29].

An alternative strategy to generate consistent triangulations of the iso-surface was presented in [16,32,36] and is

based on the following observations. Given an iso-value ι_0 , the iso-surface is defined as:

$$S_{\iota_0} = \{(x, y, z) \in \mathbb{R}^3 | T(x, y, z) = \iota_0\}, \quad (1)$$

where $T : \mathcal{M} \rightarrow \mathbb{R}$ is the trilinear interpolant defined on the hexahedral mesh $\mathcal{M}$. The intersection of the iso-surface with a cell is a set of closed curves, each being a C^0 -continuous collection of hyperbolic arcs. Up to four different components or branches of the iso-surface can intersect a cell, resulting in up to four independent closed curves, hereafter referred to as *MC polygons*. They can be easily computed and are consistent across cell borders if the asymptotic decider [18,31] is used to resolve ambiguous cases.

In a structured hexahedral mesh, an edge is shared by four cells. If an edge is intersected by the iso-surface, then all four cells contain a MC polygon incident to this edge. A quadrilateral is obtained by connecting the vertex representatives of the four MC polygons.

A novel dual marching cubes (DMC) algorithm was presented in [19], which does not require a lookup table, generates watertight meshes, and runs in parallel on GPU. The algorithm output is a high-quality quad only mesh which accurately represents the underlying geometry as defined by Eq. (1). In this work, we extend the results presented in [19].

✉ Daniel Zint
daniel.zint@fau.de

Roberto Grosso
roberto.grosso@fau.de

¹ Visual Computing, Friedrich-Alexander-Universität
Erlangen-Nürnberg, Erlangen, Germany

A different back projection method for the vertex representatives was developed which positions the vertices exactly on the iso-surface of Eq. (1) almost everywhere. Only in extreme cases with numerical instability it might be that a fall-back is required which is not on the iso-surface, but in our tests such a case never appeared. A parallel vertex and face coloring scheme is introduced. Neighborhood and connectivity information is encoded using a halfedge data structure. A limitation of DMC was discovered concerning tunnels in the iso-surface which cover two neighbor cells, i.e., a piece of surface homeomorphic to a cylinder that extends across two neighbor cells. Such tunnels create non-manifold edges in the DMC mesh, which are shared by four faces. We show how to generate a valid halfedge data structure that handles these non-manifold edges correctly. Furthermore, we implemented parallel simplification methods to reduce vertices and elements with vertex valence pattern 3-X-3-Y, with $X, Y \geq 5$ and 3-3-3-3 which commonly appear in DMC meshes computed from volume data, thus improving the distribution of vertex valences in the mesh. For the pattern 3-X-3-Y, we use the face colors to remove a much higher number of irregular elements in comparison to the method proposed in [19]. A side effect of simplification is that some vertices are not positioned exactly on the iso-surface anymore.

In the following, we first present a brief review of previous work emphasizing the problem of topological consistency and parallel reconstruction. In Sect. 3, we present the parallel DMC algorithm, and in Sect. 4, we describe our parallel implementation of two mesh simplification techniques. In Sect. 5, we show the performance of the methods presented, and in Sect. 6, we give some comments on the results. The source code is available at GitHub [17].

2 Related work

Research contributions in the area of iso-surface extraction from volume data can be classified into three main groups, *standard* marching cubes (MC) and its extensions to resolve topological correctness and consistency; dual marching cubes which extract quad meshes dual to the MC polygons; and primal contouring, which computes an iso-surface from the dual grid or the dual of an octree.

Methods for computing iso-surfaces based on the standard MC, which was presented in [25], have to deal with the problem of inconsistent meshes across cell borders. Furthermore, MC methods might generate meshes which are not homeomorphic to the iso-surface as defined by Eq. (1). Methods presented so far are either based on extended lookup tables, or they first compute the intersection of the iso-surface with a cell resulting in a MC polygon which is consistently triangulated.

After Dürst [11] observed that MC does not consistently triangulate the iso-surface across cell borders, Nielson and Hamann [31] introduced the asymptotic decider to resolve ambiguities at the cell faces. Natarajan [28] discovered interior ambiguities and introduced an extended lookup table. Chernyaev [5] modified the lookup table increasing the number of cases up to 33 in total, which is commonly called MC33. Different authors proposed new techniques to improve performance or to solve topological inconsistencies [6,7,12,22,24,26,27,29]. For each ambiguous face, special subcases have to be considered which results in a large number of configurations. All these methods have the same issue; namely, they generate many triangles with poor quality.

Algorithms were presented which resolve ambiguities without using a lookup table. The first method to compute iso-surfaces based on the intersection of the surface with the cell faces was proposed by Pasko et al. [32]. Renbo et al. [36] developed a triangulation algorithm which does not use lookup tables. These methods process unambiguous and ambiguous cells in the same manner, which is much more computationally intensive than the MC algorithm. Grosso [16] developed a hybrid technique which processes unambiguous cells with the standard MC. Ambiguous cells are triangulated based on a set of rules applied to the MC polygons. This method has the advantage of not relying on lookup tables in ambiguous cases.

In order to improve performance and to overcome the problem of generating a large amount of triangles many parallel strategies were proposed in the literature. A parallel iso-surface algorithm which is combined with edge collapses was presented in [10,42]. It is a modification of the *tandem* algorithm introduced in [2]. Parallel implementations become more complex if the output has to be a data structure with connectivity and neighborhood information. A GPU-based technique to reconstruct and smooth the iso-surface by repositioning the vertices without changing mesh topology was introduced in [4]. A method to compute standard MC on multiple GPUs was described in [9]. A major problem of these algorithms is that they use large buffers to compute a consistent numbering of the vertex indices. Usually, a unique vertex index is computed by counting cells or edges via a prefix sum. This is not viable in our case, as buffers become very large for volume data consisting of hundreds of millions of vertices. We opted for a different technique which results in a much simpler algorithm that delivers good performance results and allows to compute iso-surfaces from very large volume data.

The dual marching cubes algorithm presented by Nielson [30] is a different strategy to reconstruct an iso-surface from volume data. The intersection of the iso-surface with the cell can be approximated by a polygon on the cell faces. This is what we call the MC polygon in the previous section. The DMC algorithm computes the dual of these MC polygons.

Fig. 1 Local index convention for vertices and edges in a cell

The dual to the MC polygons is a quad only mesh because each edge in the volume mesh is shared by four cells. This algorithm relies on the lookup table introduced in [29] which consists of 23 basis cases. Ambiguities are resolved by introducing sub-configurations. This method is a generalization of the SurfaceNets proposed in [8,14]. A parallel implementation of the DMC algorithm is presented in [23]. The method generates a 1-ring neighborhood data structure and approximates the surface by using error quadrics. Nevertheless, it relies on large buffers whose size depends on the number of edges to compute unique vertex indices via prefix sums.

Primal contouring [38] is an alternative method for extracting iso-surfaces from volume data. It first approximates the data by computing an octree. For each cell of the octree, the method computes a single vertex representing the surface intersection with the cell. The vertex is placed within the cell by optimizing a quadratic functional based on error quadrics. The method just computes a single vertex for each cell in the octree, and therefore the resulting mesh might not be manifold. Subsequently different works were proposed that mainly deal with the problem of computing manifold meshes out of an octree [21,34,37]. All these methods generate hybrid meshes from a hierarchical data structure.

3 Dual marching cubes

We use the index convention for vertices and edges shown in Fig. 1. For instance, in the unit reference cell $[0, 1] \times [0, 1] \times [0, 1]$ we have $v_0 = (0, 0, 0)$, and $e_0 = \{v_0, v_1\}$.

The restriction of the trilinear interpolant T to a unit reference cell has the form

$$\begin{aligned} F(u, v, w) = & (1-w)[f_0(1-u)(1-v) + f_1u(1-v) \\ & + f_2(1-u)v + f_3uv] \\ & + w[f_4(1-u)(1-v) + f_5u(1-v) \\ & + f_6(1-u)v + f_7uv], \end{aligned} \quad (2)$$

Fig. 2 MC polygons approximating the intersection of the iso-surface with the cell faces

Fig. 3 A plane cutting through a uniform grid. The intersections of cell faces with the plane give the MC polygons

where $(u, v, w) \in [0, 1]^3$ are local coordinates and f_i are the function values at the cell vertices v_i . Up to four branches of the iso-surface obtained from $F(u, v, w) = \iota_0$ might intersect the cell. In Fig. 2a, we show an iso-surface with only one component intersecting the cell at all twelve edges. In Fig. 2b, we see the corresponding hyperbolic arcs at the cell faces, and in Fig. 2c, the MC polygon used to approximate the hyperbolic arcs.

For each branch of the iso-surface, DMC selects a single vertex within the cell which *represents* the surface. For the

Fig. 4 Two configurations which have the topology of a cylinder (tunnel). The second configuration causes trouble for DMC because both voxels get only one vertex which eventually results in a non-manifold edge connecting those two vertices

case shown in Fig. 2d, three vertex representatives have to be computed.

Every edge in a hexahedral grid is shared by four cells, excluding boundary edges. An iso-surface branch that intersects an edge, will also intersect all four cells sharing this edge. Therefore, connecting the representative vertices of this branch from all four incident cells will generate a quadrilateral. If the MC polygons are constructed using the asymptotic decider [31], the generated mesh is topologically consistent across cell borders. The mesh is called dual, because each vertex in the MC polygon has a corresponding quadrilateral in the dual mesh, and each vertex of the dual mesh represents a MC polygon.

For example, assume a plane iso-surface branch cutting through a uniform grid. Figure 3a shows the intersections of the branch with the cells, i.e., the MC polygons. The dual of this polygonal mesh is the quadrilateral DMC mesh (Fig. 3b).

DMC generates meshes which have less vertices and better shaped elements than the meshes generated by the standard MC algorithm [30]. Nevertheless, the generated iso-surface might not be topologically correct. For certain configurations which can typically be found in medical data, the iso-surface (1) will not be homeomorphic to the reconstructed DMC mesh. The DMC algorithm as it was formulated above cannot reconstruct tunnels of sub-voxel size [16] (Fig. 4a). The standard MC algorithm cannot reconstruct these tunnels either. If the iso-surface forms a tunnel which extends across two cells, DMC generates a non-manifold edge connecting the vertex representatives of the MC polygons in each of the neighbor cells (Fig. 4b). This non-manifold edge will be shared by four faces (Fig. 7). In Sect. 3.2, we show how to keep the halfedge data structure consistent in that case.

The parallel DMC algorithm we propose generates an indexed face set for the quadrilateral mesh, where the ele-

ments are all consistently oriented. Optionally, a halfedge data structure carrying neighbor information can be computed. After initializing buffers the global structure of the proposed algorithm consists of two main steps: (1) compute the DMC mesh; (2) generate a halfedge data structure. In the initialization step, buffers are created and default values are set with the help of simple CUDA kernels. In the next subsections, we briefly describe how to compute the DMC quadrilateral mesh and subsequently generate the halfedge data structure.

3.1 DMC quadrilateral mesh

The DMC quadrilateral mesh is computed with two CUDA kernels. The first kernel proceeds cell wise and computes the vertex representatives assigning to each vertex a unique index. For each edge intersected by the MC polygon, the vertex index is stored by the kernel in a hash table where the key is the edge index in the volume grid. For each edge in the hash table, the indices of the vertices constituting the quadrilateral are stored by the threads processing the incident cells. It is mandatory to use a hash table to enable processing of volume data consisting of hundreds of millions of vertices, see Sect. 5, only few edges are intersected by the iso-surface compared to the total number of edges in a large volume grid. The second kernel collects the quadrilaterals from the hash table and stores each element into an index buffer. Each thread started by the first kernel has to carry out the following processing steps for the cell: (1) compute MC polygons, (2) estimate vertex representatives for each MC polygon, and (3) store vertex index in the hash table for each edge being intersected by the MC polygon and (4) compute face colors. In order to improve performance, the kernel returns immediately if the iso-surface does not intersect the cell. In the following, these processing steps are explained in more detail.

3.1.1 Computation of MC polygons

A cell might be intersected by up to four disconnected branches of the iso-surface. Therefore, we expect to obtain up to four closed MC polygons. The method implemented for this purpose returns the number of polygons, the size of each polygon, and the indices of the intersected edges. This quantities can be computed in two steps. First, the cell is processed face wise. The intersection of the iso-surface with a face is given by a segment, i.e., an edge of the MC polygon (Fig. 2). For each segment on a face, the indices of the start and end edge are computed. Segments are oriented such that vertices with function values larger than the iso-value are located to the left of the segments with respect to the face. Ambiguous cases are solved with the asymptotic decider [31]. In a second step, segments are connected to

Fig. 5 Positioning of vertex representatives by sampling the iso-surface branches. Samples that lie outside the cell or on a different branch are removed. The sample which is closest to the center of gravity of the MC polygon is chosen as vertex representative

build up closed polygons. MC polygons, i.e., their size and the indices of the edges being intersected, can be encoded in a single 64 bit unsigned long long integer.

3.1.2 Positioning of vertex representatives

Within a cell, a vertex is computed for each iso-surface branch. The vertex representatives are placed exactly on the iso-surface S_{ι_0} with the iso-value ι_0 defined by Eq. (1). It must be ensured that vertex representatives are positioned on the right surface branch, otherwise the resulting mesh will be non-manifold. We find vertex representatives by sampling the iso-surface branches. Two of the three parameters (u, v, w) are sampled, whereas the remaining parameter is computed from the trilinear interpolant, Eq. (2). For example, if (u, v) are sampled, parameter w is:

$$w = \frac{\iota_0 - g_1(u, v)}{g_2(u, v) - g_1(u, v)}, \quad \text{with} \quad (3)$$

$$g_1(u, v) = f_0(1-u)(1-v) + f_1u(1-v) + f_2(1-u)v + f_3uv, \quad \text{and}$$

$$g_2(u, v) = f_4(1-u)(1-v) + f_5u(1-v) + f_6(1-u)v + f_7uv. \quad (4)$$

Similar equations are obtained if (u, w) or (v, w) are sampled. Using the right parameters for sampling is of importance for stability. We choose as sampling space the two parameters where the MC polygon has the largest projection. Furthermore, we restrict the parameter space to the bounding box of the MC polygon, e.g., $[u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$. If

a sample lies outside of the cell, e.g., $w < 0$ or $w > 1$, it is discarded. Finally, the sample which is closest to the center of gravity of the MC polygon is chosen as vertex representative. Figure 5 shows the sampling process for a cell with three branches.

The set of sampling points can be further restricted to ensure that the vertex representative is placed on the right branch. We use the bounding boxes of the branches to assure that samples are placed on the correct branch. If there are several branches within a cell, only one of them can intersect more than three edges [16]. There is only one exception, which is when two branches intersect four edges each. If a branch intersects three edges, the sample which is closest to the center of gravity of the MC polygon is on this branch (Fig. 5). For more complex configurations, we discard all samples within the bounding boxes of other branches. Thus, we ensure that samples are only considered when they are on the correct branch.

Theoretically, one could construct a case where two bounding boxes almost completely overlap. In that case, we would need a very fine sampling to find positions which are not within the bounding box of another branch. We use a size 7×7 set of samples. If we do not find any valid position, we try again with a size 25×25 set of samples. If this also does not deliver a valid position, we use the center of gravity of the MC polygon. However, this fall-back was never required in any of our tests.

Normals are computed in two steps. First, the gradient of the scalar function is estimated at the cell vertices by using central difference. Second, the gradient is interpolated trilinearly at the position of the vertex representative and then normalized. We use central differences because it has a better truncation error than forward or backward difference. The computation of the gradient using the trilinear interpolant has the same approximation error as forward difference. Computing normals using central difference results in more appealing visualizations.

3.1.3 Computation of the quadrilaterals

For each edge that is intersected by the iso-surface, a quadrilateral is generated by connecting the vertex representatives of the corresponding MC polygons in the incident cells. Quadrilaterals are oriented such that their normals point toward function values larger than the iso-value.

Each quadrilateral is uniquely assigned to an edge of the volume mesh. Quadrilaterals are stored in a hash table where the key is the unique index of the edge in the grid. We use open hashing and quadratic probing to find an empty bucket in the hash table. Hash tables were chosen to be twice as large as the expected number of elements in table. A quadrilateral is represented by an array of four integers. For each cell, the index of the vertex representative has to be stored at the

Fig. 6 Construction of a quadrilateral viewed from the four correspondent cells. The cell that contributes vertex B has the iso-surface intersection on edge $e_0 = \{v_0, v_1\}$. The very same edge is called $e_2 = \{v_2, v_3\}$ in the cell that contributes vertex A. All possible edge cases are listed in Table 1

Table 1 How to build quadrilaterals depending on the edge configuration

$e = \{v_i, v_j\}$	$f_i \geq t_0, f_j < t_0$	$f_i < t_0, f_j \geq t_0$
$e_0 = \{v_0, v_1\}$	B	B
$e_1 = \{v_1, v_3\}$	B	B
$e_2 = \{v_2, v_3\}$	C	A
$e_3 = \{v_0, v_3\}$	C	A
$e_4 = \{v_4, v_5\}$	A	C
$e_5 = \{v_5, v_7\}$	A	C
$e_6 = \{v_6, v_7\}$	D	D
$e_7 = \{v_4, v_6\}$	D	D
$e_8 = \{v_0, v_4\}$	B	B
$e_9 = \{v_1, v_5\}$	C	A
$e_{10} = \{v_3, v_7\}$	D	D
$e_{11} = \{v_2, v_6\}$	D	D

The table indicates which edge contributes which vertex of a quadrilateral

right position within this array to construct quadrilaterals which are consistently oriented. We are using the naming convention presented in Fig. 1. Figure 6 demonstrates how to save vertex indices properly. Edge $e_0 = \{v_0, v_1\}$ is shared by four neighbor cells. In the other three cells, it will have the names $e_4 = \{v_4, v_5\}$, $e_6 = \{v_3, v_7\}$, and $e_2 = \{v_2, v_3\}$. Assume that $f_0 \geq t_0$ and the index B of the vertex is stored at the first position of the index array. The thread processing the cell where this edge has the name e_4 has to store the index C of the vertex representative at the second position in the array. Similarly, the thread processing the cell where the edge has the name e_6 stores the index D at the third position and finally, the thread processing the cell where the edge has the name e_2 stores the vertex A at the fourth position. All possible cases are summarized in Table 1. Each quadrilateral is computed by four threads and stored in a hash table as $\{\text{key}, [B, A, D, C]\}$.

A kernel is in charge of computing the vertex representatives from each cell. The kernel processes the input data

cellwise. It computes the vertex representatives and corresponding normals for all iso-surface branches in each cell. These vertices are interior to the cell; thus, the kernel can assign a unique global index to the vertices which is required by the mesh data structure. The index corresponds to the position of vertex and normal within a buffer and is obtained using `atomicAdd` on an `atomic` counter. As indicated above this unique address for vertex and normal is stored in a hash table, where the key is the unique index of the edge in the volume grid.

Finally, a second kernel will collect the quadrilaterals from the hash table and save the elements into an index buffer. Boundaries are easily handled by this kernel. A bucket in the hash table contains a quadrilateral if all entries are *valid* indices. If an entry in a bucket is an *invalid* index, the cell was a boundary cell. In this case, no quadrilateral is generated.

3.1.4 Vertex and face coloring

A fast and simple coloring is applied in parallel to vertices and quadrilaterals by using the uniform structure of the volume data. Vertex and quadrilateral colors are derived from cell colors. The color of a cell is defined by applying a bit pattern to its three dimensional index (i, j, k) ,

$$cv = (i \& 1) \mid (j \& 1) \ll 1 \mid (k \& 1) \ll 2.$$

This pattern results in 8 colors. We can transfer these colors onto the vertices of the quadrilateral mesh as they only have neighbors in adjacent cells.

From the cell coloring, we derive edge colors. Each cell colors its edges e_0, e_3, e_8 . By giving unique colors to the local edges incident to v_0 , i.e., $e_0 = 0, e_3 = 1, e_8 = 2$, we get an edge coloring with 24 values,

$$ce = 3 * cv + e,$$

where cv is the cell color and $e \in \{0, 1, 2\}$ are the local edge colors. As each cell edge corresponds to maximum one quadrilateral, we can transfer the edge colors onto the quadrilaterals. In a second step, we reduce the number of

quadrilateral colors to 5 by adding a second kernel which processes face-wise. It iterates through all quadrilaterals with colors greater or equal 5 and assigns a new color between 0 and 4 by checking which of these colors does not yet appear in its neighborhood.

Fig. 7 Non-manifold edge connecting four edges. Faces are oriented according to the normal direction. Thus, there are two possible ways to consistently connect faces with the same orientation

3.2 Halfedge data structure

The halfedge data structure is computed using two kernels. In the first kernel, each thread processes a quadrilateral and collects the *local* information required by the data structure. For each vertex, we store the halfedge which starts at the vertex, and for the face the first halfedge is stored. For the four halfedges in a quadrilateral, we store the index of the vertex at which the halfedge starts, the face, and the index of the next halfedge. This kernel saves the indices of halfedges which belong to the same edge in a hash table. The key is constructed using the indices of the incident vertices and saved in a 64-bit unsigned long long int. The smaller index is saved in the first 32 bits, the larger index in the second 32 bits. The hash table also contains a counter for the number of halfedges stored in each bucket. Distinct threads processing the same edge will compute the same key. Thus, each thread stores its halfedge in the same bucket of the hash table. The exact storage position within the bucket is computed using the halfedge counter which is increased with `atomicAdd`.

Global information is collected by a second kernel which processes the entries of the hash table and connects *twin* edges. If an edge contains only one halfedge, it is a boundary edge. If an edge contains two halfedges, we set each other as twin. Due to tunnels between two cells, it might be that an edge contains four halfedges, i.e., it is non-manifold (Fig. 7). In that case, we set the twins in a way that the data structure is consistent. We achieve this by connecting halfedges which

point in opposite directions such that faces have the same orientation.

4 Mesh simplification

Due to the transitions between layers in the volume data, the DMC mesh has numerous quadrilaterals with the valence pattern 3-X-3-Y, where $X, Y \geq 5$, i.e., two non-consecutive vertices with valence 3 and the other two vertices with valence equal to or larger than 5 (Fig. 8a); and quadrilaterals with the valence pattern 3-3-3-3 (Fig. 8b). These elements can be easily removed from the mesh as follows. For the case 8a, vertices in red are merged into a new vertex and the red element is removed. For the case 8b, edges in red are collapsed moving vertices in red toward vertices in blue. The red elements are removed. For the configuration 8c, no element can be removed in order to keep the mesh manifold.

These patterns are well known in the subject of quad mesh post-processing [1,3,35,41]. The novelty here is that we perform simplification efficiently in parallel using the face coloring which is implicit to the dual marching cubes method. One should keep in mind that simplification improves mesh quality and repositions vertices. Mesh simplification is optional and can be turned off.

4.1 Pattern 3-X-3-Y

We use the face coloring scheme presented in Sect. 3.1.4 to avoid race conditions. Elements are removed from the mesh with the following CUDA kernels:

1. Compute vertex valence. For this purpose, a kernel iterates through the halfedges and increases the valence of its vertex by using `atomicAdd()`.
2. Mark elements with the valence pattern if no neighboring element has a lower color id than the element itself.
3. Remove marked elements by merging vertices with valence three into a new vertex, in Fig. 8a.
4. Find remaining elements with the valence pattern. There is a buffer with an `int` entry for each vertex, a `counter`, which counts how often quadrilaterals with this valence pattern share a vertex with valence three. This way neighbor elements with the same valence pattern are identified easily.
5. Remove elements if no neighboring element has the same valence pattern, i.e., none of the valence three vertices is shared.
6. Remove vertices. This kernel works vertex wise. It copies vertices to a new vertex buffer, if they are not marked for removal, and it maps old to new vertex indices.
7. Remove quadrilaterals. This kernel works elementwise and copies quadrilaterals to a new element buffer, if they are not marked for removal. It uses index mapping from

Fig. 8 Valence pattern 3-X-3-Y, where $X, Y \geq 5$ and 3-3-3-3. For the case **a** on the left vertices in red are collapsed into a new vertex, removing the red element. For the case **b** in the middle the edges in red are

collapsed moving the vertices in red toward the vertices in blue. The red elements are removed. For the configuration on the right no element will be removed in order to keep the mesh manifold

(a) DMC mesh of a human skull, iso-value 900.1

(b) DMC mesh of a human body (respiratory tract), iso-value 700

(c) DMC mesh human skeleton, iso-value 1200

Fig. 9 DMC meshes of different CTs

the previous kernel to connect vertices which form a quadrilateral.

8. Re-build halfedge data structure.

4.2 Pattern 3-3-3-3

There are two configurations for which elements with this valence pattern cannot be removed from the mesh: two neighbor elements have the same valence pattern, e.g., they are the faces of a hexahedron; or the elements are faces of a configuration as shown in Fig. 8c. In the latter case, two quadrilaterals sharing the same four vertices would remain, i.e., the mesh would be non-manifold. The following processing steps were implemented to remove elements with the valence pattern 3-3-3-3:

1. Compute vertex valence. Similar to the kernel presented in Sect. 4.1.
2. Mark vertices and elements for removal. This kernel processes the mesh elementwise. For each element, it tests if

all the vertices have valence three. In this case, the complete neighborhood is reconstructed. The kernel checks if the element has a neighbor with the same valence pattern or if it is the case shown in Fig. 8c. If not, the element and its four neighbors (Fig. 8b) are marked for removal.

3. Remove vertices. Similar to Sect. 4.1.
4. Remove elements. Similar to Sect. 4.1.
5. Re-compute halfedge data structure.

The algorithm requires four kernels to remove elements from the mesh. We remark that neighborhood is reconstructed using the halfedge data structure.

5 Results

We evaluate performance and quality of the parallel DMC and simplification algorithms presented in this work and compare them with several other reconstruction methods.

Fig. 10 Synthetic data sets described in Eqs. (5) and (6)

We compute the iso-surface from two CT data sets, a human *skull* of size $512^2 \times 641$ shown in Fig. 9a and a human torso of size $512^2 \times 743$ for which iso-surfaces were extracted with different iso-values. For the iso-value 700.1 (Fig. 9b), we call the data *body*, and for the iso-value 1200.1 *skeleton* (Fig. 9c). Furthermore, we generated synthetic data representing a triple periodic level set called *iWP*, which is the zero surface of the scalar function:

$$f(x, y, z) = \cos x \cos y + \cos y \cos z + \cos z \cos x - \cos x \cos y \cos z \quad (5)$$

and visualize five periods in all three directions. The scalar function was discretized on a grid of size 512^3 (Fig. 10a). Another synthetic data are *gen2* [46], shown in Fig. 10b,

$$f(x, y, z) = 2y(y^2 - 3x^2)(1 - z^2) + (x^2 + y^2)^2 - (9z^2 - 1)(1 - z^2). \quad (6)$$

We use the iso-value $t_0 = 0$ and evaluate it on grids of size 64^3 , 128^3 , and 256^3 . Additionally, we use qualitatively different data sets to prove robustness. The experiments were carried out on a desktop computer with an Intel Core i7-8700 with 32 GB memory and a NVIDIA GeForce RTX 2080 Ti 11GB.

We compare DMC to three other methods: the original marching cubes (MC) [25], a topologically correct marching cubes (TMC) which uses the asymptotic decider and also reconstructs tunnels within cells [18], and dual contouring (DC) [20]. DC was designed for generating hybrid meshes containing quadrilaterals and triangles and works on octrees. By restricting DC to uniform voxel grids, it generates pure quadrilateral meshes similar to DMC but with different vertex positioning and handling of cells with multiple branches. The connectivity of vertices is determined by using the *SurfaceNets* method described in [15]. First, we analyze the performance of the parallel DMC algorithm and present the effects of mesh simplification. Afterward, we compare the surface reconstruction methods in terms of quality.

Table 2 Processing time in milliseconds to extract an iso-surface with the DMC and MC algorithms

	Skull	Body	Skeleton	iWP	gen2_256
DMC	68	60	84	99	7
MC	31	36	42	65	2

5.1 Performance of parallel DMC

The performance of the parallel DMC is evaluated by measuring computation time and number of vertices and quadrilaterals generated. We compare the results with the corresponding meshes obtained with the MC algorithm. TMC and DC were both implemented on CPU and therefore are not included in the performance comparison.

Performance of DMC in comparison with MC is given in Table 2. Times are given in milliseconds. Both algorithms run on GPU and for comparison both generate a shared vertex data structure. Table 3 gives the number of vertices and elements generated by DMC compared to MC, where DMC generates a pure quadrilateral mesh and MC a triangle mesh. DMC generates slightly less vertices and correspondingly less elements than MC. DMC is slower than MC, but it still only requires approximately 100 milliseconds to reconstruct surfaces from large data sets.

We demonstrate robustness by computing iso-surfaces from numerous data sets. Table 9 gives numbers of elements and times in milliseconds. DMC computed valid iso-surfaces for all examples. Furthermore, the vertex positioning worked in all cases shown in Fig. 16.

5.2 Mesh simplification

Simplification algorithms are evaluated in terms of computational performance and the number of vertices and elements being eliminated. The histograms presented in Fig. 11 show that the elimination of elements with valence patterns 3-X-3-Y, $X, Y \geq 5$, and consecutively 3-3-3-3 considerably reduce the number of irregular vertices.

Simplifying elements with pattern 3-X-3-Y eliminates one vertex and element. Pattern 3-3-3-3 simplification eliminates four vertices and elements. The effect of the pattern 3-X-3-Y simplification is shown in Fig. 12, and the result of the simplification of elements with the pattern 3-3-3-3 in Fig. 13. Table 4 gives runtime and total number of elements removed from the original DMC mesh after each step and the number of elements removed from the mesh in total after the two mesh simplification steps. The element simplification reduces the number of elements for the human skull data set in about 10%. A similar result is achieved for the skeleton data set. The human body and the iWP level set data are smoother and therefore do not have so

Table 3 Number of elements generated with the DMC and MC algorithms

	Vert. DMC	Vert. MC	Quads DMC	Tris MC
Skull	5,054,418	5,074,557	5,074,547	10,243,456
Body	4,248,072	4,250,819	4,233,291	8,470,072
Skeleton	6,291,530	6,290,033	6,276,513	12,554,264
iWP	6,528,938	6,558,756	6,498,576	13,056,698
gen2_256	158,902	158,904	158,904	317,812

The head data set has $512^2 \times 641$ cells. The data set used to generate the torso and body surfaces has $512^2 \times 743$ cells, the synthetic data set iWP has 512^3 cells, and gen2 has 256^3 cells

(a) Valence distribution before simplification (b) Valence distribution after simplification of vertex valence pattern 3-X-3-Y (c) Valence distribution after simplification of vertex valence pattern 3-3-3-3

Fig. 11 Distribution of the vertex valences in the DMC mesh for the human skull before simplification in blue, after simplifying elements with vertex valence pattern 3-X-3-Y in red, and after simplifying elements with valence pattern 3-3-3-3 in cyan

Fig. 12 DMC mesh after removing elements with vertex valence pattern 3-X-3-Y

Fig. 13 DMC mesh after removing elements with vertex valence pattern 3-3-3-3

many elements with the vertex valence pattern 3-3-3-3. In these cases, the simplification reduces the total number of elements by 5%. The mesh simplification based on face coloring reduces much more elements than the method presented in [19]. For the skull surface, a total of 352,337 elements with the vertex valence pattern 3-X-3-Y were simplified. With the previous method, only 167,537 could be eliminated from the mesh. We remark that the halfedge data structure has to be recomputed after each mesh simplification step.

The times required for the computation of the halfedge data structure and the element coloring are given in Table 5. Quadrilaterals are colored using five different colors. The face coloring is used by the simplification of elements with the vertex valence pattern 3-X-3-Y. The computation of the halfedge data structure requires many global memory access on the GPU which is very time consuming.

5.3 Surface reconstruction quality

We evaluate the surface reconstruction methods using four different criteria: deviation from the iso-value, element quality, amount of irregular vertices, and amount of non-manifold vertices. In this section, DMC corresponds to dual marching cubes without simplification and DMCS to dual marching cubes with simplification. For comparing DMC and DC to

Table 4 Simplification of elements with valence pattern 3-X-3-Y and 3-3-3-3. Times are given in milliseconds

	Time (ms)	Number of removed quads		
		3-X-3-Y	3-3-3-3	Total
Skull	13	352,335	189,408	541,743
Body	11	335,116	18,304	353,420
Skeleton	17	509,630	165,092	674,722
iWP	15	346,907	384	347,291
gen2_256	1	16,158	0	16,158

The columns give the number of removed quadrilaterals for each simplification method

MC and TMC, quadrilaterals were subdivided into two triangles using the MaxMin angle criterion. The deviation from the iso-value is measured at the vertices. We use Eq. (2) for computing the value at vertices of medical data sets. On synthetic data sets we use the input functions, i.e., Eqs. (5) and (6). The deviation for a vertex v is computed as

$$\Delta f(v) = |t_0 - f(v)|. \quad (7)$$

Quality of triangles is computed using the mean ratio metric,

$$q_{\text{mtri}} = 4\sqrt{3} \frac{A}{\sum_{i=1}^3 l_i^2}, \quad (8)$$

where A is the area of the triangle, and l_i is the length of their incident edges [13,33]. For quadrilaterals we use the shape quality metric of Stimpson et al. [39],

$$q_{\text{quad}} = \min_{i \in [1,4]} \frac{2A_i}{l_i^2 + l_{i-1}^2}, \quad (9)$$

where A_i is the area of the parallelogram defined by the two incident edges of a vertex v_i , with lengths l_i and l_{i-1} . In a triangle mesh, vertices with a valence unequal to 6, and in a quad mesh, vertices with a valence unequal to 4 are considered as irregular.

Table 6 shows that DMC, TMC, and MC have no deviation from the iso-value for medical data sets. This behavior is expected as all these methods place their vertices exactly on the trilinear interpolant. DMCS has some deviation which is caused by the repositioning after the 3-X-3-Y pattern simplification. The deviation is still a

Table 5 Computation times of the halfedge data structure and face coloring in milliseconds

	Skull	Body	Skeleton	iWP	gen2_256
Halfedge	58	48	71	73	2
Coloring	15	13	17	18	2

Fig. 14 Deviation from the iso-value for DC, DMC, and DMCS on *iWP*

magnitude lower than the one of DC. Figure 14 shows $\Delta f(v)$ for the data set *iWP* where vertices are sorted according to their deviation from the iso-value. DMC has its vertices closest to the iso-surface. Its deviation originates from the approximation error of the trilinear interpolant as DMC positions vertices exactly on it. The 3-X-3-Y pattern simplification repositions vertices and thus increases the deviation from the iso-value for these vertices. MC and TMC place their vertices also on the trilinear interpolant but along the voxel edges not within the voxel cells. Therefore, the deviation is similar to DMC but not exactly the same. DC uses quadratic error functions to compute its vertex positions. Therefore, vertices are in general not positioned on the iso-surface. For more smooth surfaces like gen2 all methods behave very similar.

Table 6 Average deviation from the iso-value for different surface reconstruction methods

	Average $\Delta f(v)$				
	DMC	DMCS	DC	TMC	MC
Skull	0.00	4.34	39.09	0.00	0.00
Body	0.00	0.86	5.26	0.00	0.00
Skeleton	0.00	0.74	4.41	0.00	0.00
iWP	0.97	1.06	3.44	1.04	1.04
gen2_64	0.69	0.69	0.68	0.69	0.69
gen2_128	0.70	0.69	0.70	0.70	0.70
gen2_256	0.70	0.69	0.70	0.70	0.70

For *iWP* all values were multiplied by 10^4 . The best values are highlighted in bold

Fig. 15 Quality of quadrilaterals generated by DC and DMC with and without simplification on *skull*

Element quality and number of irregular vertices are stated in Table 7 for triangular and in Table 8 for quadrilateral meshes. We observe an interesting effect for the element quality when simplifying DMC. While simplification improves the quality for quads, it is reduced for triangles. The reason is that the 3-X-3-Y pattern can often be triangulated better than its simplified version, especially in cases like in Fig. 12. The same holds for the number of irregular vertices. It is improved for quadrilaterals but not for triangular meshes. If a triangular mesh is preferred, the 3-X-3-Y pattern simplification should not be applied. In general, the dual methods deliver higher-quality triangles than TMC and MC. DMC has the best triangles most of the time and DMCS has the best quadrilaterals. It is expectable that the suggested simplifications are not suited for triangle meshes as they were explicitly designed to improve the quadrilateral structure. Figure 15 compares the shape quality of quadrilaterals generated by DC, DMC and DMCS for the data set *skull*. DC delivers slightly less quality than DMC. DMCS

Table 7 Quality comparison of surface reconstruction methods generating triangular meshes average

	Average $q_{m_{tri}}$					# Irregular vertices [%]				
	DMC	DMCS	DC	TMC	MC	DMC	DMCS	DC	TMC	MC
Skull	0.83	0.82	0.82	0.69	0.69	53.97	60.10	56.66	59.41	60.17
Body	0.81	0.78	0.80	0.67	0.67	47.62	51.35	51.74	45.03	45.11
Skeleton	0.83	0.81	0.81	0.67	0.67	53.33	59.03	55.71	60.00	60.55
iWP	0.84	0.82	0.86	0.65	0.65	23.85	28.57	24.13	25.17	25.13
gen2_64	0.87	0.83	0.84	0.71	0.71	34.88	48.30	43.37	46.15	46.15
gen2_128	0.87	0.82	0.84	0.71	0.71	31.86	44.48	36.71	45.03	45.03
gen2_256	0.87	0.82	0.84	0.71	0.71	28.96	41.01	34.40	44.13	44.13

For the element quality, values are averaged. The number of irregular and non-manifold vertices is stated relative in percent. The best values are highlighted in bold

Table 8 Quality comparison of surface reconstruction methods generating quadrilateral meshes

	Average $q_{m_{quad}}$			# Irregular vertices (%)			# Non-manifold vertices (%)		
	DMC	DMCS	DC	DMC	DMCS	DC	DMC	DMCS	DC
Skull	0.79	0.81	0.72	57.94	31.96	57.90	0.84	0.94	5.61
Body	0.80	0.81	0.71	40.70	11.76	40.65	0.03	0.03	0.20
Skeleton	0.77	0.80	0.68	58.03	28.80	57.91	0.26	0.29	2.15
iWP	0.89	0.90	0.84	23.51	4.22	23.50	0	0	0.02
gen2_64	0.85	0.87	0.73	43.37	8.85	43.37	0	0	0
gen2_128	0.86	0.87	0.73	42.60	5.95	42.60	0	0	0
gen2_256	0.86	0.87	0.74	42.28	5.14	42.28	0	0	0

For the deviation from the iso-value and the element quality, values are averaged. The number of irregular and non-manifold vertices is stated relative in percent. The best values are highlighted in bold

Table 9 Set of qualitatively differing data sets for evaluation of DMC

	Source	Type/modality	Size	Vert. DMC	Quads DMC	Time (ms)
Thorax angiography	[40]	MR angiography	$384 \times 512 \times 80$	420,109	407,904	5
Baby head	[40]	CT	$256 \times 256 \times 98$	175,100	174,435	3
Pelvic bones	[40]	CT	$512 \times 512 \times 1047$	3,361,127	3,353,352	92
Bruce Gooch's Brain	[40]	FMRI	$256 \times 256 \times 156$	1,192,334	1,164,349	13
Arterial cardix	Non-public	CT	$512 \times 512 \times 452$	2,409,280	2,401,573	54
Carp	[40]	CT	$256 \times 256 \times 512$	446,420	446,310	12
Arterial cenovix	Non-public	CT	$512 \times 512 \times 361$	1,313,523	1,304,892	28
Abdomen	[40]	CT	$512 \times 512 \times 147$	1,378,959	1,363,737	15
Male head (CT)	[40]	CT	$256 \times 256 \times 113$	616,113	611,189	7
Engine	[40]	CT	$256 \times 256 \times 256$	310,996	311,040	8
Knee	[40]	MRI	$512 \times 512 \times 87$	3,467,679	3,433,967	26
Kummer surface	[44]	Synth	$256 \times 256 \times 256$	265,752	263,856	5
Male head (MRI)	[40]	MRI	$256 \times 256 \times 256$	1,979,810	1,962,482	20
Female head	[40]	MRI	$256 \times 256 \times 109$	732,292	724,980	7
Neovius surface	[43]	Synth	$256 \times 256 \times 256$	644,956	640,992	9
Porsche	[40]	CT	$559 \times 1023 \times 347$	1,420,200	1,420,238	62
Steiner surface	[45]	Synth	$512 \times 512 \times 512$	1,232,672	1,232,184	34
Thorax	[40]	MRI	$320 \times 320 \times 72$	230,574	226,518	3
Thorax	Non-public	CT	$512 \times 512 \times 497$	2,358,976	2,351,819	47
Tomato	[40]	MRI	$256 \times 256 \times 64$	186,476	182,594	3
Visible Male	[40]	CT	$128 \times 256 \times 256$	290,968	290,552	5

has higher-quality elements. Especially the 3-3-3-3 pattern contains low-quality elements which are eliminated by simplification.

In Sect. 3, it was mentioned that DMC generates non-manifold edges when there is a tunnel in between two cells. This also causes the generation of non-manifold vertices stated in Table 8. DMCS has a higher percentage of non-manifold vertices as it has overall less vertices. The absolute number of irregular vertices is the same for DMC and DMCS. DC has another topological issue. Whenever a cell is intersected by more than one branch, a non-manifold vertex is generated. Thus, the amount of non-manifold vertices is one order of magnitude higher than for DMC and DMCS. In Table 7, the amount of non-manifold vertices is not stated because for DMC, DMCS, and DC the numbers are exactly the same as in Table 8. The marching cubes methods MC and TMC do not generate non-manifold vertices.

6 Conclusions

We presented a parallel implementation of the DMC algorithm which efficiently processes volume data consisting of hundreds of millions of voxels. The DMC meshes are topologically consistent across cell borders. This is due to the

fact that we compute the intersection of the iso-surface with the cells using the asymptotic decider to solve ambiguities. The output of the algorithm is a quadrilateral mesh stored in a halfedge data structure. We use the fact that the data are already on GPU and perform some mesh simplification by eliminating vertices and quadrilaterals with vertex valence pattern 3-X-3-Y, $X, Y \geq 5$ and 3-3-3-3. Mesh simplification is based on a face coloring, using 5 colors which is also implemented on the GPU. This way, vertices in the DMC mesh have a better valence distribution than meshes generated by DC. The DMC algorithm generates meshes with better triangle quality than marching cubes and better quadrilateral quality than DC. Nevertheless, the DMC algorithm is not able to generate topologically correct surface as defined by Eq. (1). The case of a tunnel in a cell or a tunnel across two neighbor cells, shown in Fig. 4, cannot be solved by the DMC algorithm.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Fig. 16 Reconstructed surfaces from synthetic, CT, and MRI data sets listed in Table 9

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. A Regularization Approach for Automatic Quad Mesh Generation. Zenodo (2020). <https://doi.org/10.5281/zenodo.3653416>
2. Attali, D., Cohen-Steiner, D., Edelsbrunner, H.: Extraction and simplification of ISO-surfaces in tandem. In: Proceedings of the 3rd Eurographics Symposium on Geometry Processing, SGP'05. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2005)
3. Canann, S.A., Muthukrishnan, S., Phillips, R.: Topological improvement procedures for quadrilateral finite element meshes. *Eng. Comput.* **14**(2), 168–177 (1998)
4. Chen, J., Jin, X., Deng, Z.: GPU-based polygonization and optimization for implicit surfaces. *Vis. Comput.* **31**(2), 119–130 (2015)
5. Chernyaev, E.V.: Marching cubes 33: construction of topologically correct isosurfaces. Technical Report (1995)
6. Cignoni, P., Ganovelli, F., Montani, C., Scopigno, R.: Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Comput. Graph.* **24**, 399–418 (2000)
7. Custodio, L., Etienne, T., Pesco, S., Silva, C.: Practical considerations on marching cubes 33 topological correctness. *Comput. Graph.* **37**(7), 840–850 (2013)
8. de Bruin, P.W., Vos, F.M., Post, F.H., Frisken-Gibson, S.F., Vossepoel, A.M.: Improving triangle mesh quality with surfacenets. In: Medical Image Computing and Computer-Assisted Intervention—MICCAI 2000, pp. 804–813. Springer, Berlin (2000)
9. D'Agostino, D., Seinstra, F.J.: A parallel isosurface extraction component for visualization pipelines executing on GPU clusters. *J. Comput. Appl. Math.* **273**(C), 383–393 (2015)
10. Dupuy, G., Jobard, B., Guillon, S., Keskes, N., Komatitsch, D.: Parallel extraction and simplification of large isosurfaces using an extended tandem algorithm. *Comput. Aided Des.* **42**(2), 129–138 (2010)
11. Dürst, M.J.: Re: additional reference to “marching cubes”. *SIGGRAPH Comput. Graph.* **22**(5), 243 (1988)
12. Etienne, T., Nonato, L.G., Scheidegger, C., Tienry, J., Peters, T.J., Pascucci, V., Kirby, R.M., Silva, C.T.: Topology verification for isosurface extraction. *IEEE Trans. Vis. Comput. Graph.* **18**(6), 952–965 (2012)
13. Freitag, L., Jones, M., Plassmann, P.: A parallel algorithm for mesh smoothing. *SIAM J. Sci. Comput.* **20**(6), 2023–2040 (1999)
14. Gibson, S.F.F.: Constrained elastic surface nets: generating smooth surfaces from binary segmented data. In: Medical Image Computing and Computer-Assisted Intervention—MICCAI'98, pp. 888–898. Springer, Berlin (1998)
15. Gibson, S.F.F.: Using distance maps for accurate surface representation in sampled volumes. In: IEEE Symposium on Volume Visualization (Cat. No. 989EX300), pp. 23–30. IEEE (1998)
16. Grosso, R.: Construction of topologically correct and manifold isosurfaces. *Comput. Graph. Forum* **35**(5), 187–196 (2016)
17. Grosso, R.: TMC (2016). <https://github.com/rogrosso/tmc>
18. Grosso, R.: An asymptotic decider for robust and topologically correct triangulation of isosurfaces. In: Proceedings of the Computer Graphics International Conference, CGI'17, pp. 39:1–39:5. ACM, New York (2017)
19. Grosso, R., Zint, D.: Parallel reconstruction of quad only meshes from volume data. In: Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications—Volume 1: GRAPP, pp. 102–112. INSTICC, SciTePress (2020)
20. Ju, T., Losasso, F., Schaefer, S., Warren, J.: Dual contouring of hermite data. In: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2002), pp. 339–346. ACM Press (2002)
21. Kazhdan, M., Klein, A., Dalal, K., Hoppe, H.: Unconstrained isosurface extraction on arbitrary octrees. In: Proceedings of the 5th Eurographics Symposium on Geometry Processing, pp. 125–133 (2007)
22. Lewiner, T., Lopes, H., Vieira, A.W., Tavares, G.: Efficient implementation of marching cubes' cases with topological guarantees. *J. Graph. Tools* **8**(2), 1–15 (2003)
23. Löffler, F., Schumann, H.: Generating smooth high-quality isosurfaces for interactive modeling and visualization of complex terrains. In: VMV (2012)
24. Lopes, A., Brodlie, K.: Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Trans. Vis. Comput. Graph.* **9**, 2003 (2003)
25. Lorensen, W.E., Cline, H.E.: Marching cubes: a high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.* **21**(4), 163–169 (1987)
26. Matveyev, S.V.: Marching cubes: surface complexity measure. In: Proceedings of SPIE—The International Society for Optical Engineering, vol. 3643, pp. 220–225 (1999)
27. Montani, C., Scateni, R., Scopigno, R.: A modified look-up table for implicit disambiguation of marching cubes. *Vis. Comput.* **10**(6), 353–355 (1994)
28. Natarajan, B.K.: On generating topologically consistent isosurfaces from uniform samples. *Vis. Comput.* **11**(1), 52–62 (1994)
29. Nielson, G.M.: On marching cubes. *IEEE Trans. Vis. Comput. Graph.* **9**(3), 283–297 (2003)
30. Nielson, G.M.: Dual marching cubes. In: Proceedings of the Conference on Visualization'04, VIS'04, pp. 489–496. IEEE Computer Society, Washington, DC (2004)
31. Nielson, G.M., Hamann, B.: The asymptotic decider: resolving the ambiguity in marching cubes. In: Proceedings of the 2nd Conference on Visualization'91, VIS'91, pp. 83–91. IEEE Computer Society Press, Los Alamitos (1991)
32. Pasko, A., Pilyugin, V., Pokrovskiy, V.: Geometric modeling in the analysis of trivariate functions. *Comput. Graph.* **12**(3), 457–465 (1988)
33. Rangarajan, R., Lew, A.J.: Provably robust directional vertex relaxation for geometric mesh optimization. *SIAM J. Sci. Comput.* **39**(6), A2438–A2471 (2017)
34. Rashid, T., Sultana, S., Audette, M.A.: Watertight and 2-manifold surface meshes using dual contouring with tetrahedral decomposition of grid cubes. *Proc. Eng.* **163**, 136–148 (2016)
35. Remacle, J.F., Lambrechts, J., Seny, B., Marchandise, E., Johnen, A., Geuzaine, C.: Blossom-quad: a non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. *Int. J. Numer. Methods Eng.* **89**(9), 1102–1119 (2012)
36. Renbo, X., Weijun, L., Yuechao, W.: A robust and topological correct marching cube algorithm without look-up table. In: Proceedings of the the 5th International Conference on Computer and Information Technology, CIT'05, pp. 565–569. IEEE Computer Society, Washington, DC (2005)

37. Schaefer, S., Ju, T., Warren, J.: Manifold dual contouring. *IEEE Trans. Vis. Comput. Graph.* **13**(3), 610–619 (2007)
38. Schaefer, S., Warren, J.: Dual marching cubes: primal contouring of dual grids. In: 2004. PG 2004. 12th Pacific Conference on Computer Graphics and Applications, pp. 70–76. IEEE Computer Society (2004)
39. Stimpson, C., Ernst, C., Knupp, P., Pébay, P., Thompson, D.: The verdict library reference manual. Sandia National Laboratories Technical Report, vol. 9(6) (2007)
40. The Volume Library. <http://schorsch.efi.fh-nuernberg.de/data/volume/>
41. Tarini, M., Pietroni, N., Cignoni, P., Panozzo, D., Puppo, E.: Practical quad mesh simplification. In: *Computer Graphics Forum*, vol. 29, pp. 407–418. Wiley Online Library (2010)
42. Ulrich, C., Grund, N., Derzapf, E., Lobachev, O., Guthe, M.: Parallel ISO-surface extraction and simplification. In: *WSCG 2014: Communication Papers Proceedings*, pp. 361–368. Vaclav Skala–Union Agency, Plzen (2014)
43. Wikipedia Contributors: Neovius surface—Wikipedia, the free encyclopedia (2017) (Online). https://en.wikipedia.org/w/index.php?title=Neovius_surface&oldid=808117164. Accessed 25 Mar. 2021
44. Wikipedia Contributors: Kummer surface—Wikipedia, the free encyclopedia (2020) (Online). https://en.wikipedia.org/w/index.php?title=Kummer_surface&oldid=993123659. Accessed 25 Mar. 2021
45. Wikipedia Contributors: Roman surface—Wikipedia, the Free Encyclopedia (2020) (Online). https://en.wikipedia.org/w/index.php?title=Roman_surface&oldid=975063355. Accessed 25 Mar. 2021
46. Wikipedia Contributors: Implicit Surface—Wikipedia, the Free Encyclopedia (2021) (Online). https://en.wikipedia.org/w/index.php?title=Implicit_surface&oldid=1001234443. Accessed 25 Jan. 2021

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Roberto Grosso is teaching and research assistant at the computer graphics group at the Friedrich-Alexander-University Erlangen-Nuremberg. He got his Diploma in Physics at the Universidad Nacional de Córdoba, Argentina, and his Ph.D. in Theoretical Physics at the Friedrich-Schiller-Universität Jena. His research interests are mesh generation, geometric modeling, and geometric processing.

Daniel Zint is currently a Ph.D. student in the computer graphics group of Prof. Dr. Marc Stamminger at the Friedrich-Alexander-University Erlangen-Nuremberg. He got his Bachelor's degree in Mechatronics and his Master's degree in Computational Engineering. His research interests are mesh generation, geometric modeling, and geometric processing.