Proof of Correctness of a Marching Cubes Algorithm Carried out with Coq

Andrey N Chernikov and Jing Xu

Department of Computer Science
Old Dominion University
{*achernik,jxu*}*@cs.odu.edu*

Abstract

The Marching Cubes algorithm is a well known and widely used approach for extracting a triangulated isosurface from a three-dimensional rectilinear grid of uniformly sampled data values. The algorithm relies on a large manually constructed table which exhaustively enumerates all possible patterns in which the isovalue relates to the values in the nodes of a cubical cell of the grid. For each pattern the table contains the local connectivity of the triangles. The construction of this table is labor intensive and error prone. Indeed, the original paper allowed for topological holes in the surface. This problem was later fixed by several authors, however a proof of correctness to our knowledge was never presented. Another issue, the possibility of intersecting triangles inside a single cube, has not been addressed. In this paper we present our formal proof of correctness of a Marching Cubes implementation with respect to both of these conditions. Our proof is developed with the Coq proof assistant, and the script is available online.[1]

Keywords: Marching Cubes, Computer-aided proofs, Coq

[1] http://mc-proof.sourceforge.net

1. Introduction

Representation and visualization of three-dimensional isosurfaces is an important building block for a large number of applications in computational biology, bioinformatics, graphics, finite element simulations, and other areas [1, 2]. The most widely used algorithm for the construction of isosurfaces is called Marching Cubes (MC) which was proposed in 1987 by Lorensen and Cline [3]. For example, the MC algorithm is used for the reconstruction of contour surfaces in the popular Chimera software [4], see Figure 1. The algorithm relies on a large lookup table which defines a separate triangulation rule for each of the 256 cases that can arise at run-time. More specifically, it samples a given real-valued field (which is often represented through a discrete or continuous distance function) at regularly placed nodes, and triangulates the interior of each resulting cubical cell using a predefined table of triangle connectivity.

In Figure 2 we show an example of the application of a two-dimensional Marching Squares algorithm to a simple data grid. The table we used to create the segments of the isocontour was published previously [1] and is shown in Figure 3. Each cell has four corners, the value of the function in each corner can be either less or greater than the isovalue[2], therefore the

[2] In this analysis we treat the case of the sampled value being equal to the isovalue together with the case when it is greater than the isovalue. In an implementation this simplifying assumption can lead to some triangles being squeezed to an edge or a point. Such triangles can be easily pruned out by a post-processing step.

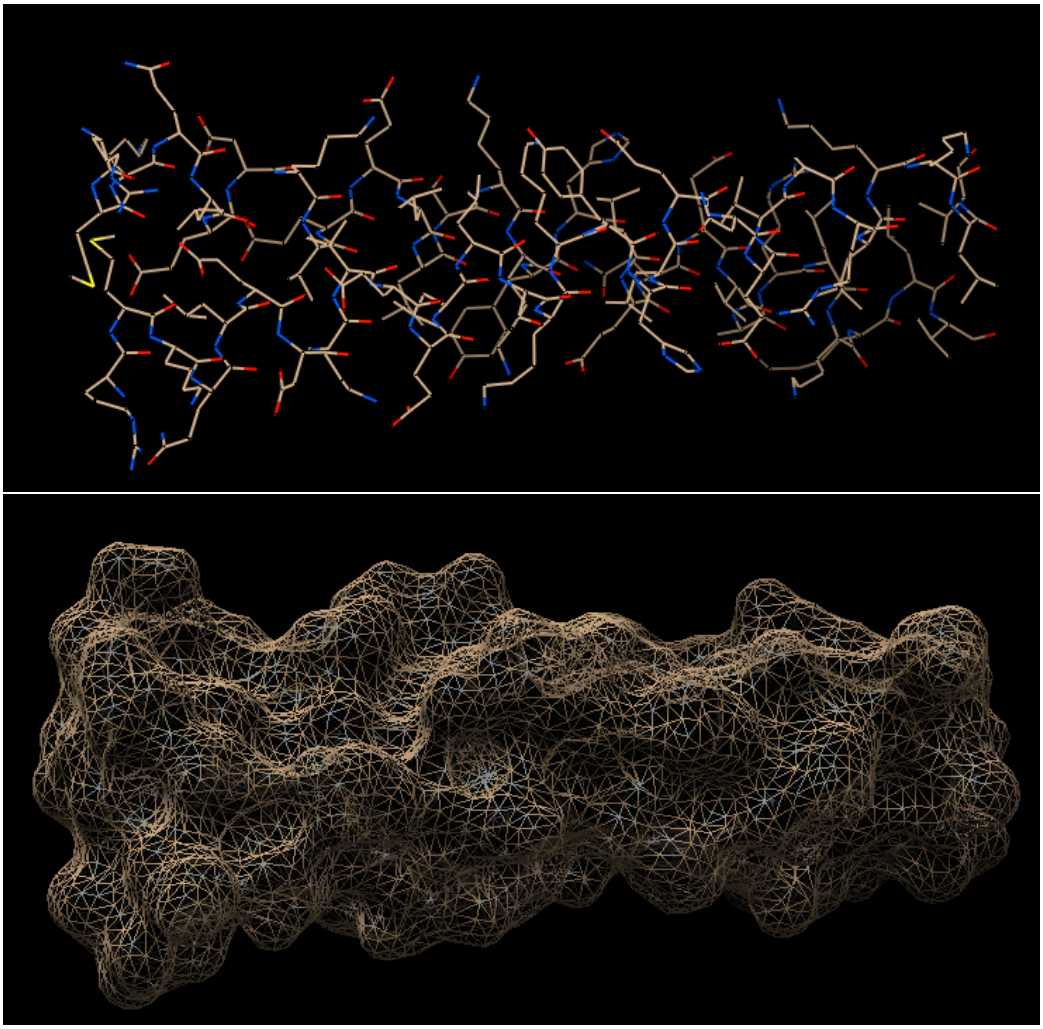


Figure 1: An example of a triangulated surface (bottom) obtained with the Chimera software [5] for the "1zik" structure (top) from the Protein Data Bank.

total number of distinct cases is $2^4 = 16$. In a three-dimensional grid each cube has eight corners, and the total number of cases is $2^8 = 256$.

The resemblance of the resulting isosurface to the true surface is usu-

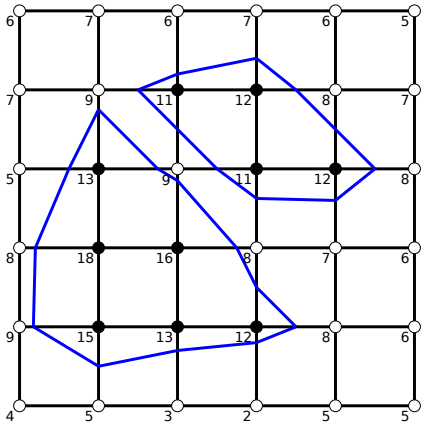


Figure 2: A two-dimensional example of the use of a Marching Squares algorithm. The resulting isocontour (blue) corresponds to an isovalue $\xi = 10$. The nodes of the rectilinear grid corresponding to values less than ξ are shown with white circles, and the nodes corresponding to values greater than ξ are shown with black circles.

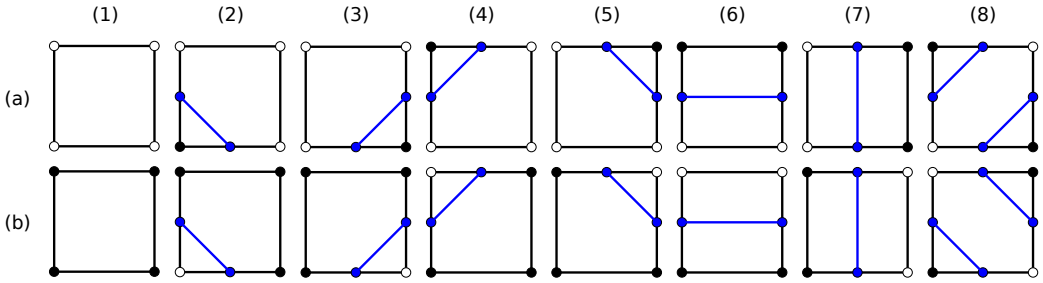


Figure 3: Two-dimensional rules for creating intersection vertices and intersection edges. Blue circles and segments show the intersection vertices and intersection edges created by the algorithm.

ally measured in terms of their geometric and/or topological proximity. The stronger properties of the true surface are known, the tighter proximity conditions can be proven. In the absence of any information of the true surface, a

minimal correctness requirement we can expect of a Marching Squares/Cubes algorithm is the following: all the nodes of the rectilinear grid with values less than ξ be separated by the resulting isocontours/isosurfaces from all the nodes with values greater than ξ. For the three-dimensional algorithm, we refine this requirement down to two components:

(i) *Two- and three-dimensional cohesion*, i.e., every axis-aligned plane of the three-dimensional rectilinear grid (passing through the nodes) contains zero or more isocontours from the three-dimensional Marching Cubes isosurfaces that are a correct output of a two-dimensional Marching Squares algorithm.

(ii) *Water-tightness*, i.e., the absence of holes. The output of the Marching Cubes algorithm consists of zero or more water-tight triangulated isosurfaces. We further decompose the requirement of water-tightness into two conditions:

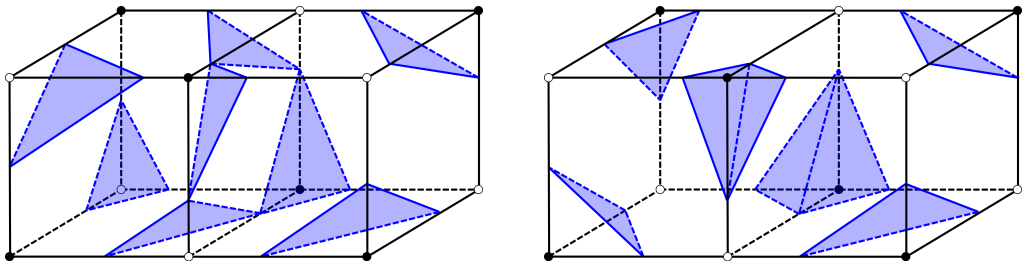
(ii-a) the resulting triangles are conforming along their edges, i.e., every edge of every triangle is incident upon exactly one other triangle, and

(ii-b) the triangles never intersect in their interiors, independently of the positions of their vertices which are computed at run time.

Requirement (i) ensures the separation in all planes of the grid, while requirement (ii) guarantees a hole-free three-dimensional surface which connects all of the two-dimensional contours.

The authors of the original paper [3] reduce the complexity of the algorithm through exploring two types of symmetry: grouping two cases with the opposite relations to the isovalue in all corners into one case, and also

grouping rotationally symmetric cases. Unfortunately, as it was later pointed out [6, 7, 8, 9, 10, 11], some symmetric cases cannot be treated as one case, as we show in Figure 4. The authors [6, 7, 8, 9, 11] state that they solved this problem, each by their own extension of the lookup table, however they do not provide any proofs. Another potential problem is the possibility that the triangles inside a single cube intersect each other as shown in Figure 5.

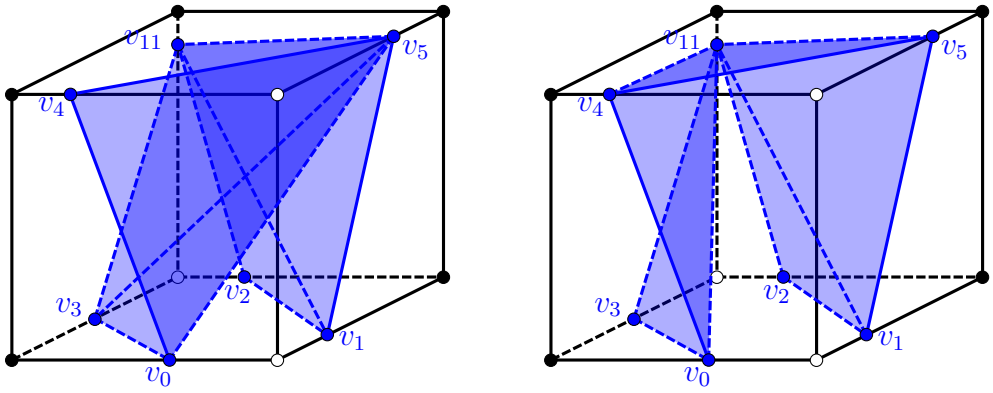


(a) The sets of triangles created by the application of the same triangulation pattern to both cubes form a hole in the surface at the shared face.

(b) The triangulations are consistent in the shared face.

Figure 4: Two cubes of a sampled grid sharing a common face. The corresponding corners of the cubes have pairwise opposite relations to the isovalue. Both cubes correspond to a single case (13) in the original paper [3] which combines these two cubes into one case due to symmetry.

Below we describe our Coq [13] proof written in a functional programming language Gallina based on a formal language Calculus of Inductive Constructions [14]. We examine two publicly available implementations of the MC algorithm [5, 12] that use equivalent lookup tables. Computer-assisted proofs in Coq have been used previously to support solutions of mesh generation



(a) A patch of a triangulated surface in which triangles intersect each other's interiors. The list of all triangles is $v_0v_5v_4$, $v_3v_5v_0$, $v_{11}v_5v_3$, $v_1v_5v_{11}$, and $v_2v_1v_{11}$. Edge v_3v_5 intersects the interior of triangle $v_2v_1v_{11}$, and edge $v_{11}v_1$ intersects the interior of triangle $v_3v_5v_0$. Therefore, the following pairs of triangles intersect: $v_3v_5v_0$ and $v_1v_5v_{11}$, $v_3v_5v_0$ and $v_2v_1v_{11}$, $v_{11}v_5v_3$ and $v_2v_1v_{11}$.

(b) A patch of a triangulated surface used in the implementation we study [12] in which triangles cannot intersect each other's interiors. The list of all triangles is $v_0v_4v_{11}$, $v_0v_{11}v_3$, $v_4v_5v_{11}$, $v_2v_{11}v_1$, and $v_5v_1v_{11}$.

Figure 5: An example of an intersection pattern and possible surface patches.

and other geometric problems. Dufourd and Bertot [15] presented a proof of correctness of a planar Delaunay triangulation algorithm. Gonthier proved the Four-Color Theorem [16]. Dufourd [17] developed a hypermap framework for computer-aided proofs in surface subdivisions. He uses this framework to prove the genus theorem and the Euler's formula as its corollary. Brun et al. [18] designed a two-dimensional convex hull algorithm based on hypermaps and proved its correctness. Magaud et al. [19] formalized a proof of the

Desargues theorem in Coq. Dehlinger and Dufourd [20] used Coq to prove a combinatorial part of the Surface Classification Theorem. A computer-assisted proof of dihedral angle bounds for a three-dimensional tetrahedral meshing algorithm was performed by Labelle and Shewchuk [21], although the programming language was not specified.

Our proof consists of two major parts. The first is combinatorial in its approach, and it establishes the truth of requirements (i) and (ii-a) above. This first part appeared in our preliminary results [22]. The second part is more geometric, and it establishes the truth of requirement (ii-b) above. After we briefly review the classical MC algorithm in Section 2, we describe both parts of our proof in Sections 3 and 4, respectively. Section 5 concludes the paper.

2. Classical Algorithm

The main steps of the classical Marching Cubes algorithm are shown in pseudocode in Figure 6. The function $\mathit{CaseTable\_Get}(index)$ queries a manually constructed table with a key composed of eight bits, each bit corresponding to the result of the test, $F(x) \geq \xi$ or $F(x) < \xi$, in one of the eight corners of cube b.

3. Proof of Combinatorial Correctness

In our entire development we work with a single unit cube which represents an arbitrary cube of the sampled grid. We call it the *generic cube* because our proofs are valid for any combination of the sampled values in the

Algorithm MarchingCubes(G, F, ξ)

Input: A rectilinear grid of nodes $G \subset \mathbb{R}^3$ along with a mapping $F : G \to \mathbb{R}$, and an isovalue $\xi \in \mathbb{R}$

Output: A triangular surface M embedded in $\mathbb{R}^3$ that interpolates the set $\{x \in \mathbb{R}^3 \mid F(x) = \xi\}$

1: $M \longleftarrow \emptyset$
2: For each node x in G, determine whether $F(x) \geq \xi$ or $F(x) < \xi$
3: Compute the set B of cubes by connecting adjacent nodes in G
4: **for each** $b \in B$
5: $\quad index \longleftarrow$ *Index*(b)
6: $\quad M \longleftarrow M \cup$ *CaseTable_Get*$(index)$
7: **endfor**
8: Compute vertex coordinates in M by interpolation
9: **return** M

Figure 6: A high level description of the Marching Cubes algorithm.

corners of this cube and any isovalue. Our basic data types are *Dimension* and *Coord* that allow us to define most of the other types:

```
Inductive Dimension := Dimension_X | Dimension_Y | Dimension_Z.
Inductive Coord := Coord_Zero | Coord_One.
```

Then each of the six faces of the cube is defined by a pair of *Dimension* and *Coord* values:

```
Inductive CubeFace :=
```

| *CubeFace_Cons* : *Dimension* → *Coord* → *CubeFace*.

In this definition the constructor is written with functional arrow symbols, however it can be thought of as analogous to a record or a class with two fields in conventional programming languages. An edge of the cube is similarly defined by a pair of *CubeFace*s, and a corner of the cube is defined by a triple of *CubeFace*s.

We call the nodes in the corners of the generic cube *CubeCorner*s, and the points in the intersection of the resulting isosurface with the edges of the generic cube *CutVertex*es. There is a one-to-one correspondence between cube edges and *CutVertex*es, and we opted for the use of the latter. Figure 7 shows the conventions we use for the ordering of *CubeCorner*s and *CutVertex*es.

A *CutTriangle* is a triangle in the resulting surface that is defined by three *CutVertex*es. A *CutEdge* is an edge of a *CutTriangle*, defined by two *CutVertex*es. A *Sign* is the possible result of the evaluation $F(c) - \xi$ for a *CubeCorner* c:

```
Inductive Sign := Sign_Neg | Sign_Pos.
```

Below we describe our proof of requirements (i) and (ii-a) in terms of our Coq data types. More specifically, we prove that the following conditions are satisfied.

(i) The set of *CutEdge*s of *CutTriangle*s, as returned by the function *CaseTable_Get*, that lie in *CubeFace*s is equal to the set of *CutEdge*s defined by the two-dimensional rules shown in Figure 3 and encoded by function *Cube_GetCutEdgesInFaces*.

(ii-a) For each *CutEdge* e of each *CutTriangle* from the three-dimensional

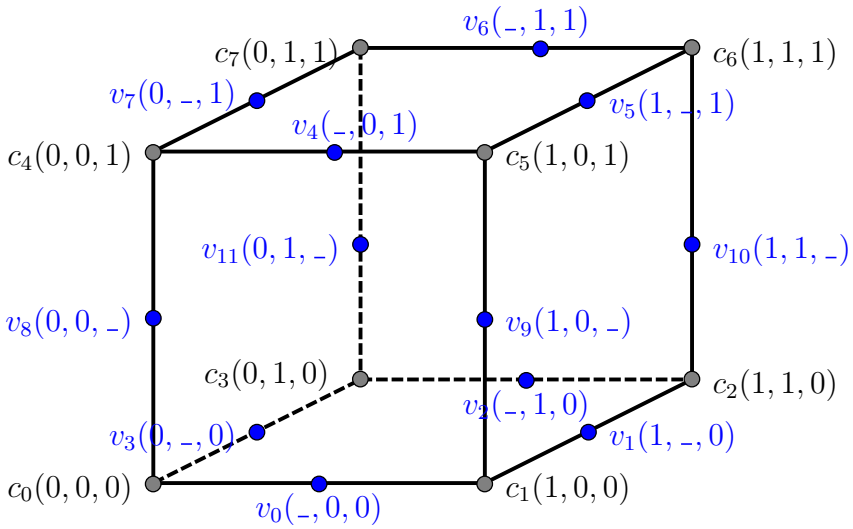


Figure 7: Ordering conventions for *CubeCorner*s and *CutVertex*es. The coordinates of *CubeCorner*s c_i are shown in black, and those of *CutVertex*es v_i are shown in blue. The triples of numbers correspond to the (X, Y, Z) coordinate values in the respective positions. The underscores represent coordinates that are not used in the construction of the corresponding *CutVertex*es.

look-up table, e can appear either exactly once or exactly twice cumulatively in all *CutTriangle*s of the current-case cube:

- if e appears exactly once, then it lies in one of the *CubeFace*s;
- if e appears exactly twice, then it does not lie in any of the *CubeFace*s (i.e., it lies in the interior of the cube).

A two-dimensional Marching Squares algorithm based on the rules of Figure 3 is correct because these rules have the following two properties.

- A *CutVertex* is created in an edge of the grid if and only if this edge has opposite *Sign* values at its ends. Therefore, assuming that the interpolating algorithm for computing the positions of *CutVertex*es is deterministic,

any two squares sharing a side must also share the same *CutVertex* or its absence.

- In every case all the corners marked *Sign_Pos* are completely separated by *CutVertex*es and *CutEdge*s from all the corners marked *Sign_Neg*.

Furthermore, if the rules of Figure 3 are respected by a three-dimensional Marching Cubes algorithm in the shared cube faces, the triangulations constructed in adjacent cubes will always include the same *CutEdge*s in the shared faces, and holes similar to the one shown in Figure 4a will not be created.

Our theorem shown below proves that conditions (i) and (ii-a) are satisfied by the MC implementations we study for all possible assignments of *Sign*s to *CubeCorner*s.

```
Theorem Combinatorial_Correctness:
∀ s0 s1 s2 s3 s4 s5 s6 s7 : Sign,
let FaceEdges := Cube_GetCutEdgesInFaces s0 s1 s2 s3 s4 s5 s6 s7 in
let i := Index [s0; s1; s2; s3; s4; s5; s6; s7] in
let CutTriangles := CaseTable_Get i in
(SetEqual FaceEdges (CollectFaceEdges CutTriangles) CutEdge_eq) = true ∧
(EdgeList_IsConsistent CutTriangles) = true.
Proof.
  intros;
  destruct s0, s1, s2, s3, s4, s5, s6, s7;
  vm_compute;
  auto.
Qed.
```

In this theorem, the function *Cube_GetCutEdgesInFaces* returns the *list* of *CutEdge*s from a direct application of the rules of Figure 3 to the six faces of the generic cube under a given assignment of *Sign*s to the corners of the cube. The function *CollectFaceEdges*, on the other hand, returns the *list* of *CutEdge*s by iterating through the *CutTriangle*s from the three-dimensional look-up table under the same assignment of *Sign*s and collecting only those *CutEdge*s that lie in the faces of the cube. The function *SetEqual* checks the two lists for equality. The function *EdgeList_IsConsistent* verifies condition (ii-a).

The proof is based on a direct enumeration of all possible assignments of eight *Sign* values. The commands used for the proof of the theorem are called *tactics*. The tactics are chosen manually from the predefined set. They allow for the transformation of the premises of a theorem to its conclusion. The proof is an interactive process in which the system presents current goals, and the user transforms or discharges them by using appropriate tactics. When all the goals are discharged and the `Qed` command accepted, the theorem is guaranteed to have been proven correctly. As in our case, the tactics can be stacked together with the ";" delimiter, such that they are applied automatically without interaction with the user. More specifically, the tactic `intros` introduces the universally quantified values into the proof context, such that they are treated as concrete parameters. The `destruct` tactic enumerates all combinations of values that its parameters can assume. The `vm_compute` tactic applies conventional computation to evaluate the current goal, and the `auto` tactic finishes off the remaining basic logical and arithmetic operations.

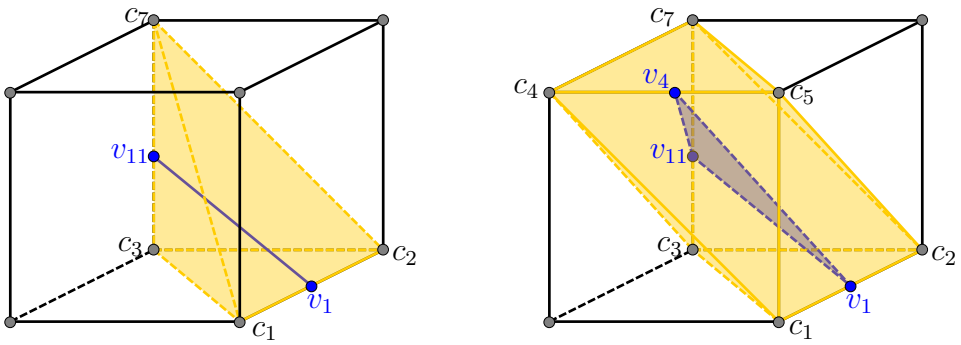
4. Proof of Geometric Correctness

In this section we describe our proof of proposition (ii-b) which is stated in terms of our Coq types as follows.

(ii-b) For any assignment of *Sign*s to the corners of the generic cube, the interiors of the *CutTriangle*s defined by the three-dimensional look-up table do not intersect each other, independently of the positions of *CutVertex*es computed by interpolation at run time.

The fact that the positions of *CutVertex*es are not known a priori implies that we need to consider all possible positions for each vertex, i.e., the whole cube edge on which the vertex lies. As a result, we can think of each *CutTriangle* spanning (or sweeping) a region of space as each of its *CutVertex*es independently spans (or sweeps) its corresponding edge of the cube. This is illustrated by examples in Figure 8 where we first show the tetrahedron whose interior is the union of all points in space that can possibly lie in *CutEdge* v_1v_{11}, and then the 8-sided polytope whose interior is the union of all points in space that can possibly lie in *CutTriangle* $v_1v_4v_{11}$.

Let the *spanned polytope of* α, abbreviated as $\mathcal{SP}(\alpha)$, where α is a *CutEdge* or a *CutTriangle*, be the union of all points in space that can possibly lie in the closure of α. Then the spanned polytope is the three-dimensional convex hull of the *CubeCorner*s incident to the cube edges that contain the *CutVertex*es of α. If a spanned polytope is flat, then we define its *interior* as the two-dimensional open region (embedded in the three-dimensional space) enclosed by the polytope's piecewise-linear boundaries. If a spanned polytope has non-zero volume, then its *interior* is the three-dimensional open region enclosed by its piecewise-planar boundaries.



(a) Spanned polytope (tetrahedron) for *CutEdge* v_1v_{11}.

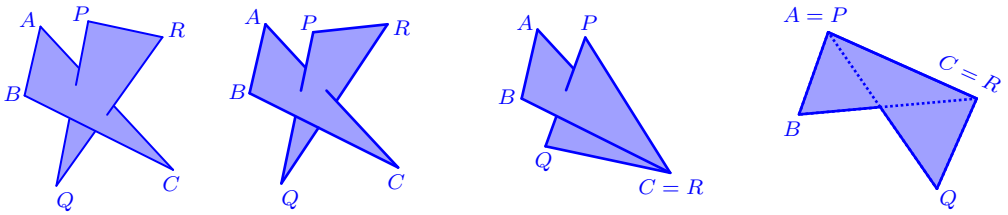
(b) Spanned polytope for *CutTriangle* $v_1v_4v_{11}$.

Figure 8: Examples of spanned polytopes.

The cases shown in Figure 8 are the most general, in the sense that the cube edges containing the *CutVertex*es are not adjacent: c_1c_2, c_3c_7, and c_4c_5 do not share any end points. In the cases when such edges share end points, some of the triangular faces of the spanned polytope degenerate to segments or single points.

Two *CutTriangle*s, ABC and PQR, may share zero, one, or two vertices. We analyze each case separately below.

- If ABC and PQR do not share any vertices (Figure 9a), then their interiors may intersect only if the interiors of $\mathcal{SP}(ABC)$ and $\mathcal{SP}(PQR)$ intersect.
- If ABC and PQR share one vertex (Figure 9b), say $C = R$, then their interiors may intersect only if the interiors of $\mathcal{SP}(ABC)$ and $\mathcal{SP}(PQ)$ intersect, or the interiors of $\mathcal{SP}(PQR)$ and $\mathcal{SP}(AB)$ intersect.
- Finally, when ABC and PQR share two vertices (Figure 9c), their interiors may intersect only if the triangles are coplanar. This last case does not create material inconsistencies in the MC output, and we do not consider



(a) No vertices are shared. (b) One vertex is shared. (c) Two vertices are shared.

Figure 9: All configurations (modulo vertex ordering) of the interior of triangle PQR intersecting the interior of triangle ABC ($PQR \neq ABC$). The same configurations can be obtained by interchanging the roles of triangles PQR and ABC.

it further.

A necessary and sufficient condition for the separability (i.e., the absence of intersection) of two polytopes is the existence of a *separating plane*, which divides the space into two parts, each fully containing one of the polytopes. All of the potential separating planes are defined by the faces of both polytopes and by the planes computed from all pairs of edges, one from each polytope [23]. For the purposes of our proof, it is sufficient to determine if two polytopes are separated through the existence of one separating plane. We found that in all cases separability can be proven by testing only the planes that pass through the faces of the polytopes. We identify all faces of each polytope by considering all non-collinear triples of vertices of this polytope. We do not need to pre-process the faces by separating them into internal and external, since the internal faces will always fail the test for separability.

To test if a plane is separating for two polytopes, we check if no vertex of one polytope lies strictly to the same side of the plane as any vertex of the other polytope. Given a plane passing through points A, B, and C, the orientation of the fourth point D with respect to the plane is determined by evaluating the sign of the vector expression $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$, where $\mathbf{a} = A - D$, $\mathbf{b} = B - D$, and $\mathbf{c} = C - D$. This test involves only addition, subtraction, and multiplication of the coordinates of cube corners (0 and 1) and therefore can be evaluated exactly in integer arithmetic readily available through Coq data type *Z*.

Our theorem which proves the absence of triangle intersections is shown below.

```
Theorem Geometric_Correctness:
∀ s0 s1 s2 s3 s4 s5 s6 s7 : Sign,
let i := Index [s0; s1; s2; s3; s4; s5; s6; s7] in
let CutTriangles := CaseTable_Get i in
(CutTrianglesSeparated CutTriangles) = true.
Proof.
  intros;
  destruct s0, s1, s2, s3, s4, s5, s6, s7;
  vm_compute;
  trivial.
Qed.
```

In this theorem, the function *CutTrianglesSeparated* examines all pairs of *CutTriangle*s in a single cube, for each pair determines the faces of the

corresponding spanning polytopes, and tests the faces for the property of defining a separating plane for these two polytopes. If at least one separating plane is found, the pair of *CutTriangle*s is deemed non-intersecting.

5. Conclusions

We presented a Coq proof of correctness of an implementation of a Marching Cubes algorithm. Our effort was driven by two trends in computer-aided design and analysis. The first trend is the increased reliance on computational techniques in areas that cannot afford erroneous results, such as health care and transportation. The second trend is the requirement that computation is performed in real time, usually involving parallel computing hardware, which effectively excludes a human operator who could evaluate and correct the computational pipeline on an as-needed basis.

Our proof can be used as-is or extended to verify other implementations of the Marching Cubes algorithm. For example, one might use a more complex lookup table if certain assumptions are made about the true surface, such as that its topology can be reconstructed correctly by trilinear interpolation [7]. One extension to our proof that might be needed for other tables is the testing of potential separating planes defined by pairs of edges of two spanning polyhedra.

6. Acknowledgment

Molecular graphics and analyses were performed with the UCSF Chimera package. Chimera is developed by the Resource for Biocomputing, Visual-

ization, and Informatics at the University of California, San Francisco (supported by NIGMS P41-GM103311).

References

[1] W. J. Schroeder, K. M. Martin, Overview of visualization, in: C. Johnson, C. Hansen (Eds.), Visualization Handbook, Academic Press, Inc., 2004.

[2] T. S. Newman, H. Yi, A survey of the marching cubes algorithm, Computers & Graphics 30 (5) (2006) 854–879.

[3] W. E. Lorensen, H. E. Cline, Marching cubes: A high resolution 3D surface construction algorithm, SIGGRAPH Comput. Graph. 21 (4) (1987) 163–169.

[4] T. D. Goddard, C. C. Huang, T. E. Ferrin, Visualizing density maps with UCSF Chimera, Journal of Structural Biology 157 (1) (2007) 281–287.

[5] E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, T. E. Ferrin, UCSF Chimera—A visualization system for exploratory research and analysis, Journal of Computational Chemistry 25 (13) (2004) 1605–1612.

[6] B. Natarajan, On generating topologically consistent isosurfaces from uniform samples, The Visual Computer 11 (1994) 52–62.

[7] E. V. Chernyaev, Marching cubes 33: Construction of topologically correct isosurfaces, Tech. Rep. CERN CN/95-17 (1995).

[8] C. Montani, R. Scateni, R. Scopigno, A modified look-up table for implicit disambiguation of marching cubes, The Visual Computer 10 (1994) 353–355.

[9] A. M. Lopes, Accuracy in scientific visualization, Ph.D. thesis, The University of Leeds (1999).

[10] M. J. Dürst, Letters: Additional reference to "marching cubes", Computer Graphics 22 (1988) 72–73.

[11] W. Heiden, T. Goetze, J. Brickmann, Fast generation of molecular surfaces from 3D data fields with an enhanced "marching cube" algorithm, Journal of Computational Chemistry 14 (1993) 246–250.

[12] P. Hammer, Matlab implementation of marching cubes, http://www.mathworks.us/matlabcentral/fileexchange/32506-marching-cubes (2011).

[13] The Coq proof assistant, version 8.4pl5, http://coq.inria.fr.

[14] Y. Bertot, P. Castéran, Coq'Art: The Calculus of Inductive Constructions, Springer, 2004.

[15] J.-F. Dufourd, Y. Bertot, Formal study of plane Delaunay triangulation, in: M. Kaufmann, L. C. Paulson (Eds.), Interactive Theorem Proving, Lecture Notes in Computer Science, Vol. 6172, Springer, 2010, pp. 211–226.

[16] G. Gonthier, Formal proof – the Four-Color Theorem, Notices of the AMS 55 (2008) 1382–1393.

[17] J.-F. Dufourd, A hypermap framework for computer-aided proofs in surface subdivisions: genus theorem and Euler's formula, in: Proceedings of the 2007 ACM symposium on Applied computing, ACM, New York, NY, 2007, pp. 757–761.

[18] C. Brun, J.-F. Dufourd, N. Magaud, Designing and proving correct a convex hull algorithm with hypermaps in Coq, Computational Geometry Theory and Applications 45 (8) (2012) 436–457.

[19] N. Magaud, J. Narboux, P. Schreck, A case study in formalizing projective geometry in Coq: Desargues theorem, Computational Geometry Theory and Applications 45 (8) (2012) 406–424.

[20] C. Dehlinger, J.-F. Dufourd, Formal specification and proofs for the topology and classification of combinatorial surfaces, Computational Geometry Theory and Applications 47 (9) (2014) 869–890.

[21] F. Labelle, J. R. Shewchuk, Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles, ACM Transactions on Graphics 26 (3) (2007) 57.1 – 57.10.

[22] A. Chernikov, J. Xu, A computer-assisted proof of correctness of a marching cubes algorithm, in: International Meshing Roundtable, Springer, Orlando, FL, 2013, pp. 505–523.

[23] C. Ericson, Real-time collision detection, Elsevier Amsterdam/Boston, 2005.