EUROGRAPHICS 2005 / M. Alexa and J. Marks
(Guest Editors)

Volume 24 (*2005*), *Number 3*

Cubical Marching Squares: Adaptive Feature Preserving Surface Extraction from Volume Data

Chien-Chang Ho[†] Fu-Che Wu[†] Bing-Yu Chen[‡] Yung-Yu Chuang[§] Ming Ouhyoung[§]

National Taiwan University

Abstract

In this paper, we present a new method for surface extraction from volume data which preserves sharp features, maintains consistent topology and generates surface adaptively without crack patching. Our approach is based on the marching cubes algorithm, a popular method to convert volumetric data to polygonal meshes. The original marching cubes algorithm suffers from problems of topological inconsistency, cracks in adaptive resolution and inability to preserve sharp features. Most of marching cubes variants only focus on one or some of these problems. Although these techniques could be combined to solve these problems altogether, such a combination might not be straightforward. Moreover, some feature-preserving variants introduce an additional problem, inter-cell dependency. Our method provides a relatively simple and easy-to-implement solution to all these problems by converting 3D marching cubes into 2D cubical marching squares, resolving topology ambiguity with sharp features and eliminating inter-cell dependency by sampling face sharp features. We compare our algorithm with other marching cubes variants and demonstrate its effectiveness on various applications.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computation Geometry and Object Modeling: Curve, surface, solid and object representations

1. Introduction

Volumetric and polygonal representations are arguably the two most popular representations for geometric objects in computer graphics. Polygonal representation allows efficient rendering on modern graphics hardware, but it is not an effective representation for time-varying applications and for performing geometric manipulations, such as Constructive Solid Geometry (CSG) modeling or boolean operations [BKZ01]. On the contrary, such geometric operations would be easier with volumetric representation, although rendering volumetric data is less efficient than polygonal meshes on modern graphics architecture.

To display the volumetric data efficiently, the well-known marching cubes algorithm [LC87] and its variants provide a convenient and efficient way to convert the volumetric data into polygonal meshes. These methods allow us to accurately represent geometric objects as volumetric data, manipulate them volumetrically, and efficiently display them by converting on the fly the volumetric data into polygonal meshes. However, although the original marching cubes algorithm is generally effective, it has problems with *topological inconsistency*, *cracks* in adaptive resolution and inability to preserve *sharp features*.

The first problem with the original marching cubes algorithm is topological inconsistency because of topology ambiguities. Such ambiguities arise when there are more than one feasible assignments for a case in the lookup table of marching cubes. In these cases, the triangulation has to choose which pairs of intersections to connect or to decide whether two components are separated or joined. An inconsistent ambiguity resolution strategy could lead to holes.

The second problem is cracks in adaptive resolution. Adaptive methods apply marching cubes to an adaptive grid to reduce the number of resulting triangles. However, they can result in cracks at the interfaces of grid cells at dif-

† e-mail:{murphyho, joyce}@cmlab.csie.ntu.edu.tw
‡ e-mail:robin@ntu.edu.tw
§ e-mail:{cyy, ming}@csie.ntu.edu.tw

ferent resolutions. Such a problem is often overcome by crack patching. While crack patching is effective, it often stretches the high-resolution edges to match with low-resolution edges and hence does not take full advantage of the finer-resolution data.

The third problem is the inability to preserve sharp features. The original marching cubes algorithm assumes that the underlying surface is smooth and does not preserve sharp edges and corners. Hence, a flat surface might become wavy. The main idea for solving this problem is to find the exact intersection of the zero-crossing points' tangent planes. To define the tangent planes, in addition to a scalar field, sharp-feature-preserving algorithms require exact information of normals for zero-crossing points.

Finally, while sharp-feature-preserving algorithms improve the accuracy of extracted surface, some of them, unfortunately, introduce another problem of *inter-cell dependency*, i.e., the extracted surface of a cell might depend on the results of its neighboring cells. The inter-cell dependency makes the computation slower and more complex. Hence, eliminating inter-cell dependency improves the performance and makes it easier to implement surface extraction algorithms on programmable graphics processing units (GPUs), which have more computing power than CPUs.

To sum up, current marching-cube-style techniques could still suffer from some problems of topological inconsistency, cracks in adaptive resolution, sharp feature preservation and inter-cell dependency. Such problems limit the speed and accuracy of applications using these techniques. These problems are often discussed and solved individually in previous literature. Although previous techniques could be combined to solve these problems altogether, such a combination might be complicated or even impossible. In this paper, we propose a new solution that converts the marching cubes into cubical marching squares, determines topology with sharp features and eliminates inter-cell dependency. We call this method *cubical marching squares* (CMS) method. By reducing three-dimensional problems into two-dimensional ones, our method effectively overcomes all the above problems in a simple and intuitive manner.

2. Related Work

Marching cubes (MC) algorithm was proposed by Lorensen and Cline in 1987 [LC87]. It analyzes the binary pattern of eight vertices of a cube to construct a surface that approximates the underlying surface. Considering rotations and symmetries, they reduce the original 256 patterns to a total of 15 configurations. As described previously, although the marching cubes algorithm has been proved effective, it has several problems and many variants were hence proposed to address these problems.

First of all, there have been two types of ambiguities found in certain configurations where there are more than one ways to triangulate. The first is *face ambiguity*. It arises when a face has two diagonally opposite vertices marked positive and the other two marked negative. Nielson and Hamann [NH91] show how this can happen between neighboring cells and may lead to holes and inconsistent topology. Another is *internal ambiguity* which occurs in the interior of a cell. Natarajan [Nat94] and Chernyaev [Che95] independently identify this type of problem and provide solutions. These ambiguities can often be resolved by adding exact sample points inside each cell. To determine these extra points, many methods assume that "the implicit function of the volumetric data is linear along an edge; bilinear on a face; and trilinear inside a cell." Under this trilinear assumption, several methods are proposed to resolve ambiguous cases on the faces [NH91] and inside the cells [LB03, Nie03], and thus to determine a consistent topology. More recently, Lewiner *et al.* [LLVT03] provide an efficient and complete implementation of Chernyaev's method. We call these algorithms topology-consistent marching cubes (TMC) in this paper.

When applying the marching cubes algorithm to a uniform grid, the number of resulting triangles could be large even if the original surface is quite simple. To reduce the number of triangles, several methods have been developed to apply marching cubes algorithm to an adaptive grid, such as an octree [WG92, SCK95]. Crack patching is performed to fill cracks where two cells of different resolutions meet [SFYC96]. Heidrich *et al.* also propose a real-time adaptive isosurfacing method [HSE99].

The original marching cubes algorithm does not represent sharp features well. By using extra information of normals, Kobbelt *et al.* [KBSS01] propose the extended marching cubes (EMC) algorithm which preserves sharp features. The EMC method has basically two operations. One is detecting and sampling the sharp features; the other is edge flipping. The main idea is to find the exact intersection points by intersecting the tangent planes of the zero-crossing points. Ju *et al.* propose dual contouring (DC) [JLSW02], a hybrid method of EMC [KBSS01] and SurfaceNets [Gib98] algorithms. They use EMC to sample sharp features and SurfaceNets to connect the features to form the surfaces. Their method preserves sharp features and prevents holes and cracks in adaptive resolution. However, these methods do not resolve ambiguous cases and may have holes due to topological errors. Furthermore, these methods introduce additional problem of inter-cell dependency because of the edge flipping operation. It makes the computation more complex and slower.

Some recent work attempts to extend the previous algorithms to solve these problems more completely. For example, several papers attempt to enhance the dual contouring to overcome the problem of inconsistent topology. Zhang *et al.* [ZHK04] propose an enhanced dual contouring method which allows more than one feature points in-

	MC	TMC	EMC	DC	CMS
adaptive refinement				√	√
topological consistency		√			√
sharp-feature preservation			√	√	√
inter-cell independence	√	√			√

Table 1: *The comparison of the five related methods, original marching cubes (MC), topology-consistent marching cubes (TMC), extended marching cubes (EMC), dual contouring (DC) and our method (CMS). Note that previous methods could be combined, but sometimes, such a combination might not be straightforward. Hence, we make this table by only considering the original version of each method.*

side a cell to preserve the topology of the cell. Schaefer and Warren [SW04] provide a primal contouring method of dual grids to represent thin features without excessive subdivision. Varadhan *et al.* [VKKM03] propose a method which allows multiple intersections along an edge to reconstruct thin features without creating unwanted handles. Varadhan *et al.* [VKSM04] propose a method to adaptively subdivide cells until the piece of surface inside a cell is topologically equivalent to a disk. However, they use the original marching cubes algorithm and do not preserve sharp features.

Another related work is the method proposed by Rodehorst and Kimia [RK02]. They apply a higher-order-polynomial interpolation, called ENO interpolation [SKS97], to several consecutive sample points to sample possibly more than one zero-crossing points, called ENO anchor points, on an edge of a cell. They prove that there are at most two anchor points on an edge and allow an edge have two zero-crossing points. The surface is reconstructed by triangulating these anchor points. Triangulation is performed step by step, adding one triangle at a time, until all anchor points are triangulated. When there is an ambiguity, i.e., multiple possible triangles to choose, they use the normals of triangles created in neighboring cells to choose the one which results in the smoothest surface. Although this approach guarantees consistent topology, it has the following drawbacks: (1) it is highly inter-cell dependent; actually, the generation of each triangle depends on the previously generated triangle; (2) it does not consider 3D sharp features and internal ambiguity; and (3) because an edge could have at most two zero-crossing points, the rule becomes more complicated.

Table 1 compares our CMS algorithm with other four methods: original marching cubes, topology-consistent marching cubes, dual contouring and extended marching cubes. Note that these methods could be combined, but sometimes, such a combination might not be straightforward. Furthermore, inter-cell independency is often ignored by sharp-feature-preserving methods based on EMC.

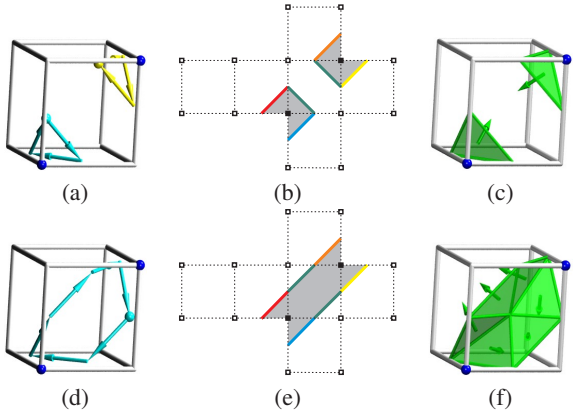


Figure 1: *Cubical marching squares. A marching cube (a, d) can be unfolded into six marching squares (b, e). Each square is processed independently. The generated segments on these faces are put back to 3D to form components (a, d). By doing so, we can achieve the goal of being adaptive without performing crack patching. In addition, face ambiguities can be resolved in 2D by resolving the ambiguous faces (the middle faces in (b, e)). Finally, the resulting components are triangulated to generate the isosurface (c, f).*

3. Cubical Marching Squares

This section describes our algorithm for extracting a triangle mesh from a given geometric representation. It first describes the input to the algorithm (Section 3.1). Then, it explains the main ideas behind the algorithm and gives an overview of the algorithm (Section 3.2). The following two sections describe in more details the two main steps of our algorithm: segment generation (Section 3.3) and surface extraction (Section 3.4).

3.1. Input

The input of our algorithm is a geometric representation for volume data such as a polygonal mesh, an implicit surface, a set of point clouds, or a scalar distance field. The first step is to convert different types of geometric representations into a uniform format. We choose the same format as the dual contouring algorithm [JLSW02], a signed grid with edges tagged with exact intersection points (sample points) and their normals (sample normals). This kind of data was called *Hermite data* by Ju *et al.* [JLSW02]. As pointed out by Kobbelt *et al.* [KBSS01], for most geometric representations, Hermite data can be computed directly or derived implicitly. Since our goal is to represent the volume data as precisely as possible, the Hermite data is acquired at a very fine resolution, say, a uniform $n_k \times n_k \times n_k$ grid. Our algorithm then generates a polygonal approximation for the Hermite data without referring to the original geometric representation.

3.2. Algorithm overview

Given the Hermite data in the previous section, we attempt to adaptively reconstruct a polygonal mesh which approximates the original volume data as precisely as possible. Our algorithm is built on the following ideas: (1) marching cubes can be unfolded as marching squares; (2) inter-cell dependency can be eliminated by adding sharp features on faces; and (3) sample normals can be used not only to sample sharp features but also to solve ambiguities and to maintain consistent topology.

As shown in Figure 1, a cube can be unfolded into six faces. For each face, we generate the isocurve using the marching squares algorithm. The resulting isocurve for each face consists of several segments. If we fold these faces back to form the original cube and connect together these segments properly, we obtain exactly the same components as marching cubes algorithm does. Finally, these components are triangulated to generate the isosurface. The triangulation can be chosen arbitrarily as long as it is consistent. Hence, a marching cube table lookup can be converted to six marching square table lookups and a component tracing operation. Hence, we call this method cubical marching squares. It is equivalent to marching cubes but generally slightly slower. However, as discussed later, it allows us to generate polygonal meshes adaptively in a simple and consistent way without performing crack patching. Furthermore, it allows us to sample sharp features on faces to eliminate inter-cell dependency.

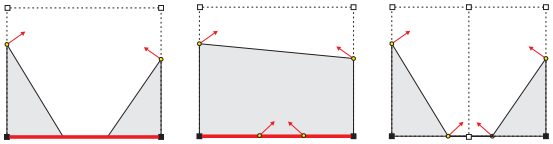
Many sharp-feature-preserving algorithms use sample normals to detect and sample the sharp features for a component of the isosurface inside a cell. These methods often suffer in the resulting surface from the problem of topology error. On the other hand, previous methods for solving ambiguities and maintaining topology use the trilinear assumption and do not take sharp features into account. In contrast, we find that the detection of sharp features can be used to solve ambiguities as well. Hence, we use the same procedure to achieve the goals of preserving sharp features and maintaining consistent topology.

Our algorithm has three stages: constructing an adaptive signed octree, generating segments for each leaf face, and, finally, extracting surfaces for each cell in the signed octree. The pseudo code in Algorithm 1 describes our algorithm more precisely. We start from a very coarse uniform $n_0 \times n_0 \times n_0$ base grid B, in our implementation, $n_0 = 8$. For each cell c in B, the procedure SUBDIVIDECELL checks whether this cell needs to be further subdivided. A cell is subdivided if one of the following conditions holds:

- It has an edge ambiguity. When there are more than one sample points on an edge of the cell as shown in Figure 2(a), the cell should be subdivided (Figure 2(c)). Otherwise, the surface will be generated incorrectly (Figure 2(b)).

Algorithm 1 Cubical Marching Squares. *Given Hermite data, this procedure generates the corresponding triangle mesh.*

```
1:  procedure CUBICALMARCHINGSQUARES(HermiteData H)
2:      InitializeBaseGrid(B);  ▷ initialize a coarse base grid B
3:      for each cell c in B
4:          SUBDIVIDECELL(H, c);
5:      end for
6:      for each leaf face f
7:          GENERATESEGMENT(f);
8:      end for
9:      for each leaf cell c
10:         EXTRACTSURFACE(c);
11:     end for
12: end procedure
```



(a) (b) (c)

Figure 2: *Edge ambiguity. In (a), the red edge has two sample points and there is an edge ambiguity. If the cell is not further subdivided, such an ambiguity can lead to a wrong surface (b). We resolve this ambiguity by subdivision (c).*

- It has the tendency to contain a complicated surface. We detect this by a heuristic, checking whether the maximal spanning angle of all pairs of sample normals inside this cell exceeds a predefined angle threshold. When this happens, it means that the surface inside a cell might not be flat enough and should be subdivided.

When subdividing a cell, we subdivide its faces first. Each face is subdivided into four subfaces and the relationships between subcells and subfaces are recorded. We stop the subdivision if it exceeds the maximal level of subdivision, k. Hence, the finest resolution is $n_k = 2^k n_0$. The result of this subdivision step is an adaptive signed octree. A cell in the octree is called a *leaf cell* if it does not have subcells. A face is called a *leaf face* if it does not have subfaces. Note that a leaf cell could have a complicated non-leaf face if any of its neighbors is at a deeper level and subdivides their shared face. The face shared by two cells subdivided at two different levels is called a *transition face*. Cracks could happen if the transition faces are not handled properly.

3.3. Segment generation for faces in 2D

Once we have built the adaptive signed octree, the next step is to use the procedure GENERATESEGMENT in Algorithm 2 to extract the segments for all leaf faces. For the face f be-

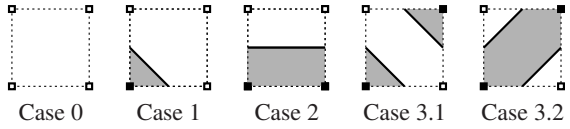


Figure 3: *Lookup table for marching squares. Case 3 has a face ambiguity, so there are two possible assignments of pairs to connect, cases 3.1 and 3.2.*

ing processed, by checking the sign patterns of f's vertices with the lookup table for marching square (Figure 3), we can determine how many segments f has. Case 0 gives no segments; Cases 1 and 2 give one segment; and Case 3 gives two segments, here there is a face ambiguity. In such an ambiguity, we do not know which pairs of sample points should be connected to form segments. With the help of sample normals, we resolve this ambiguity by checking the sharp features.

Algorithm 2 GenerateSegment. *This procedure finds all segments for a face f, resolves face ambiguity if any, and samples sharp features if necessary.*

```
1:  procedure GENERATESEGMENT(Face f)
2:      if there are two segments then      ▷ Case 3 in Figure 3
3:          {l1, l2} ← RESOLVEFACEAMBIGUITY(f);
4:          f.list ← {l1, l2};
5:          DETECTFACESHARPFEATURE(f.list);
6:      else if there is one segment l then
7:          f.list ← {l};
8:          DETECTFACESHARPFEATURE(f.list);
9:      end if
10: end procedure
```

A 2D sharp feature can be detected by finding the intersection point of the two tangent lines defined by the sample points and their normals. We resolve the face ambiguity by detecting whether sharp features overlap. As shown in Figure 4, one of two possible segment assignments (Figure 4(a)) has overlapped sharp features. This is not a valid assignment because the input Hermite data describes a volume and a volume should not intersect itself. Hence, we choose the assignment without feature overlaps (Figure 4(b)) and resolve the face ambiguity. Although the results are possibly different from the results obtained using asymptotic deciders, we found it effective to decide the face ambiguity by testing sharp feature overlap.

Finally, for each segment, we detect if there is a face sharp feature on the segment by testing whether the angle between two normals are large enough. If there is a sharp feature, we tag the segment and store the position. This face sharp feature is used to remove the inter-cell dependency. As stated in Section 2, EMC algorithm [KBSS01] has inter-cell dependency because of the edge flipping operation. For the sharp

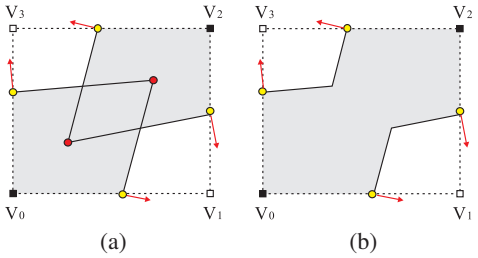


Figure 4: *Face ambiguity. Face ambiguity is resolved by testing whether sharp features overlap. Since the input data describes a volume, it should not intersect with itself. Hence, the segment assignment with feature overlapping (a) is not valid. We choose the assignment (b) to form two segments and resolve the ambiguity.*

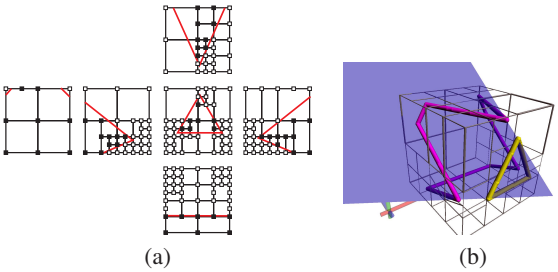


Figure 6: *Cubical marching squares. For the cell in (b), the resulting faces could describe a complicated piecewise linear curves (a). The yellow and magenta line loops in (b) are components.*

features in Figure 5(a), to correct the connectivity, EMC flips an edge to connect two sharp features as shown in Figure 5(b). However, after the flip, not all resulted triangles are located inside cells. This is called inter-cell dependency. DC method [JLSW02] has a similar drawback. Hence, these algorithms require a extra computation on adjacent cells to restore the correct connectivity. Such a dependency also makes it more difficult to extend EMC to be adaptive. To remove this dependency, we sample a face sharp feature on the interfacing face (Figure 5(d)) between two adjacent cells. This extra face feature removes the need for edge flipping and the inter-cell dependency as shown in Figure 5(c).

After this stage, for each leaf face, we find all segments of this face. For a non-leaf face, its segments are the union of the segments of its subfaces. A leaf face could have at most four line segments if it has two sharp features. However, a non-leaf face could have a set of segments representing very complicated piecewise linear curves. For example, Figure 6(a) shows the resulting segments on each face for the cell in (b). The next section explains how to construct components (the yellow and magenta line loops in Figure 6(b)) in a cell using the segments of its six faces.

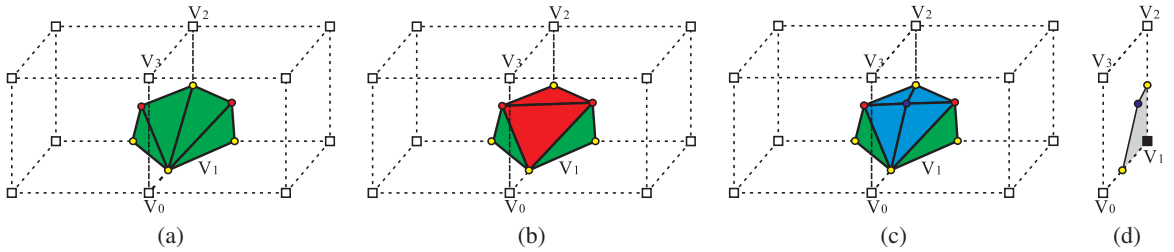


Figure 5: *Inter-cell dependency. EMC flips an edge in (a) to restore the edge features as shown in (b). It, however, introduces inter-cell dependency. CMS samples a face sharp feature as shown in (d) to eliminate inter-cell dependency (c).*

3.4. Surface extraction for cells in 3D

We use the procedure EXTRACTSURFACE in Algorithm 3 to connect segments to form components and to generate triangles. For each leaf cell, we first collect all segments belonging to this cell, that is, the union of the segments of its six faces. The procedure GETSEGMENT returns all segments belonging to a face. After collecting all segments on the faces of the cell c, we trace all the components by grouping together the segments which form a circle. This can be easily done by starting from an edge and sequentially finding the next edge which shares an end point with the current edge. Each circle represents a component o.

Algorithm 3 ExtractSurface. *This procedure uses the segments found by* GENERATESEGMENT *to construct 3D components inside a cell, resolves internal ambiguity if necessary and generates triangles as the output.*

```
1:  procedure EXTRACTSURFACE(Cell c)
2:      l ← ∅;          ▷ list of all segments belonging to the cell c
3:      for each face f_i of the cell c                  ▷ i ← 1 to 6
4:          l ← l ∪ GETSEGMENT(f_i);
5:      end for
                ▷ connecting segments whose end points coincide
6:      O ← TRACECOMPONENTS(l);
7:      for each component o in O
8:          DETECTSHARPFEATURE(o);
9:      end for
10:     if HasInternalAmbiguity(O) then
11:         RESOLVEINTERNALAMBIGUITY(O);
12:     else
13:         for each component o
14:             TRIANGULATION(o);
15:         end for
16:     end if
17: end procedure
```

By definition, a crack happens where there exists an edge owned only by a single component. This can only happen on the transition faces. In our algorithm, all edges on the transition faces are generated from segments and every segment is exactly shared by two components from two neighboring cells. Hence, the resulting mesh is guaranteed crack free.

To preserve 3D sharp features, we then sample sharp features for each resulting components. Sharp features are sampled as suggested by Kobbelt *et al.* [KBSS01], that is, solving $[\ldots n_i \ldots]^T p = [\ldots n_i s_i \ldots]$ by singular value decomposition, where s is the location of a sample point, n is a sample normal, and p is the location of the sharp feature.

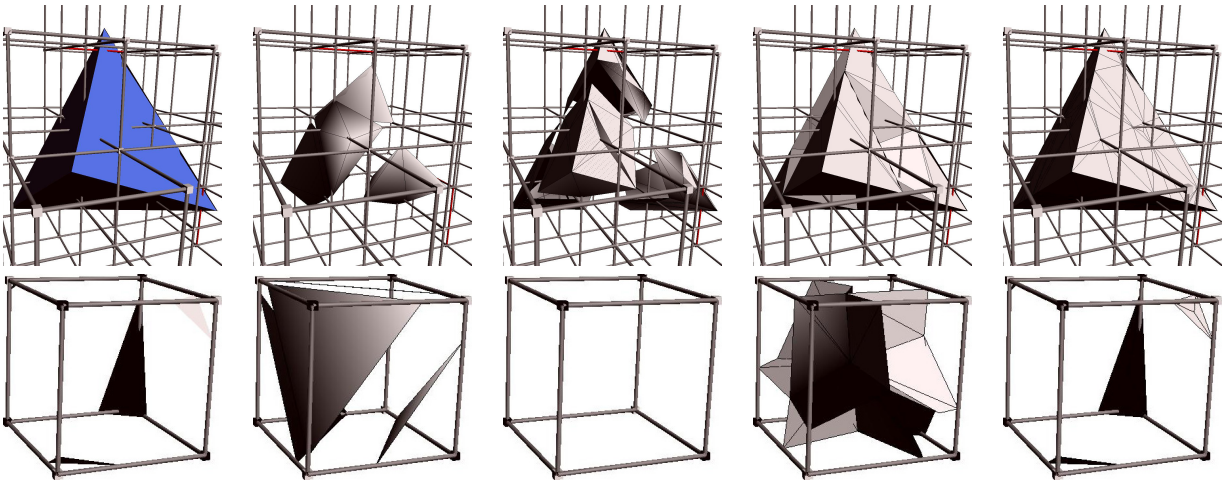
If there is a sharp feature p in a component consisting of the vertices $v_1, \cdots, v_n$ (including vertices of the segments in this component and face sharp features tagged on these segments), we use p as the center to create a triangle fan with triangles, $pv_1v_2, pv_2v_3, \cdots, pv_{n-1}v_n, pv_nv_1$. If there is no component sharp feature, we calculate the average point of all sample points on this component and use it as the center to generate the triangle fan.

3D sharp features are also used to detect and resolve internal ambiguity. Internal ambiguity occurs where we can not determine whether two components are joined or separated by only looking at the signs on the vertices of a grid. Similar to the resolution of face ambiguity, we resolve internal ambiguity by checking whether 3D sharp features of two components overlap. For each component, a cone-like volume is formed centered at its sharp feature. If the volumes of two components overlap, these components are classified as joined. Otherwise, they are separated. If two components are separated, we generate the triangle fan for each component respectively using the method in the previous paragraph. If two components are joined, the resulting surface is topologically equivalent to a cylinder. We use a dynamic programming algorithm to connect and triangulate these two components to form the surface.

4. Results

The CMS algorithm is provided as an open source library†. We first use a tetrahedron to compare the performance of marching cubes, extended marching cubes, dual contouring

† http://graphics.csie.ntu.edu.tw/CMS/



(a) ground truth (b) MC (c) EMC (d) DC (e) CMS

Figure 7: *Comparisons for the original marching cubes (b), extended marching cubes (c), dual contouring (d) and cubical marching squares (e). The input model is a tetrahedron (a). It is converted into a uniform grid of Hermite data and these algorithms are used to extract surfaces. The top row shows the extracted surfaces and the bottom row shows the close-up views of a single cell. CMS takes both topology and sharp features into account and better approximates the original tetrahedron than the other methods.*

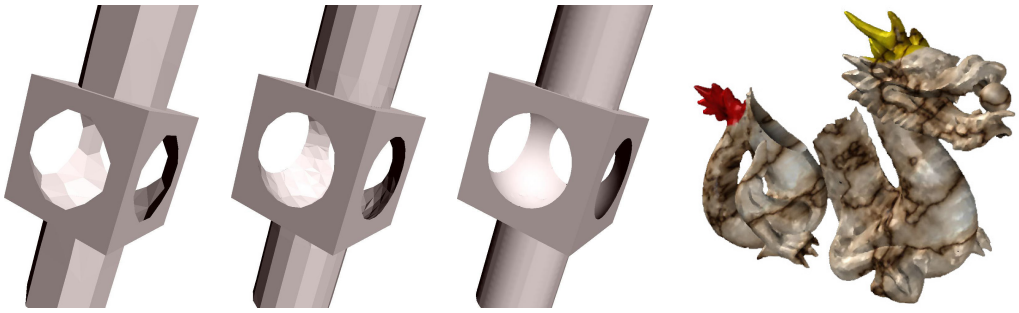
and our method. The top row of Figure 7 shows the resulting surfaces of these algorithms and the bottom row shows close-up views for a single cell. Here, we use a uniform grid and only compare their performance on preserving sharp features and maintaining consistent topology. The original marching cubes algorithm does not preserve sharp features. Except for CMS, these methods do not take topology into account, Hence, there are holes and cracks in the extracted surfaces in Figure 7(b)-(d).

To compare CMS with EMC and DC quantitatively, we performed the following experiment. Three tetrahedra are generated randomly in a limited space and their union is used as the input model. We first convert it into Hermite data and apply EMC, DC and CMS to this model to extract surfaces. We then measure the geometric distances between the resulting surface for each method and the input model. This experiment was repeated many times and Table 2 summarizes the average error for each case of the marching cube lookup table for each method. CMS has the lowest errors for all cases.

case	times	DC	EMC	CMS
1	3,590,980	0.01473	0.00586	0.00383
2	1,554,028	0.02309	0.01310	0.01013
3	207,302	0.10027	0.01801	0.01263
4	30,972	0.23064	0.00601	0.00422
5	803,311	0.03779	0.02631	0.02011
6	101,875	0.11998	0.02737	0.02633
7	12,198	0.17139	0.09565	0.01628
8	109,141	0.03979	0.02831	0.02184
9	72,201	0.04721	0.03525	0.02492
10	4,237	0.19682	0.05283	0.04541
11	70,238	0.04789	0.03535	0.02620
12	30,706	0.09845	0.04559	0.02419
13	1,405	0.85461	0.92935	0.00100
14	70,238	0.04821	0.03573	0.02653

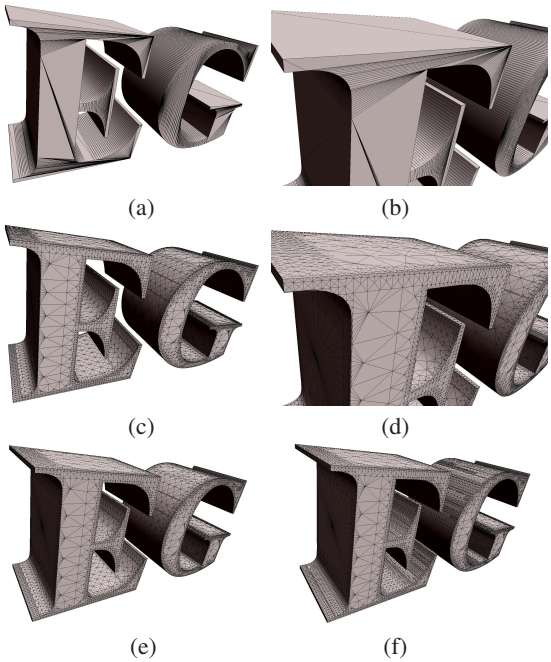
Table 2: *Average geometric errors for the experiment of randomly sampling three tetrahedra. For each case in the marching cube lookup table, we record how many times it happens and the average error for each method. Overall, CMS approximates the input model better than the other two and has the lowest average error for each case.*

We demonstrate the effectiveness of our algorithm on the several possible applications using volumetric data: CSG modeling, level of details and remeshing. CSG modeling is the classical application for volume representations. After applying boolean operations on several volume data, CMS is used to generate a mesh for the resulting CSG model. In Figure 8(a-c), we show the resulting models at different levels of details for the CSG model constructed by the union of a cube and a cylinder, and then subtracting a sphere from it. Here, to clearly show the resulting models, flat shading is used. Table 3 shows the number of resulting triangles at different levels of details and the time of extracting surfaces for them. The time is only for surface extraction from the input volume data, not including the time of converting the input model to its volumetric representation. It was measured on



(a) (b) (c) (d)

Figure 8: *The results of CSG modeling and level of details. (a)-(c) Different levels of details for a CSG model constructed by the union of a cube and a cylinder, and then subtracting a sphere from it. (d) A more complicated CSG model generated by subtracting a sphere from the body of a dragon model and then adding a sphere at its mouth.*



(a) (b) (c) (d) (e) (f)

Figure 9: *Remeshing. The input model is the polygonal model (a) for a text "EG." After applying CMS to the input model with* $\theta = 0.7$*, we obtain the remeshed result (c). For a better comparison, (b) and (d) shows the close-up views for (a) and (c). (e) and (f) shows the remeshing results while using* $\theta = 0.8$ *and* $\theta = 0.9$*.*

level	1	2	3
#triangle	1,688	4,880	14,568
time (ms)	16.23	32.14	66.08

Table 3: *Statistics for Figures 8(a-c).*

	source	$\theta = 0.7$	$\theta = 0.8$	$\theta = 0.9$
#triangle	2,460	39,826	41,118	75,084
time (ms)		126.38	125.38	208.50

Table 4: *Statistics for Figure 9.*

a desktop PC with an Intel Pentium IV 3.2GHz CPU with 1GB memory. Figure 8(d) demonstrates a more complicated CSG model generated by subtracting a sphere from the body of a dragon model and then adding a sphere at its mouth.

Remeshing is another application of volumetric representation. Given a polygonal mesh (Figure 9(a)), we first convert it into a volume representation by sampling its distance field and normals on a fine uniform grid. Applying CMS algorithm to this volume gives a remeshed version of the original mesh (Figure 9(c)), which has a better tessellation than the input. For a better comparison, Figure 9(b,d) show the close-up views of (a,c). Figure 9(e,f) show the remeshing results for different values of θ_{sharp} which is defined in EMC [KBSS01]. This value affects the quality of remeshing. In Figures 9(c,e,f), the triangles representing the flat regions such as in the character E are almost the same. On the other hand, more triangles are generated to represent the regions with higher curvatures such as in the character G for a larger θ_{sharp}. Additionally, sharp features are well preserved. Table 4 shows the number of resulting triangles and the time for surface extraction for Figure 9.

5. Conclusion

In this paper, we have proposed the cubical marching squares algorithm for surface extraction from volume data which preserves sharp features, maintains consistent topology, generates surface adaptively without crack patching and eliminates inter-cell dependency. This method has the following unique features: (1) Hermite data is used to solve the problem of topological ambiguity; (2) the problem of cracks be-

tween adjacent cells when using a multi-resolution representation for the data is solved without crack patching because the shared geometric component is common; (3) 3D features can be reconstructed starting from the 2D features located on the faces of the cells; this avoids inter-cell dependencies; hence, it has potential to be implemented on GPUs. These features make our method quite simple, relatively easy to implement and, at the same time, effective.

We have partially implemented our algorithm on GPU. However, the resulting speed is only comparable to our CPU implementation. As many other GPU algorithms, the bottleneck is the data transfer between CPU and GPU. In the future, we plan to fully implement our algorithm on GPU. We believe that our algorithm will benefit from the improvement on the bus bandwidth between CPU and GPU.

Acknowledgments

The authors wish to thank the reviewers for their helpful comments. We would also like to thank Pei-Lun Lee for his help on generating Hermite data from polygonal meshes, and Li-Jung Chiu for narrating our video. This work was partially supported by the CIET-NTU(MOE) and National Science Council of Taiwan under NSC93-2622-E-002-033 and NSC93-2752-E-002-007-PAE.

References

[BKZ01] BIERMANN H., KRISTJANSSON D., ZORIN D.: Approximate boolean operations on free-form solids. In *Proc. of ACM SIGGRAPH 2001* (2001), pp. 185–194.

[Che95] CHERNYAEV E.: *Marching Cubes 33: Construction of Topologically Correct Isosurfaces*. Tech. Rep. CN/95-17, CERN, 1995.

[Gib98] GIBSON S. F. F.: Using distance maps for accurate surface representation in sampled volumes. In *Proc. of Symposium on Volume Visualization 1998* (1998), pp. 23–30.

[HSE99] HEIDRICH W., SEIDEL R. W. H.-P., ERTL T.: Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer 15*, 2 (1999), 100–111.

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. In *Proc. of ACM SIGGRAPH 2002* (2002), pp. 339–346.

[KBSS01] KOBBELT L. P., BOTSCH M., SCHWANECKE U., SEIDEL H.-P.: Feature sensitive surface extraction from volume data. In *Proc. of ACM SIGGRAPH 2001* (2001), pp. 57–66.

[LB03] LOPES A., BRODLIE K.: Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Transactions on Visualization & Computer Graphics 9*, 1 (2003), 16–29.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proc. of ACM SIGGRAPH 1987* (1987), pp. 163–169.

[LLVT03] LEWINER T., LOPES H., VIEIRA A. W., TAVARES G.: Efficient implementation of marching cubes' cases with topological guarantees. *Journal of Graphics Tools 8*, 2 (2003), 1–15.

[Nat94] NATARAJAN B. K.: On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer 11*, 1 (1994), 52–62.

[NH91] NIELSON G. M., HAMANN B.: The asymptotic decider: resolving the ambiguity in marching cubes. In *Proc. of IEEE Visualization 1991* (1991), pp. 83–91.

[Nie03] NIELSON G. M.: On marching cubes. *IEEE Transactions on Visualization & Computer Graphics 9*, 3 (2003), 283–297.

[RK02] RODEHORST M. J., KIMIA B. B.: Subvoxel polygonization of discrete implicit surfaces using ENO interpolation. unpublished manuscript, http://www.lems.brown.edu/∼mjr/paper.pdf, 2002.

[SCK95] SHU R., CHEN Z., KANKANHALLI M. S.: Adaptive marching cubes. *The Visual Computer 11*, 4 (1995), 202–217.

[SFYC96] SHEKHAR R., FAYYAD E., YAGEL R., CORNHILL J. F.: Octree-based decimation of marching cubes surfaces. In *Proc. of IEEE Visualization 1996* (1996), pp. 335–342.

[SKS97] SIDDIQI K., KIMIA B. B., SHU C.-W.: Geometric shock-capturing ENO schemes for subpixel interpolation, computation and curve evolution. *Graphical Models and Image Processing 59*, 5 (1997), 278–301.

[SW04] SCHAEFER S., WARREN J.: Dual marching cubes: Primal contouring of dual grids. In *Proc. of Pacific Graphics 2004* (2004), pp. 70–76.

[VKKM03] VARADHAN G., KRISHNAN S., KIM Y., MANOCHA D.: Feature-sensitive subdivision and isosurface reconstruction. *Proc. of IEEE Visualization 2003* (2003), 99– 106.

[VKSM04] VARADHAN G., KRISHNAN S., SRIRAM T., MANOCHA D.: Topology preserving surface extraction using adaptive subdivision. In *Proc. Symposium on Geometry Processing 2004* (2004).

[WG92] WILHELMS J., GELDER A. V.: Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 3 (1992), 201–227.

[ZHK04] ZHANG N., HONG W., KAUFMAN A.: Dual contouring with topology-preserving simplification using enhanced cell representation. In *Proc. of IEEE Visualization 2004* (2004), pp. 505–512.