Wang M, Feng JQ. 2D-manifold boundary surfaces extraction from heterogeneous object on GPU. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 27(4): 862–871 July 2012. DOI 10.1007/s11390-012-1270-8

2D-Manifold Boundary Surfaces Extraction from Heterogeneous Object on GPU

Ming Wang (王 明), *Member*, *CCF*, and Jie-Qing Feng* (冯结青), *Senior Member*, *CCF*

State Key Lab of CAD&CG, Zhejiang University, Hangzhou 310058, China

E-mail: {wangming, jqfeng}@cad.zju.edu.cn

Received November 25, 2011; revised May 2, 2012.

Abstract The conventional isosurface techniques are not competent for meshing a heterogeneous object because they assume that the object is homogeneous. Thus the visualization method taking the heterogeneity into account is desired. In this paper, we propose a novel algorithm to extract the boundary surfaces from a heterogeneous object in one pass, whose remarkable advantage is free of the number of materials contained. The heterogeneous object is first classified into a series of homogeneous material components. Then each component is enclosed with a 2D-manifold boundary surface extracted via our algorithm. The information important to the heterogeneous object is also provided, such as the interface between two materials, the intersection curve where three materials meet and the intersection point where four materials meet. To improve the performance, the algorithm is also designed and implemented on GPU. Experimental results demonstrate the effectiveness and efficiency of the proposed algorithm.

Keywords heterogeneous object, isosurface, interface, boundary surface, GPU

1 Introduction

Extracting the boundary surfaces from a heterogeneous object is an important technique in many practical applications such as anatomy, manufacture, geology. A heterogeneous object is a solid model composed of multiple homogeneous material components in general. To identify and represent its homogeneous components can facilitate the investigations of the heterogeneous object. For example, the surgeons can diagnose the illness and perform the operation more confidently by browsing the clearly distinguished parts, the engineers can re-design the machinery parts and improve their performances in a well-targeted manner by observing their internal structures and composition.

The heterogeneous object is represented mathematically or discretely in general. Its boundary surface representation is also preferable[1]. The conventional isosurface extraction methods are hardly adapted to the heterogeneous cases. Firstly, the "case table" will become too complex to be constructed if we mimic the Marching Cube algorithm[2]. For example, there will be 6 561 (3^8) items in the case table even if there are only three materials involved. Secondly, if the isosurface extraction algorithms were adopted, only one isosurface could be extracted in one pass, which encloses only one homogeneous material. Moreover, to display multiple isosurfaces clearly and completely in one image is difficult and tends to introduce artifacts and confusion.

Furthermore, another challenge in extracting the boundary surfaces from the heterogeneous object is to identify the interface or intersection information between or among different material components. Even if the interface between different materials may not be smooth, the extracting algorithm is required to definitely distinguish homogeneous components and clearly exhibit the boundary surface of each material component. As to adjacent material components, if their boundary surfaces are extracted componentwisely, the interface between adjacent components can only be obtained via surface-surface intersection[3]. It is a time-consuming procedure and hinders us from visualizing the heterogeneous object in real time. In addition, it is difficult to guarantee the topological consistency of the interface, i.e., the interface may not be 2D manifold.

In this paper, we propose an algorithm to extract boundary surfaces of homogeneous materials from a heterogeneous object in one pass, which is abbreviated

Regular Paper

This work is supported by the National Natural Science Foundation of China under Grant Nos. 60933007 and 61170138, the National Basic Research 973 Program of China under Grant No. 2009CB320801, the Program for New Century Excellent Talents in University of Ministry of Education of China under Grant No. NCET-10-0728.

*Corresponding Author

as BSHO. It generates 2D-manifold boundary surfaces and distinguishes different material components without ambiguity. A special hierarchical data structure is designed to organize the extracted surfaces, which contains the hierarchical information of heterogeneous object, i.e., *boundary surface*, *interface* and *separating point*. Here, the boundary surface is a 2D-mainfold surface enwraping the homogeneous component, the interface is a 2D-mainfold surface patch to separate two material components, the separating point is a point at which two or more materials meet. Fig.1 demonstrates our algorithm by using three examples. The BluntFin model consists of three materials, the Hand model contains blood vessel and bones, and the Torso model is filled with organs and bones. The main contributions of our algorithm include:

• An efficient method is proposed for extracting the boundary surfaces from a heterogeneous object in one pass, and the extracted boundary surfaces are guaranteed to be 2D-manifold.

• The intersection information between or among multiple materials is obtained simultaneously.

• A special data structure is designed for the boundary surfaces and the intersection information.

• A GPU implementation of the algorithm is also designed.

The rest of the paper is organized as follows. After describing related work in Section 2, we introduce our BSHO algorithm in Section 3, which includes extracting and rendering steps. The details of the GPU implementation is presented in Section 4. Section 5 is the experimental results and discussion, followed by the conclusions in Section 6.

2 Related Work

The proposed algorithm is closely related to isosurface extraction and heterogenous object modeling.

Isosurface extraction is one of the fundamental techniques for visualizing volume data. The Marching Cube algorithm (MC) is the representative one[2]. Even though the extensions of MC algorithm[4-5] are robust and efficient, they are not able to process the heterogeneous objects. Nielson[6] ever proposed a rough idea to process heterogeneous volume data. He divided a heterogeneous tetrahedron into homogeneous material parts by splitting its edges, faces and tetrahedron at mid-edge points, mid-face points and mid-tetrahedron point respectively. Due to lack of efficient data structure, only triangle soup is obtained to separate different materials. The method is still far from being matured for extracting the boundary surfaces from the heterogeneous objects consistently. First, the extracted surfaces are not 2D-manifold. Secondly, there is no global intersection information available between or among different materials. Thirdly, the time complexity of the algorithm is too high. Bonnell *et al.*[7] solved the problem from a purely mathematical viewpoint, which can generate similar results with those of Nielson's method[6]. Wu *et al.*[8] extended the MC algorithm to the multi-material case, namely multi-material marching cubes algorithm (M3C). The approach is to separate a cube which contains more than two face-centered nodes, which may lead to complex cases in polygon triangulation step. The smoothing step will produce pleasant visual effect but might cause volume shrinkage. In addition to primal methods, dual methods are also widely adopted in heterogeneous object surface reconstruction. Wang[9] not only modeled the heterogeneous object but also accelerated the modelling process by parallel computing. And the method proposed by Ju *et al.*[10], combining the Extended Marching Cubes method with dual method, creates polygons from points and normal data stored on edges in the grid. Feng *et al.*[11] combine the tri-linear contouring and volume rendering to display the heterogeneous object on GPU directly.

In computer-aided design community, representation and modeling of heterogeneous objects is also an important topic[12-13]. Pratt *et al.*[14] classified the representation methods into three types, i.e., exact boundary-based parameterizations of object interiors, volume discretization approaches and non-boundary conformant parameterization methods. Wang[1] meshed a heterogeneous object with 2D-manifold surfaces. However the

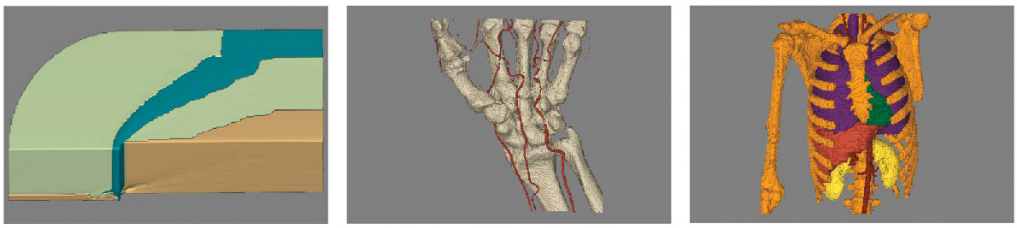


Fig.1. Examples of boundary surface extraction from heterogeneous objects. (a) The BluntFin consists of three materials. (b) The Hand contains blood vessels and bones. (c) The Torso contains multiple organs and bones.

intersection information between or among different materials cannot be obtained explicitly since each boundary surface is optimized separately. Shammaa *et al.*[15] combined region growing and graph-cut methods to classify the volumetric model into its component materials and adopted a generalized MC algorithm to generate triangulated mesh surfaces. They also provided a quality measure to evaluate the extracted isosurfaces. However, the extracted surfaces may be degraded when the number of materials is more than 3.

In general, the boundary surface extraction algorithms for heterogeneous objects are very space and time consuming. GPU provides us the opportunities to achieve performance gains. Pascucci[16] first used vertex shaders of programmable graphics hardware to extract the isosurface. Since the resulting geometries cannot be persistently stored, the extraction process has to be performed frame by frame. Klein *et al.*[17] and Kipfer *et al.*[18] gained significant improvements by taking advantage of the superbuffers provided by some ATI graphics cards. However the commonality and reusability of their algorithms are limited. Buatois *et al.*[19] obtained extra performance gains by storing isosurface in texture memories. With the advent of CUDA (Compute Unified Device Architecture)[20], the researchers are more convenient to exploit the parallel processing capabilities of modern GPUs[21]. To our knowledge, there is no attempt to mesh a heterogeneous object on GPU yet.

3 BSHO Algorithm

In our algorithm, the input is regular cubical scalar volume data. The volume data is segmented into different materials. Each grid point is assigned to a material index according to its scalar value. Our goal is to extract boundary surfaces so that each material is separated from others. Furthermore, the global intersection information between or among different materials are implied naturally in the extracted boundary surfaces. As described and analyzed above, it is extremely difficult to extract boundary surfaces from the cubical dataset directly. Alternatively, each cube is subdivided into six tetrahedra and the extraction is performed in each tetrahedron. A hierarchical data structure is carefully designed to record the 2D-manifold boundary surfaces information and the global intersection information.

3.1 Hierarchical Data Structure

Supposing there are n materials in the heterogeneous object $\mathscr{H}$, the classifying function $M(p)$ will return a material index between -1 and $n-1$ according to the scalar value $F(p)$ at the sample p, where -1 indicates the materials that we are not interested in. Let Ω_u be the boundary surface of the material u. Then there are totally $(n+1)$ boundary surfaces in $\mathscr{H}$. The boundary representation of the heterogeneous object $\mathscr{H}$ is defined as follows:

$$\mathscr{H} = \bigcup_{i=-1..n-1} \Omega_i. \tag{1}$$

Let $\Gamma_{(u,v)}$ be the interface between materials u and v. Obviously $\Gamma_{(u,v)} = \Gamma_{(v,u)}$. There are total $((n+1) \times (n+2)/2)$ interfaces in $\mathscr{H}$ at most. Then the boundary surface of material $u(u \in [0, n-1])$ can be represented as the union of related interfaces:

$$\Omega_u = \bigcup_{i=-1..n-1, i\neq u} \Gamma_{(u,i)}. \tag{2}$$

Inferring from (2), it would be a curve segment $\psi_{(u,v,w)}$ where three materials u, v and w meet:

$$\begin{aligned}\psi_{(u,v,w)} &= \Omega_u \bigcap \Omega_v \bigcap \Omega_w = \Gamma_{(u,v)} \bigcap \Gamma_{(v,w)} \\ &= \Gamma_{(u,v)} \bigcap \Gamma_{(u,w)} = \Gamma_{(u,w)} \bigcap \Gamma_{(v,w)}.\end{aligned} \tag{3}$$

Similarly, it would be an intersection point where four materials meet. The intersection point can also be described as the intersection among three interfaces as follows:

$$\begin{aligned}\vartheta_{(u,v,w,z)} &= \Omega_u \bigcap \Omega_v \bigcap \Omega_w \bigcap \Omega_z \\ &= \Gamma_{(u,v)} \bigcap \Gamma_{(u,w)} \bigcap \Gamma_{(u,z)} \\ &= \Gamma_{(u,v)} \bigcap \Gamma_{(v,w)} \bigcap \Gamma_{(v,z)} \\ &= \Gamma_{(u,w)} \bigcap \Gamma_{(v,w)} \bigcap \Gamma_{(w,z)} \\ &= \Gamma_{(u,z)} \bigcap \Gamma_{(v,z)} \bigcap \Gamma_{(w,z)}.\end{aligned} \tag{4}$$

According to above analysis, we can draw a conclusion that the interface is the key element to our boundary surfaces extraction algorithm. An interface can be fully defined by separating points on or in the tetrahedron. The definition and extraction of separating point will be introduced in detail in the following subsection. The pseudocode of our BSHO algorithm is shown in Algorithm 1.

Algorithm 1

```
1: Input: Segmented volume data
2: for each cube do
3:     Subdivide the cube into six tetrahedra;
4:     for each tetrahedron do
5:         Generate case index of the tetrahedron;
6:         Extract and compute separating points according to the case index;
7:         Triangulate the separating points to form separating triangles;
```

8: **end for**
9: **end for**
10: Combine shared separating points for preserving 2D manifold;
11: Calculate normal vector for each separating point;
12: Group separating triangles to form interfaces;
13: Group interfaces to form boundary surfaces;

3.2 Extracting Separating Points from Tetrahedron

A tetrahedron can be subdivided into parts with triangular meshes so that each part is of homogeneous material. Similar to MC-like methods, we assume that there will be a bi-point on the tetrahedral edge if its two end points are of two materials. Similarly, there will be a tri-point in a tetrahedral face if its three vertices are of three materials and a tetra-point in a tetrahedral volume if the tetrahedral vertices are of three or four materials. As shown in Fig.2(d), the points $4 \sim 9$ are bi-points, and the points $10 \sim 13$ are tri-points and the point 14 is a tetra-point. All of them are called separating points, whose data structure is described in Algorithm 2 as follows:

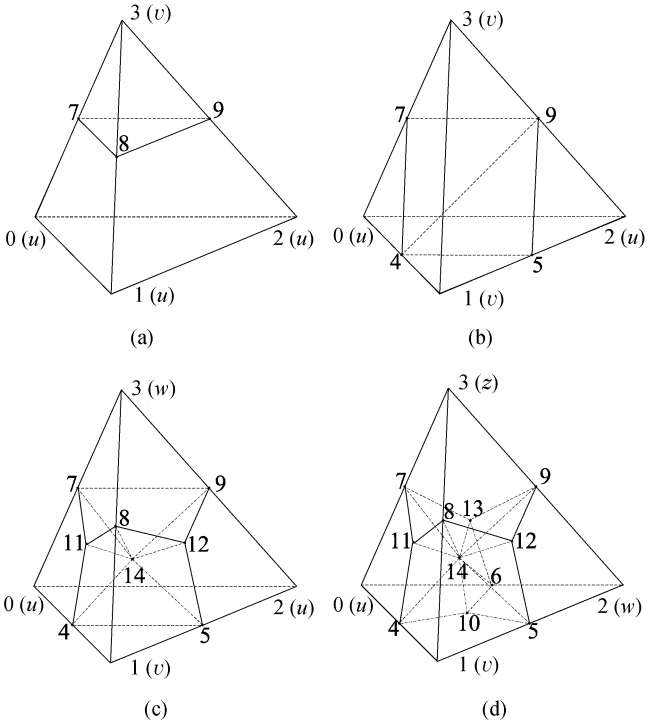


Fig.2. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4. u, v, w and z at the tetrahedron corners are material indices. Except for the homogeneous case, there are still four cases, which are of more than one material and should be subdivided. Two-material tetrahedron can be subdivided into two parts by one (a) or two (b) triangles, while three-material tetrahedron by 8 triangles (c), and four-material tetrahedron by 12 triangles (d).

Algorithm 2.

```
1:  //Data structure of a separating point
2:  struct VERTEXINFO {
3:    float 4 fPos;          //the position of the point
4:    float 4 fNormal[4];    //the normals for materials
5:    unit nMaterial[4];     //the material indices
6:    unit nCount[4];        //the number of triangles
    }
```

Each vertex of the tetrahedron can be classified and assigned as one of material indices. Except for the homogeneous case and other configurations which can be reached by rotation and symmetry, there are still four cases need to be considered. The four cases and their separating triangles are shown in Fig.2. Each interior separating triangle is defined by three separating points. The information of the separating triangle, such as the separating points and material types at both sides of the triangle, will be recorded. They are very important for us to generate 2D-manifold boundary surfaces. We distil a data structure of the triangle as a vector of quintuple elements in (5), where P_0, P_1, P_2 are indices of the separating points, while mf and mb are the material indices of the front side and back side of the triangle respectively.

$$\boldsymbol{\Delta} : (P_0, P_1, P_2, mf, mb). \tag{5}$$

We record three separating points counterclockwise such that the normal direction of the triangle is from high index material to low index material. For example, the case in Fig.2(a), if $u < v$, the separating triangle will be recorded as $(7, 9, 8, u, v)$, otherwise $(7, 8, 9, v, u)$.

The position of a separating point is calculated via linear interpolation. In general, a material type is not determined by an exact scalar value, but a scalar range. In our paper, we will not address the problem on how to choose these scalar ranges, i.e., the volume data segmentation problem. Supposing the ascending thresholds are given as $\lambda_0 = -\infty, \lambda_1, \ldots, \lambda_N = +\infty$, the sample point $p(l, m, n)$ can be classified as follows:

$$M(p) = i \;\Leftrightarrow\; \lambda_{i-1} < F(p(l, m, n)) \leqslant \lambda_i, \tag{6}$$

where (l, m, n) is the grid index of sample p, and $F(p(l, m, n))$ is the scalar value at sample p. In the case that two neighboring samples, i.e., $p(l, m, n)$ and $p(l + 1, m, n))$ are classified as materials i and $(i + 1)$ according to (6), there will be a bi-point generated at $(l + \delta, m, n)$ where δ can be simply assigned as 0.5 or calculated by the following equation.

$$\delta = \frac{\lambda - F(p(l, m, n))}{F(p(l + 1, m, n)) - F(p(l, m, n))}, \tag{7}$$

where $\lambda = \lambda_i$. If the two neighboring samples are classified as i and j $(j > i + 1)$ type materials respectively,

the threshold λ in (7) can be chosen as $\frac{\lambda_i+\lambda_{j-1}}{2}$ or any values in the interval $(\lambda_i, \lambda_{j-1}]$. As for the tri-points or the tetra-points, we compute their positions as the centroid of the related bi-points or the related tri-points, respectively.

3.3 Interfaces

After extracting the interior separating triangles from the tetrahedra, the interfaces can be obtained by clustering and grouping these triangles. An interface is defined as the interior separating triangles set:

$$\Gamma_{(u,v)} = \begin{cases} \{\boldsymbol{\Delta}s | mf = u, mb = v\}, & \text{if } u < v, \\ \{\boldsymbol{\Delta}s | mf = v, mb = u\}, & \text{if } v < u. \end{cases} \quad (8)$$

From above definition, we know that each triangle will be applied twice, i.e., one for the material front and the other for the back material. If we assigned an average normal to each vertex as in the MC-like algorithms, the rendering of the boundary surfaces would be confused. To avoid this circumstance, we will introduce a new approach to calculate the vertex normal for each interface.

3.3.1 Normal Calculation

In a tetrahedron, the boundary surface patch corresponding to the material is the interface, which is composed of corresponding interior separating triangles. However, the interior separating triangles sharing a separating point may be in different tetrahedra or even in different cubes. To display an interface properly and correctly, we compute the normal vectors of its separating points on-the-fly, which can avoid the huge amount queries of neighboring triangles in the same interface.

Traditionally, the vertex normal in an isosurface can be computed as the average normal of triangles incident to the vertex. However, the solution will cause confused rendering result for the heterogeneous object case. To address this problem, Wang[1] duplicated the intermediate surfaces and Feng[11] set extracted interfaces opaque in volume rendering. In order to represent the heterogeneous object without ambiguity, we will calculate multiple normal vectors for each separating point. Each normal vector corresponds to one material, i.e., one boundary surface. It is an average normal of the triangles, which are incident to the same separating point and in the same boundary surface. As to the interior separating triangle $\boldsymbol{\Delta} : (P_0, P_1, P_2, mf, mb)$, its normal corresponding to material u is

$$\begin{cases} N_{(\boldsymbol{\Delta},mf)} = \overrightarrow{P_2P_0} \times \overrightarrow{P_0P_1}, & \text{if } mf = u, \\ N_{(\boldsymbol{\Delta},mb)} = \overrightarrow{P_0P_1} \times \overrightarrow{P_2P_0}, & \text{if } mb = u. \end{cases} \quad (9)$$

The normal vector is then accumulated for each separating point. The one-to-one relationship between the normal vector and its material type is accomplished by keeping the same order of the normal vector in the *fNormal* array and the material type in *nMaterial* array in its VERTEXINFO in Algorithm 2. Finally, the number of interior separating triangles incident to the separating point is recorded in the $nCount[i]$. The separating point's normal vector of the boundary surface for material u is obtained by dividing $fNormal[i]$ by $nCount[i]$.

3.3.2 Rendering Scheme

According to above description, the boundary representation of a heterogeneous model is the union of the boundary surfaces of different materials. Each boundary surface is again the union of the interfaces related to the material. Thus properly rendering of the interfaces is important to reveal the structure of the heterogeneous object. Let $\Gamma_{(u,v)}$ be the interface between materials u and v. It is rendered twice, one with the normals about material u and one with v. Our rendering scheme can provide the users the adjacent materials information immediately and intuitively.

3.4 2D Manifold Preservation of Boundary Surfaces

Supposing the heterogeneous object $\mathscr{H}$ has $(L+1)\times(M+1)\times(N+1)$ samples and $L\times M\times N$ domain cubes. Each cube is subdivided into 6 tetrahedra as shown in Fig.3.

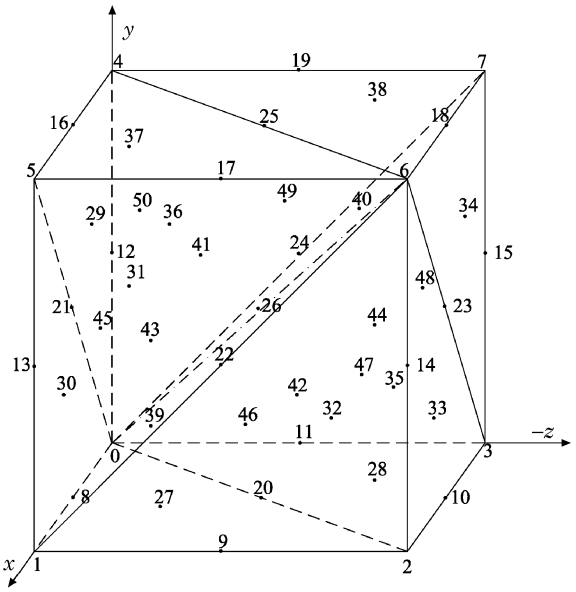


Fig.3. A cube is subdivided into six tetrahedra. The points labeled from 8 to 50 are separating points.

After extraction of the interior separating triangles from tetrahedra, the triangle soup approximating the heterogeneous object boundaries is obtained. Our aim is to traverse the triangle soup and check the shared separating points of the triangles so that the 2D-manifold boundary surface for each material is generated. The rules of generating 2D-manifold interfaces and boundary surfaces from the interior separating triangles are described as follows:

• *Merge the Separating Points Shared by Neighboring Cubes.* The separating points on the cube edge shared by the neighboring cubes will be merged. Only the point with the smallest label should be preserved. For example, vertex 13 in Cube(i, j, k) will be merged and replaced with vertex 12 in Cube$(i+1, j, k)$.

• *Merge the points shared by neighboring tetrahedra in a cube.* The separating points on the tetrahedron edge shared by the neighboring tetrahedra in a cube will be merged. Only the point in the tetrahedron with the smallest label should be preserved as shown in Table 1. In Table 1, $T_{i=0}^{5}$ are six tetrahedra in the cube, while $V_{i=0}^{14}$ are vertices in a tetrahedron as shown in Fig.2(d). The entries in Table 1 are the indices of vertices in a cube as shown in Fig.3. For example, the vertex V_4 in T_3, labeled 11 in cube, will be merged and replaced by the vertex V_6 in T_2.

The first rule takes priority over the second one.

Through above separating points merging rules, the generated boundary surface for each material is 2D-manifold, in which there is no redundant vertex.

4 GPU Implementation Details

As described above, the proposed BSHO algorithm is well compatible with the parallel computing model. Thus it can be accelerated by general-purpose GPU.

4.1 Software & Hardware Condition

CUDA presents a general-purpose parallel computing architecture[20]. It provides APIs to access the Vertex Buffer Object (VBO) directly. Therefore, our algorithm is implemented with CUDA on an NVIDIA Quadro FX 5800 GPU which has 4 G video memory and 240 CUDA parallel processing cores. Since it is the most efficient way to render the triangular mesh by combining VBO with shaders in OpenGL, we store the extracted boundary surface meshes in VBOs.

4.2 GPU Implementation

There are a lot of GPU-based isosurface extraction algorithms. However, none of them is related to representation of heterogeneous object.

To generate 2D-manifold boundary surfaces, a merging operator is defined for the separating points. The auxiliary arrays involved are CAV, $V2P$ and VA, which are shown in Fig.4. CAV records the indices of tetrahedra containing interfaces, $V2P$ records the addresses of separating points, and VA indicates the first separating point in a tetrahedron in $V2P$. The GPU implementation details are described as follows:

Table 1. Separating Points Merging Lookup

Index	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}	V_{13}	V_{14}
T_0	0	5	1	6	21	13	8	26	17	22	30	43	31	39	45
T_1	0	1	2	6	8	9	20	26	22	14	27	39	32	42	46
T_2	0	2	3	6	20	10	11	26	14	23	28	42	33	44	47
T_3	0	3	7	6	11	15	24	26	23	18	35	44	34	40	48
T_4	0	7	4	6	24	19	12	26	18	25	36	40	38	41	49
T_5	0	4	5	6	12	16	21	26	25	17	29	41	37	43	50

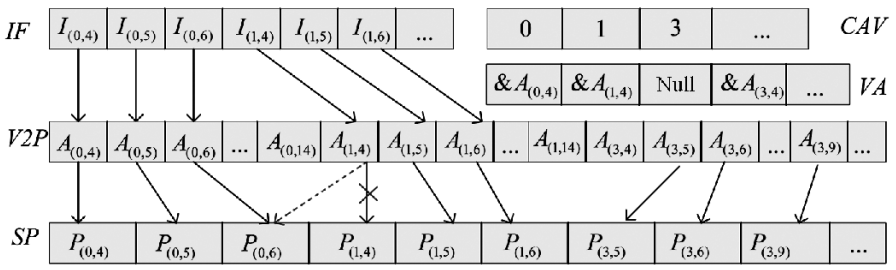


Fig.4. Frame of our kernels on GPU. (a) IF is the index VBO. It records the separating points' locations in $V2P$ before performing merging operation, and records the element indices after performing merging operation. (b) The dotted arrow means a merging operation. It indicates the 4th separating point $P_{(1,4)}$ in Tetrahedron 1 and the 6th separating point $P_{(0,6)}$ in Tetrahedron 0 will be merged as $P_{(0,6)}$.

• *Preprocessing* (CPU side). The heterogeneous object dataset is copied from main memory to video memory via cudaMemcpy function. Then several VBOs (Vertex Buffer Objects) are allocated on the video memory. As shown in Fig.4, the VBO *SP* with vertex array buffer is for preserving the separating points information, the other VBOs with element array buffer are for recording the indices to the *SP* elements in the interfaces. Each VBO can be efficiently accessed by mapping a pointer to its buffer via the cudaGraphicsResourceGetMappedPointer function. The video memories for auxiliary data structures are also allocated in advance. At last, the case table, listed in Appendix, is binded on a constant memory via the cudaMemcpyToSymbol function.

• *Classify and Compact* (GPU side). All the tetrahedra are numbered in an ascending order. It will be video memory consuming to process all the tetrahedra, which is still precious for modern GPU. Thus, we only record the indices of tetrahedron containing more than one materials in *CAV* array. By coding the material indices of tetrahedron vertices in ascending order starting from 1, we can obtain an unsigned integer case flag. In this way, all the tetrahedron can be classified into 15 cases no matter how many materials the heterogeneous object involves. The case table of 15 items could be found in Appendix.

• *Generate Triangles* (GPU side). The separating points which define the interior subdivision triangles are recorded in the *SP* array, while the addresses of these points are recorded in the $V2P$ array. As shown in Fig.4, in the $V2P$ array, there are 11 elements corresponding to a tetrahedron since a tetrahedron contains 11 separating points at most. The interior subdivision triangles are generated according to the case table in Appendix. The VBO for an interface (*IF*) collects its triangles by recording their vertices locations in the $V2P$ array instead of the *SP* array.

• *Merge Shared Separating Points* (GPU side). Each element in the $V2P$ array will be assigned to a thread. The thread will merge the corresponding separating points shared by cubes and by tetrahedra according to the two merging rules in Subsection 3.4. Here the GPUs with computing capability 1.1 or higher and supporting atomic operator are required because the normal and count accumulation operations should be executed exclusively.

• *Average of Normal Vectors* (GPU side). Each separating point in *SP* has four normal vectors at most. Each normal in each *SP* element corresponds to a material. It is an average of the normals of triangles related to the material.

• *Calculate Indices* (GPU side). Till now, the number in an element of the interface VBO is a location recording the address of the corresponding separating point. It should be updated as the index of the *SP* element. We first obtain the address of the corresponding separating point via $V2P$ array. Then we compute the index by dividing the address offset from the start address of *SP* by the size of *SP*'s element.

• *Vertex Shader* (GPU side). The boundary surface to the specified material is rendered via openGL pipeline. The proper normal is selected via vertex shader.

5 Experimental Results and Discussion

Our BSHO algorithm has been implemented with C++, CUDA3.2 and OpenGL on a PC with Intel Core i5-2300 2.8 GHz CPU, 8 GB main memory and NVIDIA Quadro FX 5800 GPU. Among the examples, both real and synthetic datasets are adopted.

The BluntFin dataset is the "teapot of CFD visualization", which was provided by Hung and Buning[22] and widely adopted as the test dataset in geometric and visualization communities. The dataset adopted in our implementation is a re-sampled version which is freely available at http://www9.informatik.uni-erlangen.de/External/vollib/. As shown in Fig.5, the BluntFin (Fig.5(a)) is composed of three materials, i.e., brown material (Fig.5(b)), olive material (Fig.5(c)) and teal material (Fig.5(d)).

The sample value of the brown material is in the range $(10, 35]$, olive material in $(35, 60]$ and teal

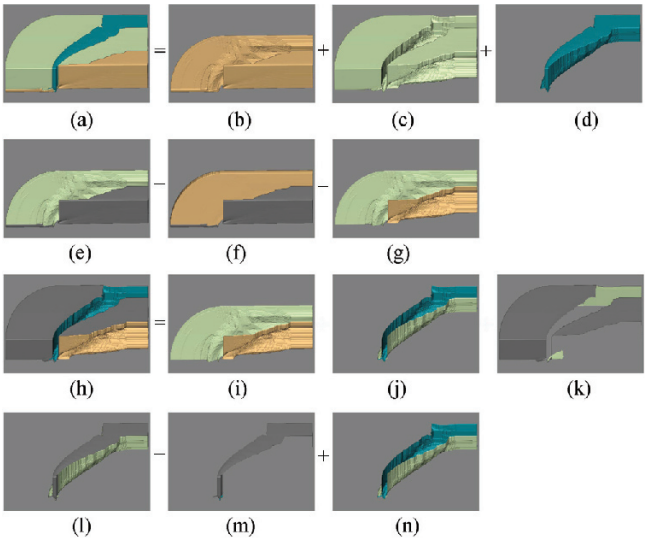


Fig.5. BluntFin dataset (a) is composed of three materials (b), (c), and (d). The boundary surface of each material is formed by merging related interfaces. We can obtain not only the boundary information of each component but also the adjacent component information using our BSHO algorithm.

material in $(60, 255]$ respectively. We first generate the interfaces between the materials with the thresholds $\{10, 35, 60, 255\}$ and then group the related interfaces to form boundary surfaces. Let "b", "o" and "t" be noted as the brown, olive, teal materials and "-1" be the material that we are not interested in. As shown in Fig.5, there are five interfaces generated in total. They are $\Gamma_{(b,-1)}$ (Fig.5(f)), $\Gamma_{(b,o)}$ (Fig.5(g)), $\Gamma_{(o,t)}$ (Fig.5(j)), $\Gamma_{(0,-1)}$ (Fig.5(k)), and $\Gamma_{(t,-1)}$ (Fig.5(m)), respectively. The boundary surface of olive material, Ω_o (Fig.5(h)), is composed of the interfaces $\Gamma_{(o,-1)}$, $\Gamma_{(o,b)}$ and $\Gamma_{(o,t)}$. Thus Ω_o directly represents the boundary surface of olive-material component and also intuitively provides the adjacent material information. Ω_b and Ω_t can also be obtained similarly by merging their related interfaces. Fig.6 is the results of MC method with isovalues 10, 35 and 60 respectively. It is evident that the MC method cannot extract the boundary surfaces of different materials thoroughly. Furthermore, it cannot provide the interface information of adjacent materials either.

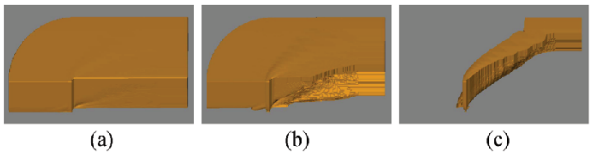


Fig.6. Isosurfaces in (a), (b), and (c) are extracted by the MC algorithm with isovalues 10, 35 and 60 respectively. The MC algorithm is not able to extract multiple boundary surfaces in one pass. It cannot extract the boundary surface of the olive material as shown in Fig.5(c) either.

Fig.7 gives an example of synthetic dataset, which is composed of three material components and is surrounded by air. The thresholds for the three materials are $\{20, 60, 110, 160\}$. As shown in Fig.7(a), three material components meet or intersect at red curves, which can be obtained by connecting the related tri-points and tetra-points. Four materials, including three materials and the air, meet at yellow points. Fig.7(f) is the local zooming in of the area marked by blue rectangle in Fig.7(e). The sharp features on the boundary surface cannot be preserved probably. It is the weakness of the BSHO algorithm.

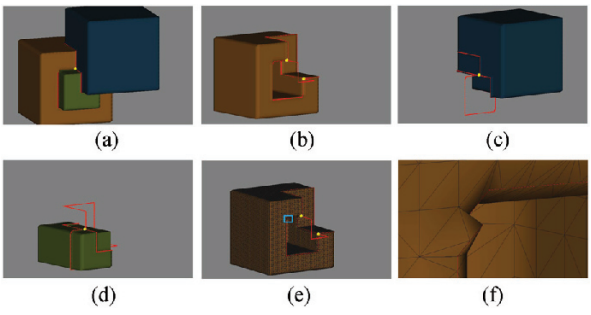


Fig.7. The heterogeneous object (a) is composed of three materials (b), (c), and (d). It is surrounded by air, which is the uninterested material. Three of four materials meet at red curves and four materials meet at yellow point. (f) is the zoom in of the are of marked by blue rectangle of (e), where the sharp features are not well extracted.

The third example is a hand model (Fig.8), where the vessel and bone are segmented in advance. The boundary surfaces of vein and the bone are extracted.

Fig.8. Vein (a) and bone (b) are two components of the hand model.

The last dataset is the Utah torso model. It is a tetrahedral volumetric mesh. The conductivity tensors are stored with the elements. The details about the model can refer to [23]. We get the sample values of conductivity by SCIRun software[24]. As shown in Fig.1(c), it contains much more organs than hand. In our implementation, six organs are extracted simultaneously in one pass. As shown in Fig.9, they are liver, kidney, lung, bone, artery, and heart.

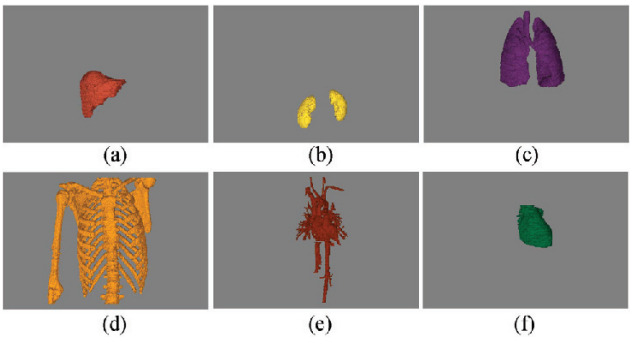


Fig.9. Utah torso model contains liver (a), kidney (b), lung (c), bone (d), artery (e) and heart (g).

The statistic of the examples is listed in Table 2. In general, our GPU implementation is faster than the

Table 2. Details of Our Test

Dataset	Materials	Size	Triangles	CPU (s)	GPU (s)
BluntFin	3	$256 \times 128 \times 64$	961 350	42.36	1.62
Synthetic	3	$64 \times 64 \times 64$	42 848	0.07	0.08
Hand	2	$256 \times 128 \times 256$	1 420 234	11.01	2.45
Torso	6	$199 \times 199 \times 249$	2 262 800	34.60	3.78

CPU implementation one order of magnitude except for the synthetic dataset since the size of the dataset is very small. According to our experiments, it will take much more time to process the dataset which contains more tri-points and tetra-points. It is the reason why the BluntFin model is the most time-consuming on CPU. In general, for a dataset of $(L+1)\times(M+1)\times(N+1)$ samples and J materials involved, the time complexity of the BSHO algorithm is $O(L\times M\times N\times J)$. Thus the runtime is dependent on the dimension of the dataset and the number of materials involved. Another aspect is the space complexity. The proposed BSHO algorithm requires much more memory than the MC-like algorithms. Because there are multiple boundary surfaces are extracted in one pass so the stage of extraction separating points will makes heavy use of the video memory. For example, the Utah torso dataset needs nearly 2 G video memory.

6 Conclusions

In this paper, we propose a novel algorithm for extracting the boundary surfaces of different materials from a heterogeneous object. The boundary surfaces are 2D-manifold. The intersection information among different materials are explicitly recorded in the interfaces. The GPU implementation of the algorithm is also given.

In future, we will improve our algorithm from two aspects. One is to adopt more reasonable material classification approach. The other is to compact the extracted results. We will also extend the algorithm to the cubical case directly.

Acknowledgment The authors would like to thank Professor Wei Chen who generously provided the Hand dataset and Jeroen Stinstra who provided us Utah torso volume data processed by SCIRun software.

References

[1] Wang C C L. Direct extraction of surface meshes from implicitly represented heterogeneous volumes. *Computer-Aided Design*, 2007, 39(1): 35-50.

[2] Lorensen W E, Cline H E. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 1987, 21(4): 163-169.

[3] Moller T. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 1997, 2(2): 25-30.

[4] Nielson G M. On marching cubes. *IEEE Transactions on Visualization and Computer Graphics*, 2003, 9(3): 283-297.

[5] Rosenthal P, Linsen L. Direct isosurface extraction from scattered volume data. In *Proc. Eurograhpics/IEEE-VGTC Symposium on Visualization*, May 2006, pp.99-106.

[6] Nielson G, Franke R. Computing the separating surface for segmented data. In *Proc. the 8th Conference on Visualization*, Oct. 1997, pp.229-233.

[7] Bonnell K S, Joy K I, Hamann B, Schikore D R, Duchaineau M. Constructing material interfaces from data sets with volume-fraction information. In *Proc. the Conference on Visualization 2000*, Oct. 2000, pp.367-372.

[8] Wu Z, Sullivan J M. Multiple material marching cubes algorithm. *International Journal for Numerical Methods in Engineering*, 2003, 58(2): 189-207.

[9] Wang C C L. Computing on rays: A parallel approach for surface mesh modeling from multi-material volumetric data. *Computers in Industry*, 2011, 62(7): 660-671.

[10] Ju T, Losasso F, Schaefer S, Warren J. Dual contouring of hermite data. In *ACM Trans. Graph.*, 2002, 21(3): 339-346.

[11] Feng P, Jn T, Warren J. Piecewise tri-linear contouring for multi-material volumes. In *Proc. the 6th International Conference on Advances in Geometric Modeling and Processing*, June 2010, pp.43-56.

[12] Patil L, Dutta D, Bhatt A D *et al.* Representation of heterogeneous objects in ISO 10303 (STEP). In *Proc. the ASME International Mechanical Engineering Congress and Exposition*, Nov. 2000, pp.355-363.

[13] Jackson T R, Cho W, Partikalakis N M *et al.* Memory analysis of solid model representations for heterogeneous objects. *Journal of Computing and Information Science in Engineering*, 2002, 2(1): 1-10.

[14] Pratt M J, Bhatt A D, Lyons K W *et al.* Progress towards an international standard for data transfer in rapid prototyping and layered manufacturing. *Computer-Aided Design*, December 2002, 34(14): 1111-1121.

[15] Shammaa M H, Suzuki H, Ohtake Y. Extraction of isosurfaces from multi-material CT volumetric data of mechanical parts. In *Proc. the 2008 ACM Symposium on Solid and Physical Modeling*, June 2008, pp.213-220.

[16] Pascucci V. Isosurface computation made simple: Hardware acceleration, adaptive refinement and tetrahedral stripping. In *Proc. IEEE TVCG Symposium on Visualization*, May 2004, pp.293-300.

[17] Klein T, Stegmaier S, Ertl T. Hardware-accelerated reconstruction of polygonal isosurface representations on unstructured grids. In *Proc. the 12th Pacific Conference on the Computer Graphics and Applications*, Oct. 2004, pp.186-195.

[18] Kipfer P, Westermann R. GPU construction and transparent rendering of iso-surfaces. In *Proc. the Modeling and Visualization*, 2005, pp.241-248.

[19] Buatois L, Caumon G, Lévy B. GPU accelerated isosurface extraction on tetrahedral grids. In *Proc. the 2nd International Conference on Advances in Visual Computing*, Part I, Nov. 2006, pp.383-392.

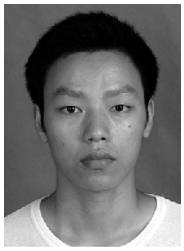
[20] NVIDIA Corporation. NVIDIA CUDA C Programming Guide: Version 3.2. NVIDIA Corporation, 2010.

[21] Dias S, Bora K, Gomes A. CUDA-based triangulations of convolution molecular surfaces. In *Proc. the 19th ACM International Symposium on High Performance Distributed Computing*, June 2001, pp.531-540.

[22] Hung C M, Buning P G. Simulation of blunt-fin-induced shock-wave and turbulent boundary-layer separation. *Journal of Fluid Mechanics*, 1984, 154: 163-185.

[23] MacLeod R S, Johnson C R, Ershler P R. Construction of an inhomegeneous model of the human torso for use in computational electrocardiography. In *Proc. the 13th Annual International Conference of IEEE Engineering in Medicine and Biology Society*, Oct. 31-Nov. 3, 1991, pp.688-689.

[24] Scientific Computing and Imaging Institute (SCI) of Utah University. SCIRun: A scientific computing problem solving environment. http://www.scirun.org.



Ming Wang obtained his Master's degree in computer science from the Hangzhou Dianzi University in 2008, and is currently a computer science Ph.D. candidate in the State Key Lab of CAD&CG, Zhejiang University, China. His current research focuses on digital geometry processing.

Jie-Qing Feng is a professor in the State Key Lab of CAD&CG, Zhejiang University, China. He received his B.Sc. degree in applied mathematics from the National University of Defense Technology in 1992 and his Ph.D. degree in computer graphics from Zhejiang University in 1997. His research interests include geometric modeling, real-time rendering and computer animation.

Appendix

In theory, there are total 4^4 cases for a heterogeneous tetrahedron. Only five cases need to be considered, since the other cases can be reached via rotation and symmetry. As a compromise between theory and implementation, we classify the cases into 15 items and list them as below. Each item is a quintuple-number vector in the form of $\{P_0, P_1, P_2, mf, mb\}$. P_0, P_1 and P_2 are the indices of separating points in the tetrahedron as shown in Fig.2(d). The normal vector for material *mf* of each separating point is calculated by $\overrightarrow{P_2P_0} \times \overrightarrow{P_0P_1}$, while $\overrightarrow{P_0P_1} \times \overrightarrow{P_2P_0}$ for *mb*. The 15 items are listed as below:

- $0x1111$:
  None;
- $0x1112$:
  {7, 9, 8, 0, 3};
- $0x1121$:
  {5, 9, 6, 0, 2};
- $0x1211$:
  {4, 8, 5, 0, 1};
- $0x1122$:
  {5, 8, 7, 1, 2};
- $0x1212$:
  {4, 7, 9, 0, 1}, {4, 9, 5, 0, 1};
- $0x1221$:
  {4, 8, 9, 0, 1}, {4, 9, 6, 0, 1};
- $0x1222$:
  {4, 7, 6, 0, 1};
- $0x1123$:
  {12, 14, 5, 0, 2}, {12, 8, 14, 0, 3},
  {12, 14, 9, 2, 3}, {13, 6, 14, 0, 2}, {13, 14, 7, 0, 3},
  {13, 9, 14, 2, 3}, {14, 6, 5, 0, 2}, {14, 8, 7, 0, 3};
- $0x1213$:
  {11, 14, 4, 0, 1}, {11, 7, 14, 0, 3},
  {11, 14, 8, 1, 3}, {12, 5, 14, 2, 1}, {12, 14, 9, 2, 3},
  {12, 14, 8, 3, 1}, {14, 5, 4, 0, 1}, {14, 7, 9, 0, 3};
- $0x1231$:
  {10, 4, 14, 0, 1}, {10, 14, 6, 0, 2},
  {10, 5, 14, 1, 2}, {12, 14, 8, 3, 1}, {12, 9, 14, 3, 2},
  {12, 14, 5, 1, 2}, {14, 4, 8, 0, 1}, {14, 9, 6, 0, 2};
- $0x1223$:
  {11, 14, 4, 0, 1}, {11, 7, 14, 0, 3},
  {11, 14, 8, 1, 3}, {13, 6, 14, 0, 2}, {13, 14, 7, 0, 3},
  {13, 9, 14, 2, 3}, {14, 6, 4, 0, 1}, {14, 9, 8, 2, 3};
- $0x1232$:
  {10, 4, 14, 0, 1}, {10, 14, 6, 0, 2},
  {10, 5, 14, 1, 2}, {13, 14, 7, 0, 3}, {13, 6, 14, 0, 2},
  {13, 14, 9, 3, 2}, {14, 4, 7, 0, 1}, {14, 5, 9, 1, 2};
- $0x1233$:
  {10, 4, 14, 0, 1}, {10, 14, 6, 0, 2},
  {10, 5, 14, 1, 2}, {11, 14, 4, 0, 1}, {11, 7, 14, 0, 3},
  {11, 8, 14, 3, 1}, {14, 7, 6, 0, 2}, {14, 5, 8, 1, 2};
- $0x1234$:
  {10, 4, 14, 0, 1}, {10, 14, 6, 0, 2},
  {10, 5, 14, 1, 2}, {11, 14, 4, 0, 1}, {11, 7, 14, 0, 3},
  {11, 14, 8, 1, 3}, {12, 14, 5, 1, 2}, {12, 8, 14, 1, 3},
  {12, 14, 9, 2, 3}, {13, 14, 7, 0, 3}, {13, 9, 14, 2, 3},
  {13, 6, 14, 0, 2}