Robust and Error Controllable Boolean Operations on Free-Form Solids Represented by Catmull-Clark Subdivision Surfaces

Shuhua Lai[1] and Fuhua (Frank) Cheng[2]

[1]Virginia State University, slai@vsu.edu
[2]University of Kentucky, cheng@cs.uky.edu

ABSTRACT

A method for performing robust and error controllable Boolean operations on free-form solids represented by Catmull-Clark subdivision surfaces (CCSSs) is presented. The method is voxelization based but, different from previous voxelization based approaches, a continuous geometric representation is provided for the result of each Boolean operation. This is achieved by doing the Boolean operations in the parameter spaces of the solids, instead of the object space. This approach allows us to easily compute a parametric approximation of the intersection curve and, consequently, build a continuous geometric representation of the Boolean operation result. The new method also allows secondary local voxelization to be performed on intersecting subpatches to increase accuracy of the Boolean operation result. Because voxelization process of the new method is very fast and robust, the overall process is fast and robust. Most importantly, error of each Boolean operation result can be precisely estimated, hence error control is possible. The new method can handle any types of Boolean operations as long as the given solids are represented by CCSSs. Hence there are no special or degenerated cases to take care of. The new method is presented for CCSSs only, but the concept works for any subdivision scheme whose limit surfaces are parametrizable.

Keywords: Catmull-Clark subdivision surfaces, voxelization, Boolean operations.

1. INTRODUCTION

Boolean operations are a nature way to construct complex solid objects from simple primitives. For example, the *Constructive Solid Geometry* (CSG) representation scheme allows a user to define complex 3D solid objects by hierarchically combining simple geometric primitives using Boolean operations and affine transformations. However, for many applications CSG is not the most efficient approach. Another major representation scheme used in solid modeling is *boundary representation* (B-rep). A major problem with B-rep is: it is difficult to find the intersection curve analytically for complicated objects. Besides, cares always have to be taken to handle special and degenerated cases [16]. Hence, accurate Boolean operations usually are neither fast nor robust, although excellent results have been achieved by some commercial solid modeling engines.

Voxelization of 3D objects has been studied and used for object modeling and rendering for a while [10-12]. With voxelization, it is actually a very simple task to perform Boolean operations on free-from solids because Boolean operations now become simple set operations on voxels. The difficult part is finding a proper and accurate representation for the resulting object of a Boolean operation. Traditionally, the resulting object is represented as a set of voxels [24, 25] and special volumetric rendering algorithms are developed to visualize this set of voxels [14, 27]. The main disadvantage of this approach is that the representation is not continuous and, therefore, the result cannot be scaled seamlessly and smoothly.

Most previous methods in this area work on solids represented by spline or NURBS surfaces, with one [28] on solids represented by subdivision surfaces. A subdivision surface based representation is more powerful because subdivision surfaces have the capability of representing any shape with only one surface, no matter how complicated the geometry or topology [1]. In this paper a method for performing robust and error controllable Boolean operations on free-form solids represented by Catmull-Clark subdivision surfaces (CCSSs) is presented. Different from the approach of using a multi-resolution surface to approximate the Boolean operation result [28], the given objects in the new method are voxelized to make Boolean operations more efficient. However, a continuous geometric

representation is provided for the result of each Boolean operation. This is achieved by performing Boolean operations subpatch by subpatch in 2D parameter space. Each subpatch is small enough to ensure the resulting voxels are either adjacent or overlapping. Consequently, connectivity of adjacent voxels can be easily established and the intersection curve can be easily identified. The new method can handle Boolean operations of any type. There are no special cases or degenerated cases to take care of. Therefore, the new method is robust. Most importantly, in the new method, error of each Boolean operation result can be precisely estimated. Therefore, error control is possible. Another important feature of the new method is, secondary local voxelization can be performed on intersecting subpatches to increase accuracy of Boolean operations.

The remaining part of the paper is arranged as follows. A brief review of background and previous works in this area are given in Section 2. A description of our voxelization technique is given in Section 3. The process of performing Boolean operations on solids represented by CCSSs is discussed in Section 4. Local voxelization technique is presented in Section 5. Error control is discussed in Section 6. Implementation issues and test results are shown in Section 7. Concluding remarks are given in Section 8.

2. BACKGROUND AND RELATED WORK

2.1 Subdivision Surfaces

Given a control mesh, a subdivision surface is generated by iteratively refining (subdividing) the control mesh to form new and finer control meshes. The refined control meshes converge to a limit surface called a *subdivision surface*. So a subdivision surface is determined by the given control mesh and the mesh refining (subdivision) process. Popular subdivision surfaces include Catmull-Clark subdivision surfaces (CCSSs) [1], Doo-Sabin subdivision surfaces [2] and Loop subdivision surfaces [3].

Subdivision surfaces can model/represent complex shape of arbitrary topology because there is no limit on the shape and topology of the control mesh of a subdivision surface. Subdivision surfaces are intrinsically discrete. Recently it was proved that subdivision surfaces can also be parametrized [4-7]. Therefore, subdivision surfaces cover both *parametric forms* and *discrete forms*. Parametric forms are good for design and representation, discrete forms are good for machining and tessellation (including FE mesh generation). Hence, we have a representation scheme that is good for all graphics and CAD/CAM applications. Subdivision surfaces by far are the most general surface representation scheme. They include non-uniform B-spline and NURBS surfaces as special cases [9]. In this paper we only consider solids represented by CCSSs. But our approach works for any subdivision scheme whose limit surfaces are parametrizable.

2.2 Voxelization

Like pixelization of 2D items, voxelization of 3D surfaces [10-11] is a powerful technique for representing and modeling complex 3D objects. This is proved by many successful applications of volume graphics techniques reported recently. For example, voxelization can be used for visualization of complex objects or scenes [12, 14, 27]. It can also be used for measuring integral properties of solids, such as mass, volume and surface area. Most importantly, it can be used for intersection curve calculation and, consequently, Boolean operations of free-form objects such as objects represented by CSG trees [25].

A good voxelization method should meet three criteria in the voxelization process: *separability*, *accuracy*, and *minimality* [10-11]. The first criterion demands analogy between the continuous space and the discrete space to be preserved and the resulting voxelization not to be penetrable since the given solid is closed and continuous. The second criterion ensures that the resulting voxelization is the most accurate discrete representation of the given solid according to some appropriate error metric. The third criterion requires the voxelization does not contain any voxels that, if removed, make no difference in separability and accuracy. The mathematical definitions of these criteria can be found in [10-11].

Note that a voxelization process does not render the voxels but merely generates a database of the discrete digitization of the continuous object [10]. Some previous voxelization methods use quad-trees to store the voxelization results [26]. This approach can save memory space but might sacrifice in time when used for applications such as Boolean operations or intersection curves determination. Nevertheless, with cheap and giga-byte memory chips becoming available, storage requirement is no longer a major issue in the design of a voxelization algorithm, people

care more about the efficiency of the algorithm. Our new method stores the voxelization result in a *Cubic Frame Buffer* [10] directly for the purpose of operation performance.

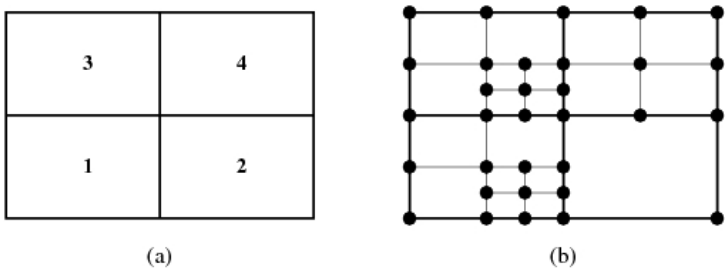

Fig. 1: Basic idea of parameter space based recursive voxelization.

2.3 Boolean Operations on Solids

Performing Boolean operations on solids is a classic problem in geometric modeling. Many approaches have been reported in the literature, such as [13, 19, 22, 24, 25, 26, 28], to name a few. Most solid modelers nowadays can support Boolean operations on solids composed of polyhedral models or bounded by quadric surfaces (like spheres, cylinders, etc.). Over the last few years, modeling with free-form surfaces has become a consensus throughout the commercial CAD/CAM industry. The bottleneck, however, is in performing robust, efficient and accurate Boolean operations on free-form objects. The topology of a surface patch becomes quite complicated once a number of Boolean operations have been performed on the surface patch, finding a compact and reliable representation for such a topology has been a major challenge. As a result, some solid modelers [13] use polyhedral approximations to these surfaces and apply Boolean operations on these approximate polyhedral objects. A problem with this seemingly simple approach is there are always difficult, special or degenerated cases to take care of [16]. Some modelers use *point* (or *surfel*) based approach [26] to perform Boolean operations on solids and quite good results have been obtained. *Error control* is difficult for such an approach. Zorin et al. proposed a method [28] to perform approximate Boolean operations on free-form solids represented by subdivision surfaces. The major contribution of their method is the algorithms that can generate a control mesh for a multi-resolution surface approximating the Boolean operation result.

A major task in performing Boolean operations on free-form objects is to compute surface intersection [15, 17, 18, 20, 21, 23]. Algebraic degree of the resulting curve usually is very high (e.g., up to 324 for a pair of bicubic Bézier surfaces) [13] and the genus is usually non-zero. Therefore it is difficult to represent the intersection curve analytically, if not at all possible. Current methods focus on computing an approximation to the intersection curve.

3. VOXELIZATION BASED ON RECURSIVE PARAMETER SPACE SUBDIVISION

Given a free-form object represented by a CCSS and a *cubic frame buffer* of resolution $M_1 \times M_2 \times M_3$, the goal is to convert the CCSS represented free-form object (a continuous geometric representation) into a set of voxels that best approximates the geometry of the object. We assume each face of the representation's control mesh is a quadrilateral and each face has at most one *extra-ordinary vertex* (a vertex with a *valence* different from 4). In the following we will show voxelization of the object's boundary representation only. Voxelization of the interior is just a flooding operation once the boundary is voxelized.

We first consider voxelization of a single patch of the CCSS representation. Given a patch $\boldsymbol{S}(u, v)$ defined on $[u_1, u_2] \times [v_1, v_2]$, we voxelize it by recursively subdividing its parameter space until each subpatch is small enough (hence, flat enough) so that voxelization of that subpatch can be done simply by voxelizing its four corners. It is easy to see that if the voxels corresponding to the four corners of a subpatch are not N-adjacent ($N \in \{6, 18, 26\}$) to each other [10-12], then there exist holes between them. In this case, the subpatch is considered not small enough yet. A *midpoint subdivision* is performed on the parameter space to get four smaller subpatches and repeat the testing process on each of the subpatches. This process is recursively repeated until all the subpatches are small enough and can be voxelized using only their four corners.

The vertices of the resulting subpatches after the recursive parameter space subdivision are then used to form voxels in the voxelization process to approximate $\boldsymbol{S}(u, v)$. For example, if the four rectangles in Figure 1(a) are parameter spaces of $\boldsymbol{S}(u, v)$'s subpatches and if the rectangles shown in Figure 1(b) are parameter spaces of resulting subpatches when the above recursive testing process stops, then vertices of these subpatches (those correspond to 2D parameter space points marked with small solid circles) are used to form voxels to approximate $\boldsymbol{S}(u, v)$.

The above process guarantees that a shared boundary (vertex) of adjacent subpatches will be voxelized to the same voxels (voxel). This is true for adjacent patches as well. Hence, voxelization of the entire CCSS representation can be performed on a patch based approach. To make the process of writing voxels into the cubic frame buffer simpler, the control mesh of the CCSS representation is normalized to be of dimension $[0, M_1\text{-}1] \times [0, M_2\text{-}1] \times [0, M_3\text{-}1]$ first. Patches of the CCSS representation are then voxelized one at a time, and the resulting voxels are then written into corresponding entries of the cubic frame buffer. Result of this voxelization process satisfies the criteria of *separability*, *accuracy* and *minimality* with respect to the given N-adjacency connectivity requirement ($N \in \{6, 18, 26\}$) [10-12].

To avoid stack overflow, only small subpatches should be fed to the recursive subdivision and testing process. This is especially true when a high resolution cubic frame buffer is given or some polygons in the given control mesh are very big. Generating small subpatches is not a problem for a CCSS once parametrization techniques are available. In our implementation, the size of subpatches fed to the recursive testing process is $\frac{1}{8} \times \frac{1}{8}$, i.e. each patch is divided into 8 ×8 subpatches before the voxelization process is performed. Feeding small size subpatches to the recursive testing process also ensures assumption of our voxelization process is implicitly satisfied, because the smaller the parameter space of a subpatch, the flatter the subpatch.

4. BOOLEAN OPERATIONS ON SOLIDS

We only consider Boolean operations with two operands, A and B. Boolean operations with more operands can be treated as a series of Boolean operations each with two operands. Hence, only two *cubic frame buffers* are needed for each Boolean operation, one for each operand. The representation of each solid will be voxelized first and then a *volume flooding* is performed to mark all the voxels that are inside the given solid. Therefore, voxels in each cubic frame buffer can be classified into three categories: (1) *IN* voxels, (2) *ON* voxels and (3) *OUT* voxels. Furthermore, even though several Boolean operation types are available but, since the core operation is almost the same, we will use the *intersection* operation to illustrate the process only.

With voxelization, the process of performing a Boolean operation becomes quite straightforward. It is how to represent the result of the Boolean operation that is critical. Traditionally the result of a Boolean operation is simply represented with voxels. The disadvantage of this approach is the result cannot be scaled seamlessly because of the discrete nature of voxels. In the following, we show how to represent the final result of an intersection operation with a continuous geometric representation.

4.1 Intersection Operation based on Recursive Parameter Space Subdivision and Voxelization

Let $\boldsymbol{S}(u, v)$ be a patch of the CCSS representation of solid A. For each subpatch of $\boldsymbol{S}(u, v)$ resulted from the voxelization process, we voxelize it one more time using the method discussed in Section 3. However, this time we do not write voxels into A's cubic frame buffer, but look up the voxel values in both solid A and solid B's cubic frame buffers. If voxel values of this subpatch in both cubic frame buffers are either *IN* or *ON*, then this is a subpatch to keep. Subpatches of this type are called *K-subpatches* (recall that we are performing an intersection operation.) If voxel values of this subpatch are all *OUT* in both cubic frame buffers, then this is a subpatch to discard. Subpatches of this type are called *D-subpatches*. Otherwise, i.e., if some of the voxel values are *IN* or *ON* and some of the voxel values are *OUT*, then this is a patch with some portion to keep and some portion to discard. Subpatches of this type are called *I-subpatches* (intersecting subpatches). For example, the rectangles shown in Fig. 2 (a) are the parameter spaces of the resulting subpatches when the recursive voxelization process stops and the dashed Polyline is part of the intersection curve of the two given solids in this patch's 2D parameter space. We can see that subpatch $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_4\boldsymbol{A}_3$ in Fig. 2(a) is an I-subpatch. Note here that all the marked adjacent points, when evaluated and voxelized, will be mapped to either the same voxel or adjacent voxels (see Section 3). For example, there does not exist any voxels between the voxels

corresponding to $\boldsymbol{A}_1$ and $\boldsymbol{A}_3$. Therefore, even though the intersection curve does not pass through $\boldsymbol{A}_1$ or $\boldsymbol{A}_3$, the voxel corresponding to the intersection point $\boldsymbol{I}_1$ will fall into the closest voxel corresponding to $\boldsymbol{A}_1$ or $\boldsymbol{A}_3$. In this case, it falls into the voxel corresponding to $\boldsymbol{A}_1$.

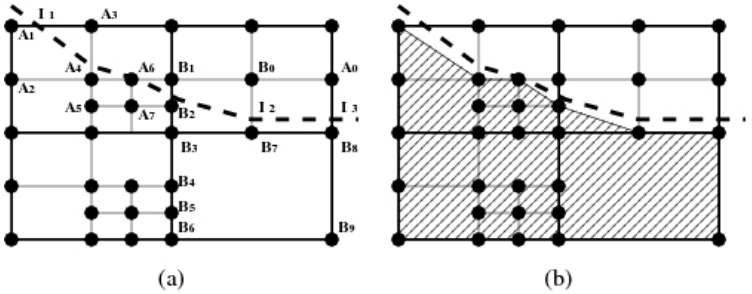

Fig. 2: Performing intersection operation on 2D parameter space.

An *I-voxel* is a voxel whose voxel value is *ON* in both cubic frame buffers. Hence it is easy to identify all I-voxels, which compose the intersection curve (but at this moment we do not know how to connect these I-voxels yet). For example, in Fig. 2(a), parameter space points $\boldsymbol{A}_1$ and $\boldsymbol{B}_7$ are I-voxels. Once all the I-voxels have been identified, a continuous geometric representation for the resulting solid can be generated. K-subpatches and D-subpatches are easy to handle. For example, in Fig. 2(b), $\boldsymbol{A}_4\boldsymbol{A}_5\boldsymbol{A}_7\boldsymbol{A}_6$ is a K-subpatch, hence $\boldsymbol{A}_4\boldsymbol{A}_5\boldsymbol{A}_7\boldsymbol{A}_6$ will be output to the tessellation or rendering process. For an I-subpatch, one can determine which part of the subpatch to keep simply by traversing all the marked points attached to this subpatch. For example, for the subpatch $\boldsymbol{B}_0\boldsymbol{B}_1\boldsymbol{B}_2\boldsymbol{B}_3\boldsymbol{B}_7$ in Fig. 2(a), after a traverse of the marked vertices, it is easy to see that the part to keep is $\boldsymbol{B}_2\boldsymbol{B}_3\boldsymbol{B}_7$. Hence $\boldsymbol{B}_2\boldsymbol{B}_3\boldsymbol{B}_7$ will be used in the tessellation and rendering process. Note here that the intersection point $\boldsymbol{I}_2$, after voxelization, maps to the same voxel as $\boldsymbol{B}_7$. In Fig. 2(b) the shaded part is the result after performing the Boolean operation in the 2D parameter space. Once we have the result of the Boolean operation in 2D parameter space, the 3D result can be easily obtained by directly evaluating and tessellating these shaded polygons. A connected intersection curve can be easily constructed as well. For example, in Figure 2, the intersection curve (inside this patch) is $\boldsymbol{A}_1\boldsymbol{A}_4\boldsymbol{A}_6\boldsymbol{B}_2\boldsymbol{B}_7\boldsymbol{B}_8$.

The above voxelization process and Boolean operations guarantee that shared boundary or vertex of patches or subpatches will be chopped, kept or discarded in exactly the same way no matter on which patch the operation is performed. Therefore, in our method, Boolean operations on free-form objects represented by CCSSs can be performed on the basis of individual patches. Consequently, no special or degenerated cases need to take care of.

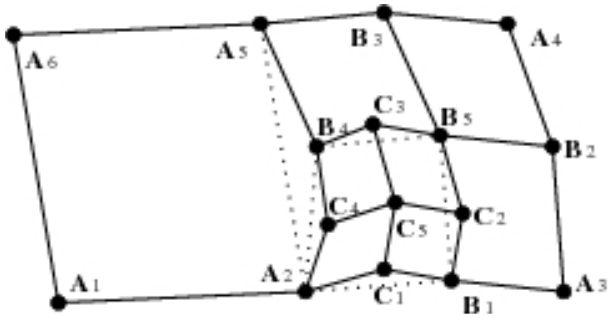

Fig. 3: Crack elimination.

4.2 Crack Elimination

Due to the fact that adjacent patches might be approximated by quadrilaterals corresponding to subpatches from different levels of the midpoint subdivision process, *cracks* could occur between adjacent patches or subpatches. For instance, in Figure 3, the left patch $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_5\boldsymbol{A}_6$ is approximated by one quadrilateral but the right patch is approximated by 7 quadrilaterals. Consider the boundary shared by the left patch and the right patch. On the left side, that boundary is a line segment defined by two vertices: $\boldsymbol{A}_2$ and $\boldsymbol{A}_5$. But on the right side, the boundary is a polyline defined by four

vertices: $\boldsymbol{A}_2$, $\boldsymbol{C}_4$, $\boldsymbol{B}_4$, and $\boldsymbol{A}_5$. They would not coincide unless $\boldsymbol{C}_4$ and $\boldsymbol{B}_4$ lie on the line segment defined by $\boldsymbol{A}_2$ and $\boldsymbol{A}_5$. But that usually is not the case. Hence, cracks would appear between the left patch and the right patch.

Fortunately cracks can be eliminated simply by replacing each boundary of a patch or subpatch with the one that contains all the evaluated points for that boundary. For example, in Figure 3, all the dotted lines should be replaced with the corresponding polylines. In particular, boundary $\boldsymbol{A}_2\boldsymbol{A}_5$ of patch $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_5\boldsymbol{A}_6$ should be replaced with the polyline $\boldsymbol{A}_2\boldsymbol{C}_4\boldsymbol{B}_4\boldsymbol{A}_5$. As a result, polygon $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_5\boldsymbol{A}_6$ is replaced with polygon $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{C}_4\boldsymbol{B}_4\boldsymbol{A}_5\boldsymbol{A}_6$ in the tessellation process. For rendering purpose this is fine because graphics systems like *OpenGL* can handle polygons with non-co-planar vertices and polygons with any number of sides. However, it should be pointed out that through a simple zigzag technique, triangulation of those polygons is actually a simple and very fast process.

Cracks could also occur if solids *A* and *B* are not connected properly in the intersecting area. For example in Fig. 2(a), intersection point $\boldsymbol{I}_1$ after evaluation and voxelization falls to the voxel corresponding to the 2D parameter point $\boldsymbol{A}_1$ of solid *A*. On the other hand, on *B*, if $\boldsymbol{I}_1$ falls to the voxel corresponding to the 2D parameter point $\overline{A}_1$ of *B*, then because $\boldsymbol{S}_A(\boldsymbol{A}_1)$ might not equal $\boldsymbol{S}_B(\overline{A}_1)$ exactly, crack occurs. To eliminate this kind of cracks, we cannot use the exact 3D positions evaluated from 2D parameter points for intersection points. Instead we use the midpoint of the corresponding 3D points, $\boldsymbol{S}_A(\boldsymbol{A}_1) + \boldsymbol{S}_B(\overline{A}_1))/2$, as the intersection point. In this way, solids *A* and *B* will not only have exactly the same intersection positions, but the same intersection curve as well. As a result, solids *A* and *B* can be connected seamlessly. Note that for K-subpatches, their vertices will be evaluated directly from parameter points. Only intersection points of partially kept I-subpatches are approximated by the centers of their corresponding voxels.

5. LOCAL VOXELIZATION

The voxelization process presented in Section 3 is called a *global voxelization*, as it is performed on the entire object space. Actually after the above mentioned Boolean operations are performed, a fine scale voxelization, called a *local voxelization*, can also be performed. The goal of local voxelization is to improve the accuracy of I-subpatches. For example, in Fig. 2(a), $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_4$ is used to approximate the area of the I-subpatch $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_4\boldsymbol{A}_3$ that should be kept. The accuracy of this approximation depends on the resolution of the global cubic frame buffer, which is always not high enough because of limited memory resource. However, we can do a secondary voxelization on individual portions of the object space. Resolution of a secondary voxelization is not as high as that of a global voxelization, but enough to cover sufficient details of the region applied. This is because the secondary voxelization is usually applied to a very small region of the object space only. As a result, high accuracy can still be achieved in critical areas.

The process and the approach used for a local voxelization are the same as a global voxelization. The only difference is that they are applied to individual regions of the object space. In order to perform local voxelization, information about which subpatches of solid $\boldsymbol{A}$ intersecting which subpatches of solid $\boldsymbol{B}$ must be known first. This information is very difficult to obtain in previous voxelization based methods. In our method, nevertheless, it can be readily obtained when performing the Boolean operations, as mentioned in Section 4.1. If we mark these *I-subpatches* of solids $\boldsymbol{A}$ and *B* during the keep-or-discard test process, we would know exactly which subpatches of solid $\boldsymbol{A}$ intersect which subpatches of solid $\boldsymbol{B}$. Once all *I-subpatches* are known, local voxelization can be performed directly on each pair of I-subpatches. For example, if subpatch p_1 of object *A* intersects subpatches q_1 and q_2 of object $\boldsymbol{B}$, then a local voxelization is performed on these 3 subpatches. Their intersection curve is used to replace the intersection curve obtained using the global voxelization process. The local voxelization process is applied to *every* pair of I-subpatches of solids $\boldsymbol{A}$ and $\boldsymbol{B}$. Consequently, more accurate intersection curve can be obtained. For instance, in Fig. 2(a), the intersection curve $\boldsymbol{A}_4\boldsymbol{A}_1$ will be replaced with $\boldsymbol{V}_1\boldsymbol{V}_2 \ldots \boldsymbol{V}_k$, if $\boldsymbol{V}_i$, $i=1\ldots k$ are the new intersecting voxels in the corresponding local cubic frame buffers and polygon $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_4\boldsymbol{V}_k\boldsymbol{V}_{k-1}\ldots\boldsymbol{V}_1$ will be used in the tessellation and rendering process instead. Similar to global voxelization, only two local cubic frame buffers are needed for local voxelization. The local cubic frame buffers can be reused for each new pair of I-subpatches. Hence local voxelization does not require much memory at all.

6. ERROR CONTROL

Given an error tolerance ε, the purpose of error control is to make sure error carried by the resulting solid of a Boolean operation is less than ε. Because the resulting solid is approximated by a polygonal mesh, to measure the difference between a patch (or subpatch) and its corresponding quadrilateral, we need to parametrize the quadrilateral and the

patch (or subpatch) first. It is well known now that any patch or subpatch $\boldsymbol{S}$ (u, v), $(u, v) \in [u_1, u_2] \times [v_1, v_2]$ of a CCSS can be explicitly parameterized [4-7]. A quadrilateral defined by four corners

$$V_1 = S(u_1, v_1), \quad V_2 = S(u_2, v_1),$$
$$V_3 = S(u_2, v_2), \quad V_4 = S(u_1, v_2).$$

can be bilinearly parameterized as follows:

$$Q(u,v) = \frac{v_2 - v}{v_2 - v_1}\left(\frac{u_2 - u}{u_2 - u_1}V_1 + \frac{u - u_1}{u_2 - u_1}V_2\right) + \frac{v - v_1}{v_2 - v_1}\left(\frac{u_2 - u}{u_2 - u_1}V_4 + \frac{u - u_1}{u_2 - u_1}V_3\right)$$

The *difference* between the patch (or subpatch) and its corresponding quadrilateral at (u, v) is defined as

$$d(u,v) = \left\| Q(u,v) - S(u,v) \right\|^2 \tag{1}$$

If the following equation is satisfied,

$$\sqrt{d(\overline{u},\overline{v})} + \sqrt{d(\hat{u},\hat{v})} \le \varepsilon \tag{2}$$

where $(\hat{u},\hat{v})$ and $(\overline{u},\overline{v})$ are 2D parameter space points such that

$$d(\overline{u},\overline{v}) = \max\{d(u,v) \mid (u,v) \in [u_1,u_2]\times[v_1,v_2], (Q(\overline{u},\overline{v}) - S(\overline{u},\overline{v})) \cdot ((V_1 - V_3)\times(V_2 - V_4)) \le 0\}$$

and

$$d(\hat{u},\hat{v}) = \max\{d(u,v) \mid (u,v) \in [u_1,u_2]\times[v_1,v_2], (Q(\hat{u},\hat{v}) - S(\hat{u},\hat{v})) \cdot ((V_1 - V_3)\times(V_2 - V_4)) > 0\},$$

then the error between the patch (or subpatch) and the corresponding quadrilateral is said to be less than ε. From the definitions of $(\hat{u},\hat{v})$ and $(\overline{u},\overline{v})$, we can see that satisfying Eq. (2) means that the patch (or subpatch) being tested is located between two quadrilaterals that are ε away from each other.

If Eq. (2) is satisfied for every patch and the corresponding quadrilateral, then the error of the approximation for the entire CCSS surface is said to be smaller than ε. It is known that $(\hat{u},\hat{v})$ and $(\overline{u},\overline{v})$ can be explicitly calculated no matter $\boldsymbol{S}$ is a regular or extraordinary patch [8]. Hence after the global voxelization process, we can estimate the error between each resulting subpatch and the corresponding quadrilateral. For example, if $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_4\boldsymbol{A}_3$ in Figure 2(a) is the parameter space of a subpatch when the recursive global voxelization process stops, then error is computed for that subpatch. If the error does not satisfy Eq. (2), midpoint subdivision is recursively applied to $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_4\boldsymbol{A}_3$ until errors of all the resulting subpatches satisfy Eq. (2).

A potential problem with a subpatch that satisfies Eq. (2) is that the new polygon generated by the crack elimination process discussed above might not satisfy the given accuracy requirement any more. Fortunately, even a subpatch with the polyline replacement in the crack elimination process, we guarantee that the newly generated polygon still satisfies Eq. (2). Note that all the evaluated points lie on the limit surface. Hence, for instance, in Fig. 3, points $\boldsymbol{A}_2$, $\boldsymbol{C}_4$, $\boldsymbol{B}_4$ and $\boldsymbol{A}_5$ of patch $\boldsymbol{A}_2\boldsymbol{A}_3\boldsymbol{A}_4\boldsymbol{A}_5$ are also points of patch $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_5\boldsymbol{A}_6$. With the test condition in Eq. (2), we know that a patch or subpatch is flat enough if it is located between two quadrilaterals that are ε away. Because boundary points $\boldsymbol{A}_2$, $\boldsymbol{C}_4$, $\boldsymbol{B}_4$ and $\boldsymbol{A}_5$ are on the limit surface, they must be located between two quadrilaterals that are ε away. So is the polygon $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{C}_4\boldsymbol{B}_4\boldsymbol{A}_5\boldsymbol{A}_6$. Now the patch (or subpatch) and its approximating polygon are both located inside two quadrilaterals that are ε away. Hence the overall error between the patch (or subpatch) and its approximating polygon is guaranteed to be smaller than ε. Hence we can accurately estimate the error caused by the surface approximation of polygonalization.

Another source that could introduce error in the result of the Boolean operations is the voxelization process. Both the global and the local voxelization can cause inaccuracy. The kind of error caused by voxelization is easy to estimate if the resolutions of cubic frame buffers are known. For example, if the cubic frame buffer resolution is $R_1 \times R_2 \times R_3$ and the object space is of size $X_1 \times X_2 \times X_3$, then we can see that each voxel is of size $\frac{X_1}{R_1} \times \frac{X_2}{R_2} \times \frac{X_3}{R_3}$. It is easy to see the maximal error of voxelization is half the size of a voxel. If we perform local voxelization for every pair of intersecting

subpatches, then global voxelization will not cause any error. Here we can also see why local voxelization can improve the accuracy dramatically. In local voxelization, because the size of the subpatches being voxelized is very small, even with a low resolution, the voxel size is still very small.

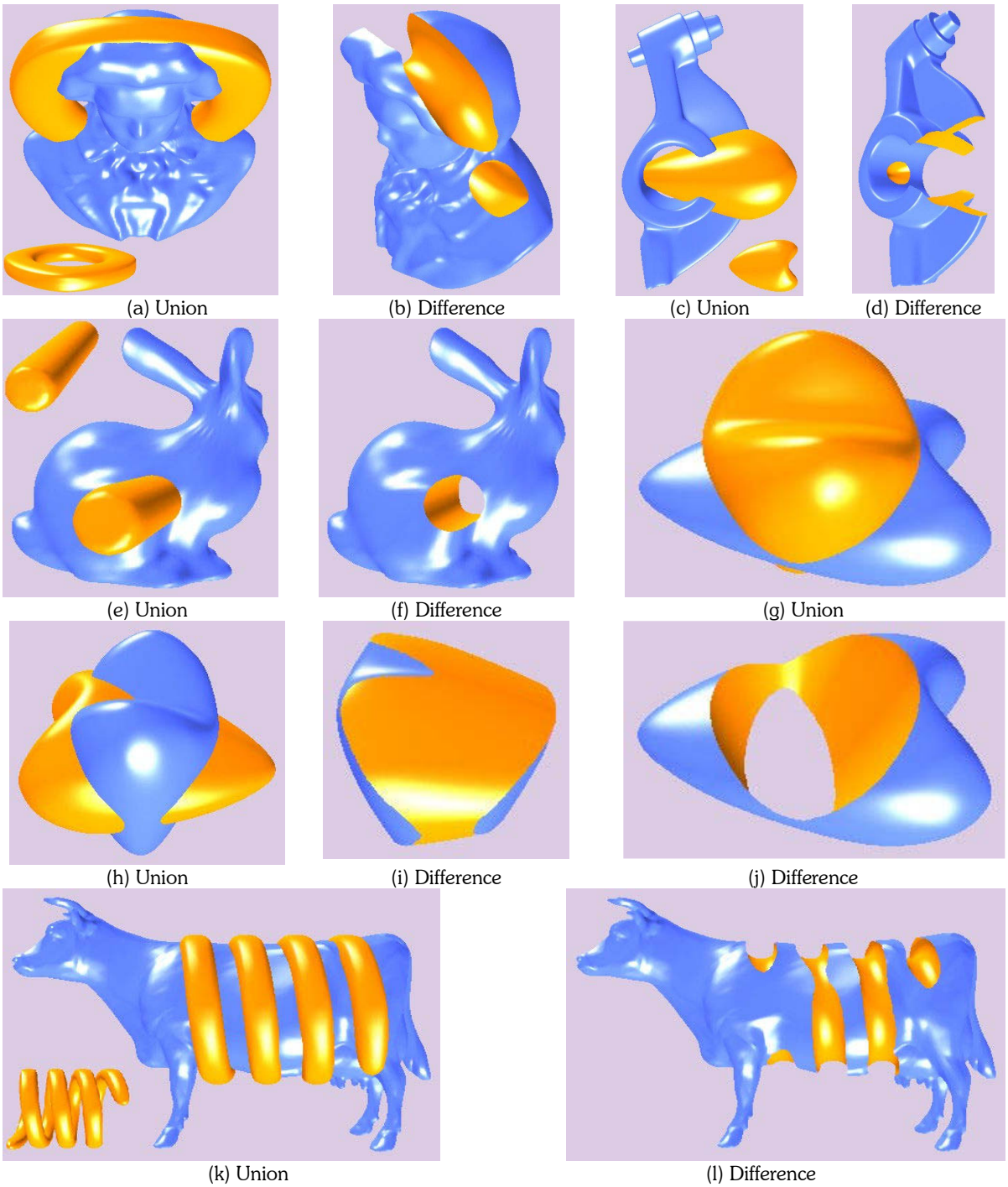

Fig. 4: Boolean Operations Performed on Solids Represented by CCSSes.

Therefore the overall error caused by polygonalization and voxelization is the sum of the errors caused by each of them. To make error of the final Boolean operation results less than the given ε everywhere, the test condition in Eq. (2) has to be changed to the following form:

$$\begin{cases} \sqrt{d(\overline{u},\overline{v})} + \sqrt{d(\hat{u},\hat{v})} & \leq \varepsilon/2 \\ size \quad of \quad each \quad voxel & \leq \varepsilon \end{cases} \tag{3}$$

where $(\hat{u},\hat{v})$ and $(\overline{u},\overline{v})$ are defined the same way as in Eq. (2). The first equation in Eq. (3) ensures the patch (or subpatch) and its approximating polygon are both located inside two quadrilaterals that are ε/2 away. The second equation in Eq. (3) ensures the error caused by voxelization is not bigger than ε/2. Hence the total error in the whole process is guaranteed to be less than ε.

7. TEST RESULTS

The proposed approach has been implemented in *C++* using *OpenGL* as the supporting graphics system on the Windows platform. Quite a few examples have been tested with the method described here. All the examples have extra-ordinary vertices. Some of the tested results are shown in Figure 4. Resolution of global voxelization is 512×512×512 for all the test examples, and error tolerance is set to 10^{-3} for all of them. The size of each example is normalized to [0, 1] before voxelization and Boolean operations are performed. Resolution of local voxelization, however, depends on value of the error tolerance and the given mesh. Hence, resolution of local voxelization is different for each of the examples shown in Figure 4. For example, resolution of local voxelization for Figures 4(k) and 4(l) is 8×8×8, while for Figures 4(g), 4(h), 4(i) and 4(j), resolution of local voxelization is 32×32×32. Although resolutions of local voxelization are different for these examples, the overall error is the same in the final results. From Eq. (3) we can see that the reason for the above difference of local voxelization resolutions is because error is caused by voxelization as well as polygonalization.

In Figure 4, all the *Difference* and *Intersection* operations are performed on solids positioned exactly the same as in the *Union* operation so that one can easily tell if results of the Boolean operations are accurate within the given error tolerance. For example, Figures 4(j) and 4(g) are results of *Difference* operation and *Union* operation, respectively, on solids placed in the same positions. Relationship between the remaining cases are: Figures 4(i) corresponds to 4(h), 4(b) corresponds to 4(a), 4(d) corresponds to 4(c), 4(f) corresponds to 4(e) and 4(l) corresponds to 4(k).

8. SUMMARY

A new method for performing robust and error controllable Boolean operations on free-form solids represented with CCSSs is presented. Test results show that this approach leads to good results even for complicated solids and solids of extra-ordinary topology. The new method has several special properties: First, Boolean operations can be performed on 2D parameter spaces on the basis of individual patches. There is no need to take care of special cases or degenerated cases. Hence the method is robust. Second, although voxelization is performed to facilitate Boolean operations, the result of a Boolean operation is still represented with a continuous geometric representation. Hence results of Boolean operations can be scaled seamlessly and smoothly. Third, error of Boolean operation results can be precisely estimated. According to the error estimating formula, a secondary local voxelization can be performed for intersecting subpatches. Hence higher accuracy can be achieved. Finally, although the new method is presented for CCSSs, the concept actually works for any subdivision scheme whose limit surfaces can be parametrized.

Acknowledgements. Research work reported in this paper is supported by NSF under grants DMS-0310645 and DMI-0422

9. REFERENCES

[1] Adams, S.-H.; Yang, M.-Y.: A study on a generalized parametric interpolator with real-time jerk-limited acceleration, Computer-Aided Design, 36(1), 2004, 27-36.

[2] Bates, J.; Ding, M.-S.; Park, J.: A Study on Rapid Prototyping Techniques, TR UMCP-97-003, University of Massachusetts, Boston, MA, 1997. http://www.um.edu/~bates.html

[3] Campbell, J. C.: Optimal work piece setup for 5-axis NC machining, Computer-Aided Design & Applications, 1(1-4), 2004, 234-145.
[4] Debra, N. L.: Principles of Mechanical Design, Oxford University Press, New York, NY, 1990.
[5] Eaton, J. A.: Layered Manufacturing Methods for Reconstructing Bone Structures, Ph.D. Thesis, University of Minnesota, Twin Cities, MN, 1998.
[6] Jewelspace, http://www.jewelspace.net, Caligari Software.
[1] Catmull, E.; Clark, J.: Recursively generated B-spline surfaces on arbitrary topological meshes, Computer-Aided Design, 10(6), 1978, 350-355.
[2] Doo, D.; Sabin, M.: Behavior of recursive division surfaces near extraordinary points, Computer-Aided Design, 10(6), 1978, 356-360.
[3] Loop, C. T.: Smooth Subdivision Surfaces Based on Triangles, MS thesis, Department of Mathematics, University of Utah, August, 1987.
[4] Stam, J.: Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values, Proceedings of SIGGRAPH 1998, 395-404.
[5] Stam, J.: Evaluation of Loop Subdivision Surfaces, SIGGRAPH'99 Course Notes, 1999.
[6] Zorin, D.; Kristjansson, D.: Evaluation of Piecewise Smooth Subdivision Surfaces, The Visual Computer, 18(5/6), 2002, 299-315.
[7] Lai, S.; Cheng F.: Parametrization of General Catmull Clark Subdivision Surfaces and its Application, Computer Aided Design & Applications, 3(1-4), 2006, 513-522.
[8] Lai S.; Cheng, F.: Adaptive Rendering of Catmull-Clark Subdivision Surfaces, 9th Int. Conf. Computer Aided Design & Computer Graphics, 2(1-4), 2005, 125-130.
[9] Sederberg, T. W.; Zheng, J.; Sewell D.: Non-uniform recursive subdivision surfaces, SIGGRAPH, 1998, 19-24.
[10] Cohen-Or, D.; Kaufman, A.: Fundamentals of Surface Voxelization, Graphical Models and Image Processing, 57, 6 (November 1995), 453-461.
[11] Huang, J.; Yagel, R.; Fillipov, V.; Kurzion, Y.: An Accurate Method to Voxelize Polygonal Meshes, IEEE Volume Visualization'98, October, 1998.
[12] Lai, S.; Cheng, F.: Voxelization of Free-Form Solids using Catmull-Clark Subdivision Surfaces, Lecture Notes in Computer Science (GMP2006), Vol. 4077, Springer, 2006, 595-601.
[13] Krishnan, S.; Manocha, D.: Computing Boolean Combinations of Solids Composed of Free-form Surfaces, Proceedings of the 1996 ASME Design for Manufacturing Conference, August 1996.
[14] Zwicker, M.; Pfister, H.; van Baar, J.; Gross, M.: Surface Splatting, SIGGRAPH 2001.
[15] Barnhill, R. E.; Farin, G.; Jordan, M; Piper, B. R.: Surface/surface intersection.,Computer Aided Geometric Design, 4(1-2), 1987, 3-16.
[16] Burnikel, C.; Mehlhorn, K.; Schirra, S.: On degeneracy in geometric computations, In Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, 1994, 16-23.
[17] Dobrindt, K.; Mehlhorn, K.; Yvinec, M.: A complete and eiÀcient algorithm for the intersection of a general and a convex polyhedron, In Algorithms and data structures, 1993, 314-324.
[18] Krishnan, S.; Manocha, D.: An eiÀcient surface intersection algorithm based on lower dimensional formulation, ACM Transactions on Graphics, 16(1), 1997, 74-106.
[19] Rappoport, A.; Spitz S.: Interactive Boolean operations for conceptual design of 3D solids, Proceedings of SIGGRAPH 97, 1997, 269-278.
[20] Sederberg, T.; Nishita, T.: Geometric hermite approximation of surface patch intersection curves, Computer Aided Geometric Design, 8(2), 1991, 97-114.
[21] Patrikalakis, M.: Surface-to-surface Intersections, IEEE Computer Graphics & Applications, 13(1), 1993, 89-95.
[22] Bieri, H.; Nef, W.: Elementary set operations with d-dimensional polyhedra, Computational Geometry and its Applications, LNCS 333, Springer-Verlag, 1988, 97-112.
[23] Chazelle, B.: An optimal algorithm for intersecting three dimensional convex polyhedra, SIAM J. Comput., 21(4), 1992, 671-696.
[24] Wiegand, T. F.: Interactive rendering of CSG models. Computer Graphics Forum, 15(4), 1996, 249-261.
[25] Liao, D.; Fang, S.: Fast CSG Voxelization by Frame Buffer Pixel Mapping, Proceedings of the 2000 IEEE Symposium on Volume Visualization, 43-48, 2000.
[26] Adams, B.; Dutre, P.: Interactive Boolean operations on surfel-bounded solids, SIGGRAPH 2003, pp651-656.
[27] Grossman, J. P.; Dally, W. J.: Point sample rendering, Eurographics Rendering Workshop 1998, pp181-192.
[28] Kristjansson, D.; Biermann, H.; Zorin, D.: Approximate Boolean operations on free-form solids, ACM SIGGRAPH 2001, 185-194.