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

Shuhua Lai and Fuhua (Frank) Cheng
Graphics & Geometric Modeling Lab, Department of Computer Science
University of Kentucky, Lexington, Kentucky 40506-0046

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 given objects are voxelized to make Boolean operations more efficient. However, different from previous voxelization based approaches, the final result of the Boolean operations in our method is represented with a continuous geometric representation. This is achieved by doing the Boolean operations in the parameter spaces of the solids, instead of the object space. The 2D parameter space is recursively subdivided until a keep-or-discard decision can be made for each resulting subpatch using results of the voxelization process. This approach allows us to easily compute a parametric approximation of the intersection curve and, consequently, build a continuous geometric representation for the Boolean operation result. To make the Boolean operation result more accurate, a secondary local voxelization can be performed for intersecting subpatches. Because the voxelization process itself is very fast and robust, the overall process is fast and robust too. Most importantly, error of Boolean operation result can be precisely estimated, hence error control is possible. In addition, our method can handle any cases of Boolean operations as long as the given solids are represented by CCSSs. Therefore there are no special or degenerated cases to take care of. Although the new method is presented for CCSSs, the concept actually works for any subdivision scheme whose limit surfaces can be parametrized.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - curve, surface, solid and object representations;

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

1 Introduction

Boolean operations are a nature way of constructing complex solid objects from simpler primitives. For example, the *Constructive Solid Geometry* (CSG) representation scheme allows users 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). But for complicated objects, because higher order B-reps are needed, it is usually very difficult to find the intersecting curve analytically. In addition, cares always to be taken to handle special cases and degenerated cases [16]. Hence, accurate Boolean operations are usually not 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 3D object modeling and rendering for a while. With voxelization, it is actually very simple to get all the resulting voxels after Boolean operations because now Boolean operations become simple set operations. The difficult part is how to represent the resulting object properly and accurately when voxelization is used in the Boolean operation process. Traditionally results of Boolean operations are represented as sets of voxels [24, 25] and special volumetric rendering algorithms are developed for visualizing Boolean operation results [14, 27]. The main disadvantage of this approach is that there is no continuous geometric representation for the resulting objects. Consequently, the results of Boolean operations cannot be scaled seamlessly or smoothly because of the nature of discretization.

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. The given objects are voxelized to Boolean operations more efficient. However, the final results of Boolean operations in our method are still represented with a continuous geometric representation. This is achieve 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 constructed and the intersection curve can be easily identified. Because Boolean operations are performed subpatch by subpatch in 2D parameter space, our method can handle Boolean operations of any type. There are no special cases or degenerated cases to take care of. Therefore our method is robust. Most importantly, error control is possible in our method. To make the Boolean results more accurate, according to our error estimation formula, a secondary local voxelization can be performed for each pair of intersecting subpatches.

The remaining part of the paper is arranged as follows. A brief review of background and previous works related to this one are given in Section 2. A description of our voxelization techniqu 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 given in Section 6. Implementation issues and test cases are shown in Section 7. Concluding remarks are given in Section 8.

2 Background & 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 also proved that subdivision surfaces can be prammetrized [4, 5, 6, 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 more general surface representation scheme. They include non-uniform B-spline and NURBS surfaces are special cases [9]. In this paper we only consider solids represented by CCSSs. However, our approach can be used for any subdivision scheme whose parametrization is available.

2.2 Voxelization

Like 2D pixelization, voxelization of 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 in research work 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 [12, 25]. For example, in [25], a series of Boolean operations are performed on objects represented by a CSG tree.

A good voxelization should meet three requirements in the voxelization process: *separability*, *accuracy*, and *minimality* [10, 11]. The first requirement demands analogy between the continuous space and the discrete space to be preserved and the resulting voxelization to be not penetratable since the given solid is closed and continuous. The second requirement ensures that the resulting voxelization is the most accurate discrete representation of the given solid according to some appropriate error metric. The third requirement requires the voxelization does not contain voxels that, if removed, make no difference in terms of separability and accuracy. The mathematical definitions of these requirements 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 result [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 directly in a *Cubic Frame Buffer* [10] for fast operation purpose.

2.3 Boolean Operations on Solids

Performing Boolean operations 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. Currently most solid modelers can support Boolean operations on solids composed of polyhedral models or quadric surfaces (like spheres, cylinders etc.). Over the last few years, modeling using free-form surfaces has become indispensable throughout the commercial CAD/CAM industry. However, the major bottleneck is in performing robust, efficient and accurate Boolean operations on free-form objects. The topology of a surface patch become quite complicated when a number of Boolean operations are performed and finding a convenient representation for these topologies has been a major challenge. As a result, some solid modelers [13] use polyhedral approximation to these surfaces and apply Boolean operations on these approximate polyhedral objects. Although this approaches seem simple, there are always some special cases or degenerated cases [16] that are difficult to take care of. Some modelers use point (or surfel) based approaches [26] to perform Boolean operations and quite good results are obtained. However, error control is difficult in such approaches. Zorin etc. proposed a method [28] to perform approximate Boolean operations on free-form solids represented by subdivision surfaces. The main contribution of their method is the algorithms that are able to generate a control mesh for a multiresolution surface approximating the Boolean results.

Most of the recent work in the literature on Boolean operations of curved models are focused on computing the surface intersection [15, 17, 18, 20, 21, 23]. However, the algebraic degree of the resulting curve can typically be very high (up to 324 for a pair of bicubic Bézier surfaces) [13] and the genus is also nonzero. Hence it is very difficult to represent the intersection curve analytically and the current methods are aimed at computing approximations 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 (i.e. continuous geometric representation) into a set of voxels that best approximates the geometry of the object. We assume each face of the control mesh is a quadrilateral and each face has at most one *extraordinary vertex* (a vertex with a *valence* different from 4). If this is not the case, simply perform Catmull-Clark subdivision on the control mesh of the CCSS twice.

With parametrization techniques for subdivision surfaces becoming available, it is possible now to model and represent any continuous but topologically complex object with an analytical representation [4, 5, 6, 7]. Given any given parameter space point (u, v), a surface point $\mathbf{S}(u, v)$ corresponding to this parameter space point can be exactly computed. Therefore, voxelization does not have to be performed in 3D object space, as the previous recursive voxelization methods did, one can do voxelization in 2D space by performing recursive subdivision and testing on the 2D parameter space.

We first consider the voxelization process of a subpatch, which is a small portion of a patch. Given a subpatch of $\mathbf{S}(u, v)$ defined on $[u_1, u_2] \times [v_1, v_2]$, we voxelize it by assuming this given subpatch is small enough (hence, flat enough) so that the voxels generated from it are the same as the voxels generated using its four corners:

$$\begin{array}{ll} \mathbf{V}_1 = \mathbf{S}(u_1, v_1), & \mathbf{V}_2 = \mathbf{S}(u_2, v_1), \\ \mathbf{V}_3 = \mathbf{S}(u_2, v_2), & \mathbf{V}_4 = \mathbf{S}(u_1, v_2). \end{array} \qquad (1)$$

In general this assumption does not hold. Hence a test must be performed before the patch or subpatch is voxelized. It is easy to see that if the voxels generated using its four corners are not N-adjacent ($N \in \{6, 18, 26\}$) to each other [10, 11, 12], then there exist holes between them. In this case, the patch or subpatch is still not small enough. So we perform a *midpoint subdivision* on the corresponding parameter space by setting

$$u_{12} = \frac{u_1 + u_2}{2} \quad \text{and} \quad v_{12} = \frac{v_1 + v_2}{2}$$

to get four smaller subpatches:

$$\begin{array}{ll} \mathbf{S}([u_1, u_{12}] \times [v_1, v_{12}]), & \mathbf{S}([u_{12}, u_2] \times [v_1, v_{12}]), \\ \mathbf{S}([u_{12}, u_2] \times [v_{12}, v_2]), & \mathbf{S}([u_1, u_{12}] \times [v_{12}, v_2]), \end{array}$$

and repeat the testing process on each of the subpatches. The 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 the limit surface. For example, if the four rectangles in Figure 1(a) are the parameter spaces of four adjacent

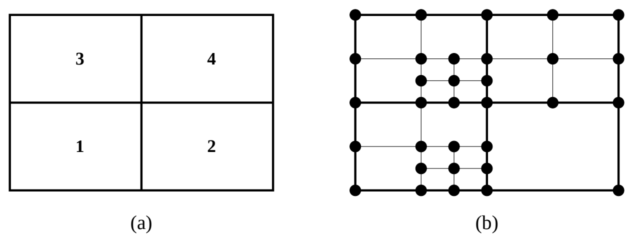


Figure 1: Basic idea of parameter space based recursive voxelization.

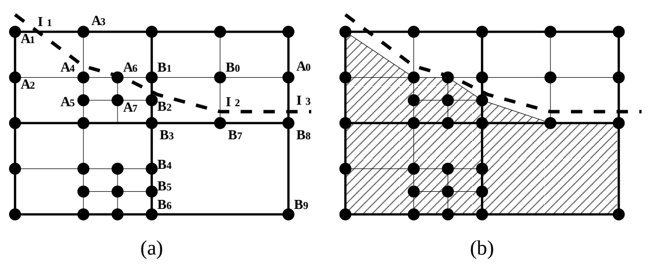


Figure 2: Performing Boolean operations on 2D parameter space.

subpatches of $\mathbf{S}(u, v)$, and if the rectangles shown in Figure 1(b) are the parameter spaces of the resulting subpatches when the above recursive testing process stops, then 3D points will be *evaluated* at the 2D parameter space points marked with small solid circles to form voxels that approximate the limit surface.

To make things simple, we first normalize the input mesh to be of dimension $[0, M_1 - 1] \times [0, M_2 - 1] \times [0, M_3 - 1]$. Then for any 2D parameter space point (u, v) generated from the recursive testing process (see Fig. 1), direct and exact evaluation is performed to get its 3D surface position and normal vector at $\mathbf{S}(u, v)$. To get the voxelized coordinates (i, j, k) from $\mathbf{S}(u, v)$, simply set

$$\begin{aligned} i &= \lfloor \mathbf{S}(u,v).x + 0.5 \rfloor, \\ j &= \lfloor \mathbf{S}(u,v).y + 0.5 \rfloor, \\ k &= \lfloor \mathbf{S}(u,v).z + 0.5 \rfloor. \end{aligned} \tag{2}$$

Once each single point marked in the recursive testing process is voxelized, the process of voxelizing the given patch is finished. The voxelization result of this method is guaranteed to satisfy the properties of *separability*, *accuracy* and *minimality* with respect to the given N-adjacency connectivity requirement ($N \in \{6, 18, 26\}$) [10, 11, 12].

Since the above process guarantees that a shared boundary or vertex of patches or subpatches will be voxelized to the same voxel, we can perform voxelization of free-form objects represented by a CCSS on a patch based approach. One thing that should be pointed out is, 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. For example, in our implementation, the size of subpatches (in the parameter space) 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. In addition, feeding small size subpatches to the recursive testing process ensures the assumption of our voxelization process is satisfied, because the smaller the parameter space of a subpatch, the flatter the subpatch.

4 Boolean Operations on Solids

Here we only consider Boolean operations performed on two free-form solids A and B. Boolean operations performed on more objects can be regarded as a series of Boolean operations each performed on two objects. Hence, only two cubic frame buffers are needed, one for each object. Once voxelization is done, a volume flooding must be performed to mark the voxels located inside a given solid. Now there are three types of voxels in each cubic frame buffer: (1) *inside voxels*, (2) *boundary voxels* and (3) *outside voxels*.

Several possible Boolean operations may be specified by the users. However, the essential process is is almost the same. Here we illustrate the process by assuming the given Boolean operation is to find the intersection of two solid objects.

With voxelization, it is actually quite simple to get the resulting voxels for a Boolean operation. For example, the voxels left after an intersection operation are those those located inside or on the boundary of both objects. The difficult part is how to represent the resulting part properly and accurately. Traditionally the results of Boolean operations are represented just with voxels. The main disadvantage of this method is the results cannot be scaled seemlessly because of the nature of discretization. In the following, we present an approach that represents the final result with a continuous geometric representation.

4.1 Boolean Operations based on Recursive Parameter Space Subdivision and Voxelization

For a subpatch of $\mathbf{S}(u, v)$ of solid A defined on $[u_1, u_2] \times [v_1, v_2]$, we voxelize it one more time using the method discussed in Section 3. However, this time we do not write the 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 all the voxel values of this subpatch in both cubic frame buffer are not *outside*, then this is subpatch to keep. Subpatches of this type are called *K-subpatches* (subpatches to be kept). (recall that we are performing an intersection operation.) If the voxel values of this subpatch are all *outside* in both A and B's cubic frame buffer, then this is a subpatch to discard. Subpatches of this type are called *D-subpatches* (subpatches to be discarded). Otherwise, i.e., if some of the voxel values are *inside*, *boundary* and some of the voxel values are *outside*, then this is a patch with some part to keep and some part to discard. Subpatches whose voxel values contain all of *inside*, *boundary* and *outside* 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 $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_4\mathbf{A}_3$ in Fig. 2 (a) is an I-subpatch. Note here 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 voxel between voxels corresponding to parameter points $\mathbf{A}_1$ and $\mathbf{A}_3$. Therefore, even though the intersection curve does not pass through $\mathbf{A}_1$ or $\mathbf{A}_3$, the voxel corresponding to the intersection point $\mathbf{I}_1$ will fall into the closest voxel corresponding to parameter point $\mathbf{A}_1$ or $\mathbf{A}_3$. In this case, it falls into the voxel corresponding to $\mathbf{A}_1$.

An *intersecting voxel* is a voxel whose voxel value is *boundary* in both cubic frame buffers. Hence it is very easy to find all the intersecting voxels, which compose the intersection curve (but at this moment we do not know how to connect these intersecting voxels yet). For example, in Fig. 2(a), parameter points $\mathbf{A}_1$ and $\mathbf{B}_7$ are intersecting voxels. Once all the intersecting voxels are 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), $\mathbf{A}_4\mathbf{A}_5\mathbf{A}_7\mathbf{A}_6$ is a K-subpatch, hence $\mathbf{A}_4\mathbf{A}_5\mathbf{A}_7\mathbf{A}_6$ will be output in the tessellation or rendering process. For an I-subpatch, one can determine which part of the subpatch to keep by traversing all the marked points attached to this subpatch For example, for the subpatch $\mathbf{B}_0\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3\mathbf{B}_7$ in Fig. 2(a), after a traverse of the marked vertices, it is easy to see that the part to keep is $\mathbf{B}_2\mathbf{B}_3\mathbf{B}_7$. Hence $\mathbf{B}_2\mathbf{B}_3\mathbf{B}_7$ will be used in the tessellation and rendering process. Note here the intersection point $\mathbf{I}_2$, after voxelization, maps to the same voxel as $\mathbf{B}_7$. 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 $\mathbf{A}_1\mathbf{A}_4\mathbf{A}_6\mathbf{B}_2\mathbf{B}_7\mathbf{B}_8$.

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 approach, Boolean operations of free-form objects represented by CCSSs can be performed on the basis of individual patches.

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 $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$ 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 : $\mathbf{A}_2$ and $\mathbf{A}_5$. But on the right side, the boundary is a polyline defined by four vertices : $\mathbf{A}_2$, $\mathbf{C}_4$, $\mathbf{B}_4$, and $\mathbf{A}_5$. They would not coincide unless $\mathbf{C}_4$ and $\mathbf{B}_4$ lie on the line segment defined by $\mathbf{A}_2$ and $\mathbf{A}_5$. But that usually is not the case. Hence, cracks would appear between the left patch and the right patch.

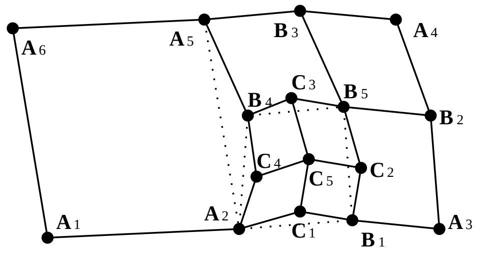


Figure 3: Crack elimination.

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 $\mathbf{A}_2\mathbf{A}_5$ of patch

$\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$ should be replaced with the polyline $\mathbf{A}_2\mathbf{C}_4\mathbf{B}_4\mathbf{A}_5$. As a result, polygon $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$ is replaced with polygon $\mathbf{A}_1\mathbf{A}_2\mathbf{C}_4\mathbf{B}_4\mathbf{A}_5\mathbf{A}_6$ 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 $\mathbf{I}_1$ after evaluation and voxelization falls to voxel corresponding to 2D parameter point $\mathbf{A}_1$ of solid A. If $\mathbf{I}_1$ falls to voxel corresponding to 2D parameter point $\bar{\mathbf{A}}_1$ of solid B, then after evaluation, $\mathbf{S}_A(\mathbf{A}_1)$ might not equal $\mathbf{S}_B(\bar{\mathbf{A}}_1)$ exactly. Hence crack occurs. To eliminate this kind of cracks, we cannot use the exact 3D positions evaluated from 2D parameter points for intersection point. Instead we use the center of the corresponding voxel as the intersection point. In this way, solids A and B will have exactly the same intersection positions and intersection curve as well. As a result, solids A and B can be connected seamlessly. Note that for K-supatches, 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*, because it is performed for the entire object space. After all the Boolean operations are performed, a fine scale voxelization, called a *local voxelization*, will also be performed. The goal of the local voxelization is to improve the accuracy of the I-subpatches. For example, in Fig. 2(a), $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_4$ is used to approximate the area of the I-subpatch $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_4\mathbf{A}_3$ 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, which has lower resolution, but is only applied to a very small portion of the object space. As a result high accuracy still can be achieved at intersecting area.

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 different size of the object space. In order to perform local voxelization, information about which subpatches of solid A intersecting with which subpatches of solid B must be known first. This information is very difficult to obtain in previous voxelization based methods. Fortunately, in our method, it can be readily obtained when performing the Boolean operations, as mentioned in Section 4.1. If we mark these intersecting subpatches of solids A and B during the keep-or-discard test process, we would know exactly which subpatches of solid A intersect which subpatches of solid B. Once all intersecting subpatches are known, local voxelization can be directly performed for each pair of intersecting subpatches. For example, suppose subpatch p_1 of object A intersects subpatches q_1 and q_2 of object B, then a local voxelization is performed on these 3 subpatches only. 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 intersecting subpatches of solids A and B. Consequently, more accurate intersection curve could be computed. For instance, in Fig. 2(a), the intersection curve $\mathbf{A}_4\mathbf{A}_1$ will be replaced with $\mathbf{V}_1\mathbf{V}_2\cdots\mathbf{V}_k$, $k = 10$, if $\mathbf{V}_i$, $i = 1\cdots10$ are the new intersecting voxels in the corresponding local cubic frame buffers and polygon $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_4\mathbf{V}_1\mathbf{V}_2\cdots\mathbf{V}_k$ will be used in the tessellation and rendering process. 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 intersecting subpatches. Hence local voxelization does not require a lot of memory.

6 Error Control

Given an ϵ, the purpose of error control is to make sure the error of 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 $\mathbf{S}(u,v)$, $(u,v) \in [u_1,u_2]\times[v_1,v_2]$ of a CCSS can be explicitly parameterized [4, 5, 6, 7]. A quadrilateral with four corners $\mathbf{V}_1$, $\mathbf{V}_2$, $\mathbf{V}_3$ and $\mathbf{V}_4$ (see eq. (1) for their definitions) can be parameterized as follows:

$$\begin{aligned}\mathbf{Q}(u,v) \;=\;& \tfrac{v_2-v}{v_2-v_1}\big(\tfrac{u_2-u}{u_2-u_1}\mathbf{V}_1 + \tfrac{u-u_1}{u_2-u_1}\mathbf{V}_2\big)\\ &+\tfrac{v-v_1}{v_2-v_1}\big(\tfrac{u_2-u}{u_2-u_1}\mathbf{V}_4 + \tfrac{u-u_1}{u_2-u_1}\mathbf{V}_3\big)\end{aligned}$$

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

$$d(u,v) \;=\; \|\, \mathbf{Q}(u,v) - \mathbf{S}(u,v)\,\|^2 \qquad (3)$$

If the following equation is satisfied, then the error between the patch (or subpatch) and the corresponding quadrilateral is said to be less than ϵ.

$$\sqrt{d\ (\ \bar{u},\ \bar{v})} + \sqrt{d\ (\ \hat{u},\ \hat{v})}\ \leq \epsilon \qquad (4)$$

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

$$\begin{cases} d(\bar{u},\bar{v}) = \max\{d(u,v) \mid (u,v) \in [u_1,u_2] \times [v_1,v_2]\} \\ (\mathbf{Q}(\bar{u},\bar{v} - \mathbf{S}(\bar{u},\bar{v})) \cdot ((\mathbf{V}_1 - \mathbf{V}_3) \times (\mathbf{V}_2 - \mathbf{V}_4)) \leq 0 \end{cases}$$

and

$$\begin{cases} d(\hat{u},\hat{v}) = \max\{d(u,v) \mid (u,v) \in [u_1,u_2] \times [v_1,v_2]\} \\ (\mathbf{Q}(\bar{u},\bar{v} - \mathbf{S}(\bar{u},\bar{v})) \cdot ((\mathbf{V}_1 - \mathbf{V}_3) \times (\mathbf{V}_2 - \mathbf{V}_4)) > 0 \end{cases}$$

From the definitions of $(\hat{u}, \hat{v})$ and $(\bar{u}, \bar{v})$, we can see that satisfying Eq. (4) means that the subpatch being tested is located between two quadrilaterals that are ϵ away from each other.

If eq. (4) is satisfied for every 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 $(\bar{u}, \bar{v})$ can be explicitly calculated no matter $\mathbf{S}$ 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 the rectangles shown in Figure 2(a) are the parameter spaces of the resulting subpatches when the recursive global voxelization process stops, then error will be calculated for each quadrilateral, say $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_4\mathbf{A}_3$. If eq. (4) is not satisfied, a midpoint subdivision will be performed on the 2D parameter subspace corresponding to this subpatch until all the subpatches satisfy eq. (4).

A potential problem for a subpatch that satisfies eq. (4) is 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 is still satisfies eq. (4). Note that all the evaluated points lie on the limit surface. Hence, for instance, in Fig. 3, points $\mathbf{A}_2, \mathbf{C}_4, \mathbf{B}_4$ and $\mathbf{A}_5$ of patch $\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5$ are also points of patch $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$. With the test condition in Eq. (4), we know that a patch or subpatch is flat enough if it is located between two quadrilaterals that are ϵ away. Because boundary points $\mathbf{A}_2, \mathbf{C}_4, \mathbf{B}_4$ and $\mathbf{A}_5$ are on the limit surface, they must be located between two quadrilaterals that are ϵ away. So is the polygon $\mathbf{A}_1\mathbf{A}_2\mathbf{C}_4\mathbf{B}_4\mathbf{A}_5\mathbf{A}_6$. 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 $\frac{X_1}{R_1} \times \frac{X_2}{R_2} \times \frac{X_2}{R_3}$. 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 are very small, even with a low resolution, the voxel size is still very small.

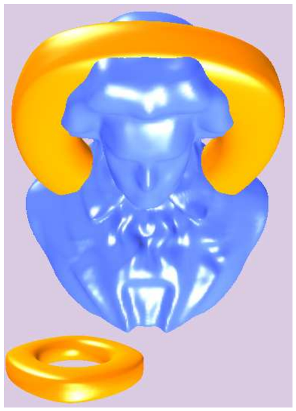
Therefore the overall error caused by polygonalization and voxelization is the sum of the errors caused by each of them. To make error of the final Boolean operation results less than the given ϵ everywhere, the test condition in eq. (4) has to be changed to the following form:

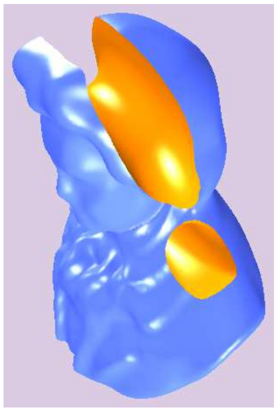
$$\begin{cases} \sqrt{d\ (\ \bar{u},\ \bar{v})} + \sqrt{d\ (\ \hat{u},\ \hat{v})} & \leq & \epsilon/2 \\ \text{size of each voxel} & \leq & \epsilon \end{cases} \qquad (5)$$

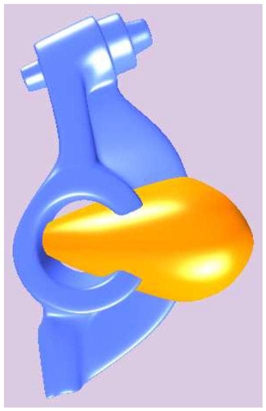
where $(\hat{u}, \hat{v})$ and $(\bar{u}, \bar{v})$ is defined the same way as in eq. (4). The first equation in eq. (5) ensures the patch (or subpatch) and its approximating polygon are both located inside two quadrilaterals that are $\epsilon/2$ away. The second equation in eq. (5) ensures the error caused by voxelization is not bigger than $\epsilon/2$. Hence the total error in the whole process is guaranteed to be less than ϵ.

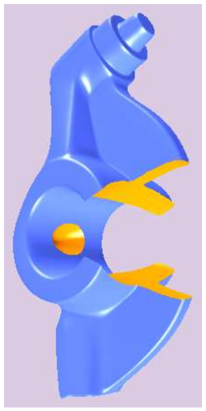
7 Test Results

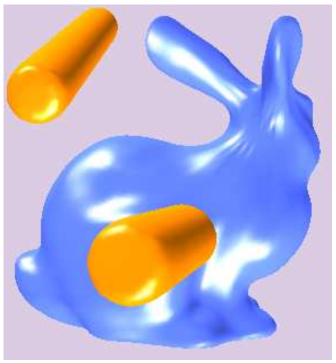
The proposed approach has been implemented in *C++* using *OpenGL* as the supporting graphics system on the Windows platform. Quite a few examples have been tested with the method described here. All the examples have extra-ordinary vertices. Some of the tested results are shown in Figures 4. The resolution of global voxelization is $512 \times 512 \times 512$ for all the test examples, and the error for all of them is set to 10^{-3}. The size of each example is normalized to $[0, 1]$ before voxelization and Boolean operations are

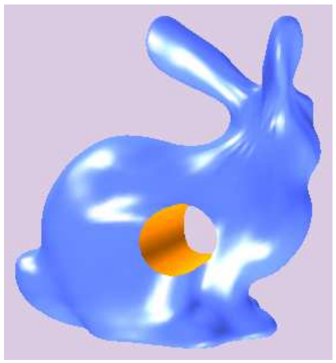

(a) Union

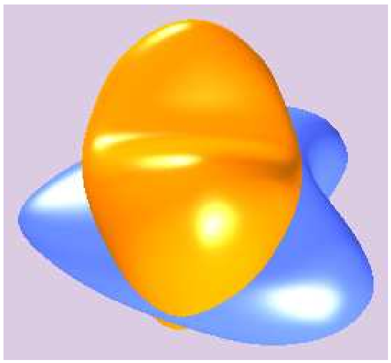

(b) Difference

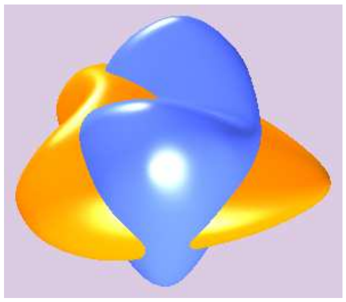

(c) Union

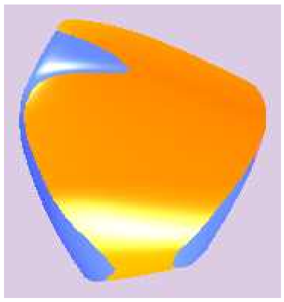

(d) Difference

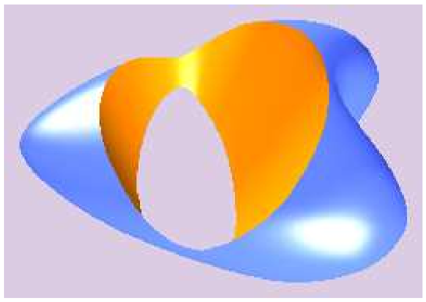

(e) Union

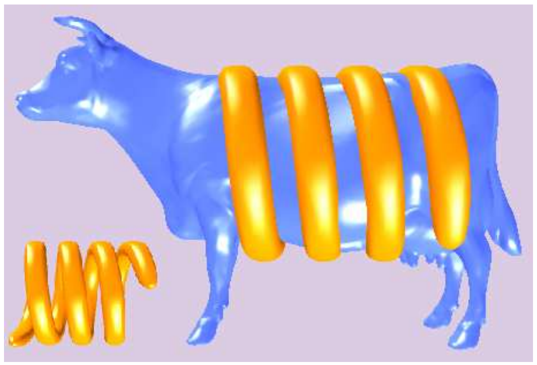

(f) Difference


(g) Union


(h) Union


(i) Intersection


(j) Difference



(k) Union

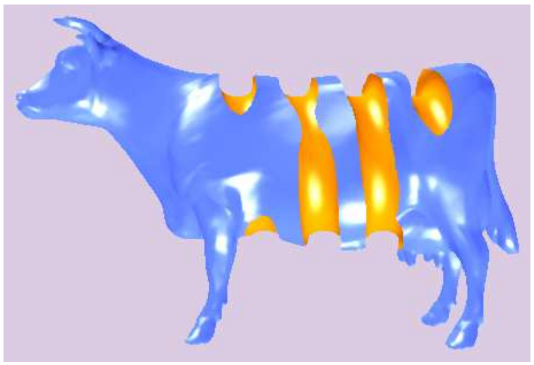

(l) Difference

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

performed. Resolutions of the local voxelization process depend on error tolerance and the given meshes. Hence resolution of local voxelization is different for each of the examples shown in Figures 4. For example, resolution of local voxelization used for Figures 4(k) and 4(l) is $8 \times 8 \times 8$, while for Figures 4(g), 4(h), 4(i) and 4(j) the resolution used for local voxelization is $32 \times 32 \times 32$. Although resolutions used for local voxelization are different, the overall error is the same in the final results. From eq. (5) we can see this difference is smaller than the error tolerance because that error is caused by voxelization and polygonalization as well.

In Figure 4, all the Difference and Intersection operations are performed on solids positioned exactly the same as in the Union operation so that we can easily tell if results of the Boolean operations are correct 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. Similarly, 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 with arbitrary 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 in our method are still represented with a continuous geometric representation. Hence our Boolean operation results can be scaled seamlessly and smoothly. Third, error of Boolean operation results can be precisely estimated. According to the error estimation formula, a secondary local voxelization can be performed for intersecting subpatches only. 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.

Acknowledgement. Data sets of Figures 4(a), 4(b), 4(e) and 4(f) are downloaded from research.microsoft.com/∼hoppe, and data sets of Figures 4(k) and 4(l) are downloaded from graphics.cs.uiuc.edu/∼garland/research/quadrics.html.

References

[1] Catmull E, Clark J. Recursively generated B-spline surfaces on arbitrary topological meshes, *Computer-Aided Design*, 1978, 10(6):350-355.

[2] Doo D, Sabin M, Behavior of recursive division surfaces near extraordinary points, Computer-Aided Design, 1978, 10(6):356-360.

[3] Loop CT, 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*, 2002, 18(5/6):299-315.

[7] Shuhua Lai, Fuhua (Frank) Cheng, Parametrization of General Catmull Clark Subdivision Surfaces and its Application, *Computer Aided Design & Applications*, 3, 1-4, 2006.

[8] Shuhua Lai, Fuhua (Frank) Cheng, Adaptive Rendering of Catmull-Clark Subdivision Surfaces, *9th Int. Conf. Computer Aided Design & Computer Graphics*, Hong Kong, 2005, 125-130.

[9] Sederberg TW, Zheng J, Sewell D, Sabin M, Non-uniform recursive subdivision surfaces, *Proceedings of 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] Jian Huang, Roni Yagel, V. Fillipov and Yair Kurzion, An Accurate Method to Voxelize Polygonal Meshes, *IEEE Volume Visualization'98*, October, 1998.

[12] Shuhua Lai, Fuhua (Frank) Cheng, Voxelization of Free-Form Solids using Catmull-Clark Subdivision Surfaces, www.cs.uky.edu/∼cheng/PUBL/voxel_new.pdf .

[13] S. Krishnan and D. Manocha, Computing Boolean Combinations of Solids Composed of Free-form Surfaces, *Proceedings of the 1996 ASME Design for Manufacturing Conference*, August 1996.

[14] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, Markus Gross, Surface Splatting, *SIGGRAPH 2001*.

[15] R. E. Barnhill, G. Farin, M. Jordan, and B. R. Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4(1-2):3-16, July 1987.

[16] Christoph Burnikel, Kurt Mehlhorn, and Stefan Schirra. On degeneracy in geometric computations. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp16-23, New York, 1994.

[17] Katrin Dobrindt, Kurt Mehlhorn, and Mariette Yvinec. A complete and efficient algorithm for the intersection of a general and a convex polyhedron. In *Algorithms and data structures*, pp314-324, 1993.

[18] Shankar Krishnan and Dinesh Manocha. An efficient surface intersection algorithm based on lower-dimensional formulation. *ACM Transactions on Graphics*, 16(1):74-106, January 1997.

[19] Ari Rappoport and Steven Spitz. Interactive Boolean operations for conceptual design of 3D solids. *Proceedings of SIGGRAPH 97*, pp269-278, 1997.

[20] T. Sederberg and T. Nishita. Geometric hermite approximation of surface patch intersection curves. *Computer Aided Geometric Design*, 8(2):97-114, 1991.

[21] N. M. Patrikalakis, Surface-to-surface Intersections, *IEEE Computer Graphics and Applications*, 13(1):89-95, 1993.

[22] Bieri, H., Nef,W., Elementary set operations with d-dimensional polyhedra. *Computational Geometry and its Applications*, LNCS 333, Springer-Verlag, 1988, pp. 97-112.

[23] Chazelle, B., An optimal algorithm for intersecting three dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671-696, 1992.

[24] Wiegand, T.F., Interactive rendering of CSG models. *Computer Graphics Forum*, 15(4):249-261, 1996.

[25] Duoduo Liao, Shiaofen Fang, Fast CSG Voxelization by Frame Buffer Pixel Mapping, *Proceedings of the 2000 IEEE Symposium on Volume Visualization*, pp. 43-48, 2000.

[26] Bart Adams, Philip Dutré, Interactive Boolean operations on surfel-bounded solids, *ACM 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., AND Zorin, D. 2001, Approximate Boolean operations on free-form solids. *ACM SIGGRAPH 2001*, pp185-194.