# Voxelization of 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 *voxelization technique* and its applications for objects with arbitrary topology are presented. The voxelization technique converts a free-form object from its continuous geometric representation into a set of voxels that best approximates the geometry of the object. Unlike traditional 3D scan-conversion based methods, the voxelization process in the new method is performed by recursively subdividing the 2D parameter space and sampling 3D points from selected 2D parameter space points. Because the new method can calculate every 3D point position explicitly and accurately, uniform sampling on surfaces with arbitrary topology is not a problem any more. Moreover, discretization of 3D closed objects in the new method is guaranteed to be leak-free when a 3D flooding operation is performed. This is ensured by showing the voxelization results satisfy the properties of *separability, accuracy* and *minimality*. In addition, a 3D *volume flooding algorithm* using dynamic programming techniques is presented which significantly speeds up the volume flooding process. Hence the new method is suitable for visualization of complex scenes, measuring object volume, mass, surface area, determining intersection curves of multiple surfaces and performing accurate Boolean/CSG operations. These capabilities are demonstrated by test examples shown in the paper.

**Keywords**: voxelization, subdivision, Catmull-Clark surfaces, visualization, parametrization.

## 1 Introduction

*Volume graphics* [9] represents voxel-based techniques aimed at modeling, manipulating and rendering of geometric objects. These techqniques have proven to be superior to traditional computer graphics approaches in many aspects. The main advantages of volume graphics include: (1) decoupling of voxelization from rendering, (2) uniformity of representation, and (3) support of Boolean, block and CSG operations. Two drawbacks of volume graphics techniques are their high memory and processing time demands. However, with the progress in both computers and specialized volume rendering hardware, these drawbacks are gradually losing their significance.

To be represented by the voxel raster, a geometric object has to go through a process called *voxelization*. This process is concerned with converting a geometric object from its continuous geometric representation into a set of voxels that best approximates the continuous object. Traditional voxelization methods (also referred to as 3D scan-conversion) mimic the 2D scan-conversion process that pixelizes (rasterizes) 2D geometric objects. Hence traditional voxelization methods work well for polygon based 3D objects. For surfaces with arbitrary topology, voxelization using 3D scan-conversion is not efficient, nor accurate.

Subdivision surfaces have become popular recently in graphical modeling, visualization and animation because of their capability in modeling/representing complex shape of arbitrary topology [1], their relatively high visual quality, and their stability and efficiency in numerical computation. 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. With parametrization techniques for subdivision surfaces becoming available [2, 5] and with the fact that non-uniform B-spline and NURBS surfaces are special cases of subdivision surfaces becoming known [7], we now know that 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.

In this paper we propose a voxelization method for free-form solids represented by Catmull-Clark subdivision surfaces. The new method is based on recursive sampling of 2D parameter space points of a surface patch, instead of direct sampling of 3D points. Hence the new method is more efficient and less sensitive to numerical error.

Note that a voxelization process does not render the voxels but merely generates a database of the discrete digitization of the continuous object [8]. Some previous voxelization methods use quad-trees to store the voxelization result. This approach can save memory space but might sacrifice in computation 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 an algorithm. People care more about the efficiency of the algorithm. The new method stores the voxelization result directly in a *Cubic Frame Buffer* for fast operation purpose.

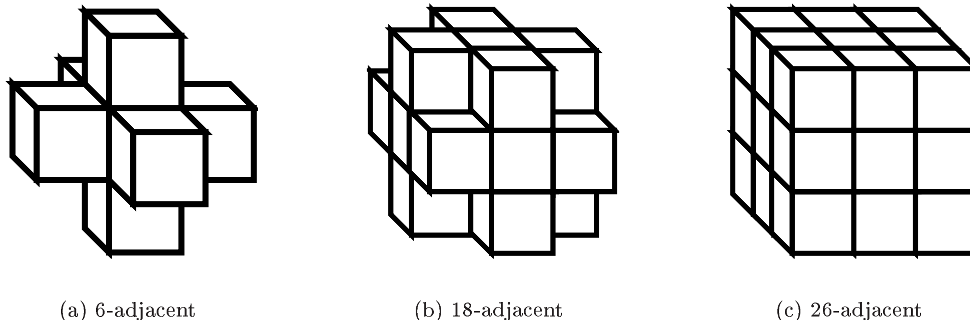
# 2 Background

## 2.1 3D Discrete Space

A 3D *discrete space* is a set of integral grid points in 3D Euclidean space defined by their Cartesian coordinates $(x, y, z)$, with $x, y, z \in Z$. A voxel is a unit cube centered at the integral grid point. Usually a voxel is assigned a value of 0 or 1. The voxels assigned an '1', called the 'black' voxels, represent opaque objects. Those assigned a '0', called the 'white' voxels, represent the transparent background. Outside the scope of this paper is a non-binary approach where the voxel values are mapped onto the interval [0,1] representing either partial coverage, variable densities, or graded opacities. Due to its larger dynamic range of values, this approach can support higher quality rendering.

Two voxels are said to be *26-adjacent* (See Fig. 1(c)) if they share a vertex, an edge, or a face. Every given voxel has 26 such adjacent voxels: eight share a vertex (corner) with the given voxel, twelve share an edge, and six share a face. Accordingly, face-sharing voxels are said to be *6-adjacent* (See Fig. 1(a)), and edge-sharing and face-sharing voxels are said to be *18-adjacent* (See Fig. 1(b)).

The prefix $N$ is used to define the adjacency relation, with $N$= 6, 18, or 26. A sequence of voxels having the same value (e.g., 'black') is called an *N-path* if all consecutive pairs are $N$-adjacent. A set of voxels are said to be *N-connected* if there is an $N$-path between every pair of its voxels. It is easy to see that $N$-connectedness is an equivalent relation. Given three disjoint sets of voxels $A$, $B$ and $C$, $A$ is said to *N-separate* $B$ and $C$ if any $N$-path from a voxel of $B$ to a voxel of $C$ intersects $A$.



(a) 6-adjacent (b) 18-adjacent (c) 26-adjacent

Figure 1: $N$-adjacency, $N \in \{6, 18, 26\}$.

## 2.2 Catmull-Clark Subdivision Surfaces

*Catmull-Clark subdivision scheme* provides a powerful method for building smooth and complex surfaces. Given a control mesh, a *Catmull-Clark subdivision surface* (CCSS) is generated by iteratively refining (subdividing) the control mesh to form new and finer control meshes [1]. The mesh refining process consists of defining new vertices and connecting the new vertices to form new edges and faces of a new control mesh. A CCSS is the limit surface

of the refined control meshes. The limit surface is called a *subdivision surface* because the mesh refining process is a generalization of the uniform B-spline surface *subdivision technique*. The *valence* of a mesh vertex is the number of mesh edges adjacent to the vertex. A mesh vertex is called an *extra-ordinary vertex* if its valence is different from four. A mesh face with an extra-ordinary vertex is called an *extra-ordinary face*. The *valance* of an extra-ordinary face is the valence of its extra-ordinary vertex. In the following, for the sake of simplicity, a mesh face and the corresponding surface patch will be treated the same and denoted by the same notation.

As we can see the number of faces in the uniformly refined meshes increases exponentially with respect to subdivision depth. Hence it is impossible to accurately sample 3D points directly on subdivided surfaces. Fortunately, parametrization techniques for subdivision surfaces have become available recently [2, 3, 4, 5]. Therefore efficient and accurate sampling for voxelization is not a problem any more. Given an extra-ordinary face $\mathbf{S}$, if the valence of its extra-ordinary vertex is $n$, then the surface patch corresponding to this extra-ordinary face is influenced by $2n + 8$ control vertices. One can use these control points to explicitly and accurately evaluate the position, normal and partical derivatives for any point of the limit surface patch. We will review the most recent parametrization techniques for CCSS in the next section.

# 3 Related Work

## 3.1 Voxelization Techniques

Voxelization techniques can be classified into two major categories. The first category consists of methods that extend the standard 2D scan-line algorithm and employ numerical considerations to guarantee that no gaps appear in the resulting discretization. As we know polygons are fundamental primitives in 3D surface graphics in that they approximate arbitrary surfaces as a mesh of polygonal patches. Hence, early work on voxelization focused on voxelizing 3D polygon meshes [10, 11, 12, 13, 14] by using 3D scan-conversion algorithm. Although this type of methods can be extended to voxelize parametric curves, surfaces and volumes [15], it is difficult to deal with freefrom surfaces of arbitrary topology.

The other widely used approach for voxelizing free-form solids is to use *spatial enumeration algorithms* which employ point or cell classification methods in either an exhaustive fashion or by recursive subdivision [18, 19, 20, 21]. However, 3D space subdivision techniques for models decomposed into cubic subspaces are computationally expensive and thus inappropriate for medium or high resolution grids. The voxelization technique that we will be presenting uses recursive subdivision. The difference is the new method performs recursive subdivision on 2D parameter space, not on the 3D object. Hence expensive distance computation between 3D points is avoided.

Like 2D pixelization, voxelization is a powerful technique for representing and modeling complex 3D objects. This is proved by many successful applications of volume graphics techniques in recently reported research work. For example, voxelization can be used for visualization of complex objects or scene [19]. It can also be used for measuring integral properties of solids, such as mass, volume and surface area [21]. Most importantly, it can be used for intersection curve calculation and performing accurate Boolean operations. For example, in [20, 22], a series of Boolean operations are performed on objects represented by a CSG tree. Voxelization is such an important technique that several hardware implementations of this technique have been reported recently [16, 17].

## 3.2 Evaluation of a CCSS Patch

Several approaches [2, 3, 4, 5] have been proposed for exact evaluation of an extraordinary patch at any parameter space point $(u, v)$. The parametrization technique presented in [5] will be used here. The representation scheme of this parametrization technique is explicit and uses only one half of the eigen basis functions in the representation. Therefore, it is computationally more efficient and can be used to compute tangents and normal for any point of the limit surface exactly and explicitly. Some most related results of [5] are summarized below.

The parametrization/evaluation approach in [5] is presented for general Catmull-Clark subdivision surface. That is, the new *vertex point* $\mathbf{V}'$ of $\mathbf{V}$ after one subdivision is computed as follows:

$$\mathbf{V}' = \alpha_n \mathbf{V} + \beta_n (\sum_{i=1}^{n} \mathbf{E}_i)/n + \gamma_n (\sum_{i=1}^{n} \mathbf{F}_i)/n$$

where $\alpha_n$, $\beta_n$ and $\gamma_n$ are positive numbers and $\alpha_n + \beta_n + \gamma_n = 1$. The new *face points* and *edge points* are computed the same way though.

The parametrization and evaluation of a surface patch can be written explicitly as follows [5].

$$\mathbf{S}(u,v) = W^T \mathrm{K}^m \sum_{j=0}^{n+5} \lambda_j^{m-1} \, \mathrm{M}_{b,j} \, G \tag{1}$$

where $n$ is the valance of the extraordinary patch, $W$ is a vector containing the 16 B-spline power basis functions and K is a constant diagonal matrix. $G$ is the vector of the $2n+8$ control points of the patch. In addition, $m$ and $b$ are two real numbers dependent on $(u,v)$ and can be calculated directly from $(u,v)$ [5]. $\lambda$ and $\mathrm{M}_{b,j}$ are independent of $(u,v)$ and their exact expressions are given in [5]. One can compute the derivatives of $\mathbf{S}(u,v)$ to any order simply by differentiating $W(u,v)$ in Eq. (1) accordingly. With the explicit expression of $S(u,v)$ and its partial derivatives, one can easily get the limit point of an extraordinary vertex in a general Catmull Clark subdivision surface:

$$\mathbf{S}(0,0) = [1,0,\cdots,0] \cdot M_{2,n+1} \cdot G$$

and the first and second derivatives:

$$\begin{aligned}
\mathbf{D}_u &= [0,1,0,0,0,0,0,\cdots,0] \cdot \mathrm{M}_{2,2} \cdot G \\
\mathbf{D}_v &= [0,0,1,0,0,0,0,\cdots,0] \cdot \mathrm{M}_{2,2} \cdot G \\
\mathbf{D}_{uu} &= [0,0,0,2,0,0,0,\cdots,0] \cdot \mathrm{M}_{2,2} \cdot G \\
\mathbf{D}_{uv} &= [0,0,0,0,1,0,0,\cdots,0] \cdot \mathrm{M}_{2,2} \cdot G \\
\mathbf{D}_{vv} &= [0,0,0,0,0,2,0,\cdots,0] \cdot \mathrm{M}_{2,2} \cdot G
\end{aligned}$$

where $\mathrm{M}_{2,n+1}$ and $\mathrm{M}_{2,2}$ are constant matrices of dimension $16 \times (2n+8)$ [5], $\mathbf{D}_u$, $\mathbf{D}_v$, $\mathbf{D}_{uu}$, $\mathbf{D}_{uv}$ and $\mathbf{D}_{vv}$ are the direction vectors of $\frac{\partial \mathbf{S}(0,0)}{\partial u}$, $\frac{\partial \mathbf{S}(0,0)}{\partial v}$, $\frac{\partial^2 \mathbf{S}(0,0)}{\partial u \partial u}$, $\frac{\partial^2 \mathbf{S}(0,0)}{\partial u \partial v}$ and $\frac{\partial^2 \mathbf{S}(0,0)}{\partial v \partial v}$, respectively. The normal vector at $(0,0)$ is the cross product of $\mathbf{D}_u$ and $\mathbf{D}_v$.

# 4 Voxelization based on Recursive Parameter Space Subdivision

## 4.1 Basic Idea

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 extra-ordinary vertex If this is not the case, simply perform Catmull-Clark subdivision on the control mesh of the CCSS twice.

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 voxels generated from this subpatch are the same as the voxels generated from its four corners:

$$\mathbf{V}_1 = \mathbf{S}(u_1,v_1), \quad \mathbf{V}_2 = \mathbf{S}(u_2,v_1), \quad \mathbf{V}_3 = \mathbf{S}(u_2,v_2), \quad \mathbf{V}_4 = \mathbf{S}(u_1,v_2).$$

Usually 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 from its four corners are not $N$-adjacent ($N \in \{6,18,26\}$) to each other, then there exist holes between them. In this case, the patch or subpatch is still not small enough. To make it smaller, 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:

$$\mathbf{S}([u_1,u_{12}] \times [v_1,v_{12}]), \quad \mathbf{S}([u_{12},u_2] \times [v_1,v_{12}]), \quad \mathbf{S}([u_{12},u_2] \times [v_{12},v_2]), \quad \mathbf{S}([u_1,u_{12}] \times [v_{12},v_2]),$$

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.

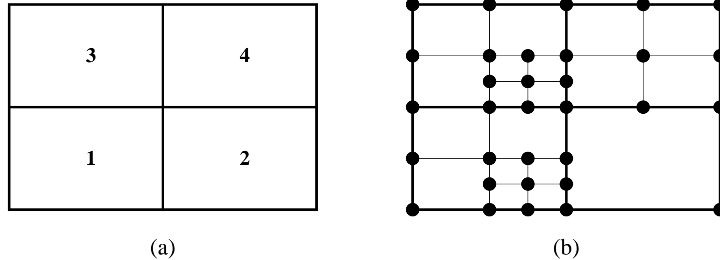



Figure 2: Basic idea of parameter space based recursive voxelization.

The vertices of the resulting subpatches after the recursive parameter space subdivision are then used as vertices for voxelization that approximates the limit surface. For example, if the four rectangles in Figure 2(a) are the parameter spaces of four adjacent subpatches of $\mathbf{S}(u, v)$, and if the rectangles shown in Figure 2(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. 2), direct and exact evaluation is performed to get its 3D surface position and normal vector at $S(u, v)$. To get the voxelized coordinates $(i, j, k)$ from $S(u, v)$, simply set

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

Once every single point marked in the recursive testing process is voxelized, the process for voxelizing the given patch is finished. The proof of the correctness of our voxelization results will be discussed in the next section.

Since the above process guarantees that 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 basis. 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 are very big in the given control mesh. Generating small subpatches is not a problem for a CCSS once the parametrization techniques are available. For example, in our implementation, the size of subpatches (in the parameter space) fed to recursive testing is $\frac{1}{8} \times \frac{1}{8}$, i.e. each patch is divided into $8 \times 8$ subpatches before the voxelization process. In addition, feeding small-size subpatches to the recursive testing process ensures the assumption of our voxelization process to be satisfied, because the smaller the parameter size of a subpatch, the flatter the subpatch.

### 4.2 Subpatch Testing

The basic assumption of our voxelization algorithm is that each subpatch is small enough (hence, flat enough) so that all the voxels generated from it are the same as the voxels generated using only the four corners of this subpatch. Now the problem is, for a given subpatch, how to tell if the assumption is satisfied or not. Usually this assumption does not hold. Hence a testing process must be performed before every 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, then there exist holes between them. In this case, the patch or subpatch is still not small enough. Conversely, we can say a given subpatch satisfies our basic assumption if the voxels generated using its four corners are $N$-adjacent ($N \in \{6, 18, 26\}$) to each other, because the given subpatch is at least $\mathrm{C}^1$ continuous, and is small enough (hence, flat enough). Therefore we can tell if a subpatch satisfies the basic assumption or not by testing the $N$-adjacency ($N \in \{6, 18, 26\}$) of the voxels generated by its four corners. For example, suppose $\Delta_i$, $\Delta_j$ and $\Delta_k$ are the maximum absolute differences of the voxels generated from the four corners in $x$, $y$ and $z$ directions, then for 6-adjacent voxelization, we can say the given subpatch satisfies our basic assumption if $(\Delta_i + \Delta_j + \Delta_k \le 1)$. Similarly, for 18-adjacent and 26-adjacent voxelization, the corresponding testing requirements are $(\Delta_i \le 1 \ \& \ \Delta_j \le 1 \ \& \ \Delta_k \le 1 \ \& \ \Delta_i + \Delta_j + \Delta_k \le 2)$ and $(\Delta_i \le 1 \ \& \ \Delta_j \le 1 \ \& \ \Delta_k \le 1)$, respectively.

Because $\Delta_i$, $\Delta_j$ and $\Delta_k$ are intergers, the testing process is very fast. In addition, the subpatch testing process generates a optimum partition of a given patch or subpatch in the voxelization process. For example Fig. 2(b) are the parameter spaces of the resulting subpatches when the above recursive testing process stops. 3D points will only be evaluated at the 2D parameter space points marked with small solid circles to form voxels that approximate the limit surface. Hence our testing process adaptively partitions given subpatches and evaluates 3D positions only if they are absolutely needed in the voxelization process. As a result, compared to other uniform 3D points sampling approaches, the cost of our adaptive 3D points sampling and evalation (which is the dominating cost of a voxelization process) is reduced dramtically.

The above testing process assumes when a given subpatch is small enough, then it is also flat enough. Certainly, this assumption might not hold in some very special cases. Hence other conditions, like flatness of a subpatch [6] can be included in the subpatch testing process as well. However, the voxelization process will be slowed down a lot in this way. All the examples we have tested show that it is good enough to use only the $N$-adjacency ($N \in \{6, 18, 26\}$) in the testing process.

### 4.3 Voxelization Algorithms

The above voxelization method, based on recursive subdivision of the parameter space, is summarized into the following algorithms: *Voxelization* and *VoxelizeSubPatch*. The parameters to these algorithms are defined as follows. $S$: control mesh of a CCSS which represents the given object; $N$: an integer that specifies the $N$-adjacency relationship between adjacent voxels; $M_1$, $M_2$, and $M_3$: resolution of the Cubic Frame Buffer; $k$: an integer that specifies the number of subpatches ($k \times k$) that should be generated before fed to the recursive voxelization process.

**Voxelization**(Mesh $S$, int $N$, int $M_1$, int $M_2$, int $M_3$, int k)
1. normalize $S$ so that $S$ is bounded by an axis-aligned cube of dimension $[0, M_1 - 1] \times [0, M_2 - 1] \times [0, M_3 - 1]$
2. for each patch $pid$ in $S$
3. for $u = \frac{1}{k} : 1$, step size $\frac{1}{k}$
4. for $v = \frac{1}{k} : 1$, step size $\frac{1}{k}$
5. VoxelizeSubPatch($N$, $pid$, $u - \frac{1}{k}$, $u$, $v - \frac{1}{k}$, $v$);

**VoxelizeSubPatch**(int $N$, int $pid$, float $u_1$, float $u_2$, float $v_1$, float $v_2$)
1. $(i_1, j_1, k_1)$ = Voxelize($S(pid, u_1, v_1)$);
2. $(i_2, j_2, k_2)$ = Voxelize($S(pid, u_2, v_1)$);
3. $(i_3, j_3, k_3)$ = Voxelize($S(pid, u_2, v_2)$);
4. $(i_4, j_4, k_4)$ = Voxelize($S(pid, u_1, v_2)$);
5. if($|u_2 - u_1| < 1/\max\{M_1, M_2, M_3\}$) return;
6. $\Delta_i = \max\{|i_a - i_b|\}$, with $a$ and $b \in \{1, 2, 3, 4\}$;
7. $\Delta_j = \max\{|j_a - j_b|\}$, with $a$ and $b \in \{1, 2, 3, 4\}$;
8. $\Delta_k = \max\{|k_a - k_b|\}$, with $a$ and $b \in \{1, 2, 3, 4\}$;
9. if($N = 6$ & $\Delta_i + \Delta_j + \Delta_k \le 1$) return;
10. if($N = 18$ & $\Delta_i \le 1$ & $\Delta_j \le 1$ & $\Delta_k \le 1$ & $\Delta_i + \Delta_j + \Delta_k \le 2$) return;
11. if($N = 26$ & $\Delta_i \le 1$ & $\Delta_j \le 1$ & $\Delta_k \le 1$) return;
12. $u_{12} = (u_1 + u_2)/2$; $v_{12} = (v_1 + v_2)/2$;
13. VoxelizeSubPatch($N, pid, u_1, u_{12}, v_1, v_{12}$);
14. VoxelizeSubPatch($N, pid, u_{12}, u_2, v_1, v_{12}$);
15. VoxelizeSubPatch($N, pid, u_{12}, u_2, v_{12}, v_2$);
16. VoxelizeSubPatch($N, pid, u_1, u_{12}, v_{12}, v_2$);

In algorithm 'VoxelizeSubPatch', corresponding surface points for the four corners are evaluated using eq. (1), where $pid$ tells us which patch we are currently working on. The routine 'Voxelize' voxelizes points by using eq. (2). Lines 9, 10 and 11 are used to test if voxelizing the four corners of a subpatch is enough to generate a 6-, 18- and 26-adjacent voxelization, respectively, while Line 5 prevents the recursive process from non-stop dead loop in case Lines 9, 10 and 11 are always not satisfied.

## 5 Separability, Accuracy and Minimality

Let $S$ be a $C^1$ continuous surface in $R^3$. We denote by $\bar{S}$ the discrete representation of $S$. $\bar{S}$ is a set of black voxels generated by some digitalization method. There are three major requirements that $\bar{S}$ should meet in the voxelization process. First, *separability* [8, 13], which requires to preserve the analogy between continuous and discrete space and to guarantee that $\bar{S}$ is not penetrable since $S$ is $C^1$ continuous. Second, *accuracy*. This requirement ensures that $\bar{S}$ is the most accurate discrete representation of $S$ according to some appropriate error metric. Third, *minimality* [8, 13], which requires the voxelization should not contain voxels that, if removed, make no difference in terms of separability and accuracy. The mathematical definitions for these requirements can be found in [13], which are based on [8].

First we can see that voxelization results generated using our recursive subdivision method satisfy the requirement of minimality. The reason is that voxels are sampled directly from the object surface. The termination condition of our recursive sampling process (i.e., Line 9, 10, 11 in algorithm 'VoxelizeSubPatch') and the coordinates transformation in eq. (2) guarantee that every point in the surface has one and only one image in the resulting voxelization. In other words,

$$\forall\ P \in S,\ \exists\ Q \in \bar{S}, \quad \text{such that} \quad P \in Q. \tag{3}$$

Note that here $P$ is a 3D point and $Q$ is a voxel, which is a unit cube. On the other hand, because all voxels are mapped directly from the object surface using eq. (2), we have

$$\forall\ Q \in \bar{S},\ \exists\ P \in S, \quad \text{such that} \quad P \in Q. \tag{4}$$

Hence no voxel can be removed from the resulting voxelization, i.e., the property of minimality is satisfied. In addition, from eq. (3) and eq. (4) we can also conclude that the resulting binary voxelization is the most accurate one with respect to the given resolution. Hence the property of accuracy is satisfied as well.

To prove that our voxelization results satisfy the separability property, we only need to show that there is no holes in the resulting voxelization. For simplicity, here we only consider 6-separability, i.e., there does not exist a ray from a voxel inside the free-form solid object to the outside of the free-form solid object in $x$, $y$ or $z$ direction that can penetrate our resulting voxelization without intersecting any of the black voxels. We prove the separability property by contradiction. As we know violating separability means there exists at least a hole (voxel) $Q$ in the resulting voxelization that is not included int $\bar{S}$ but is intersected by $S$ and, there must also exist two 6-adjacent neighbors of $Q$ that are not included in $\bar{S}$ either and are on opposite sides of $S$. Because $S$ intersects with $Q$, there exist at least one point $P$ on the surface that intersects with $Q$. But the image of $P$ after voxelization is not $Q$ because $Q$ is a hole. However, the image of $P$ after voxelization must exist because of the termination condition of our recursive sampling process (i.e., Line 8, 9, 10 in algorithm 'VoxelizeSubPatch'). Moreover, according to our voxelization method, $P$ can only be voxelized into voxel $Q$ because of eq. (2). Hence $Q$ cannot be a hole, contradicting our assumption. Therefore, we conclude that $\bar{S}$ is 6-separating.

## 6 Volume Flooding with Dynamic Programming

### 6.1 Seed Selection

A seed must be designated before a flooding algorithm can be applied. In 2D flooding, a seed is usually given by the user interactively. However, in 3D flooding, for a closed 3D object, it is impossible for a user to designate a voxel as a seed by mouse-clicking because voxels inside a closed 3D object are invisible. Hence an automatic method is needed to select an inside voxel as a seed for volume flooding. Once we can correctly choose an inside voxel, the by applying a flooding operation, all inside voxels can be obtained. To select a voxel as a seed for volume flooding, we need to tell if a voxel is inside or outside the 3D object. This is not a trivial problem. In the past In-Out test for voxels is not efficient and not accurate [21], especially for topologically complicated 3D objects.

With the availability of parametrization techniques for subdivision surfaces, we now can calculate derivatives and normals exactly and explicitly for each point located on the 3D object surface. Hence the normal for each voxel can also be exactly calculated in the voxelization process. Because the direction of a normal is perpendicular

to the surface and points towards the outside of the surface, the closest voxel in its opposite direction must be located either inside or on the surface (Assume the voxelization resolution is high enough). For a given voxel (called *start voxel*), to choose the closest voxel in its normal's opposite direction, we just need to calculate the dot product of its normal and one of the axis vectors. These vectors are: $\{1,0,0\}$, $\{-1,0,0\}$, $\{0,1,0\}$, $\{0,-1,0\}$, $\{0,0,1\}$, $\{0,0,-1\}$ corresponding to $x$, $-x$, $y$, $-y$, $z$ and $-z$ direction, respectively. The direction with smallest dot product is chosen for finding an inside voxel. If the closest voxel in this chosen direction is also a black voxel (i.e., located on the 3D object surface), another start voxel has to be selected and the above process is repeated until an inside voxel is found. The found inside voxel can be designated as a seed for inside volume flooding. Similarly, an outside voxel can also be found for outside volume flooding. In this case, the seed voxel should not be chosen from the normal's opposite direction, but along the normal's direction.

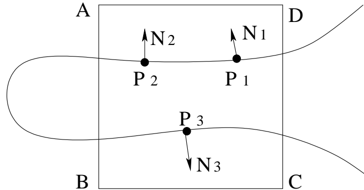



Figure 3: A voxel with multiple pieces of object surface in it.

However, if the voxelization resolution is not high enough, the closest voxel in the normal's opposite direction might be an outside voxel. For example, in Figure 3, $ABCD$ denotes a voxel and part of the object surface passes through this voxel. Differently, there are two pieces of surface that are not connected but are all inside this voxel. If we choose $P_1$ as the start point in Figure 3 to find an inside voxel using the above seed selection method, an outside voxel will be wrongly chosen. Hence the above method is no longer applicable in this case. To resolve the problem in this situation, higher voxelization resolution could be used. However, no matter how high the voxelization resolution is, we still cannot guarantee cases like the one shown in Figure 3 will not occur. Hence other approach is needed.

Fortunately, voxels that have multiple pieces of surface passing through, like the one shown in Figure 3, can be easily identified in the voxelization process. To identify these voxels, we need to calculate normals for each voxel. For example, in Figure 3, if surface point $P_1$ is mapped to voxel $ABCD$, then the normal at $P_1$ which is $N_1$, is also memorized as the normal of this voxel. Next time if another surface point, say $P_2$, is also mapped to voxel $ABCD$, then the normal at $P_2$ which is $N_2$, will be first compared with the memorized normal of voxel $ABCD$ by calculating their dot product. If $N_1 \cdot N_2 > 0$, then nothing need to be done. Otherwise, say surface point $P_3$, which is mapped to the same voxel and its normal is $N_3$, if $N_1 \cdot N_3 \leq 0$, then this voxel is marked as a voxel that has multiple piece passing through. Once every voxel that has multiple pieces of surface passing through is marked, we can easily solve the problem simply by not choosing these marked voxels as the start voxels.

## 6.2 3D Flooding using Dynamic Programming

In this section we only consider flooding algorithms using 6-separability, but the idea can be applied to $N$-separability with $N = 18$ or 26, Although 6-separability is used in the flooding process, the voxelization itself can be $N$-adjacent with $N = 6, 18$ or 26, Once a seed is chosen, 3D flooding algorithms can be performed in order to fill all the voxels that are 6-connected with this seed voxel. The simplest flooding algorithm is *recursive flooding*, which recursively search adjacent voxels in 6 directions for 6-connected voxels. This method sounds ideally reasonable but does not work in real world because even for a very low resolution, it would still cause stack overflow.

Another method that can be used for flooding is called *linear flooding*, which searches adjacent voxels that are 6-connected with the given the seed voxel, linearly from the first voxel to the last voxel in the cubic frame buffer, and marks all the found voxels with gray. The search process is repeated until no more white ('0') voxels is found that are 6-connected with one of the gray voxels. Linear flooding is simple and does not require extra memory in the flooding process. However, it is very slow, especially when a high resolution is used in the voxelization process.

In many applications, 3D flooding operations are required to be fast with low extra memory consumption. To make a 3D flooding algorithm applicable and efficient, we can combine the recursive flooding and the linear flooding methods using the so called dynamic programming technique.

Dynamic programming usually breaks a problem into subproblems, and these subproblems are solved and the solutions are memorized, in case they need to be solved again. This is the essentiality of dynamic programming. To use dynamic programming in our 3D flooding algorithm, we use a sub-routine *FloodingXYZ* which marks inside voxels having the same $x$, $y$ or $z$ coordinates as the given seed voxel, and all marked voxels are memorized by pushing them into a stack called *GRAYSTACK*. Note here the stack has a limited space, whose length is specified by the user. When the stack reaches its maximal capacity, no gray voxels can be pushed into it. Hence it guarantees limited memory consumption. The 3D flooding algorithm with dynamic programming can improve the flooding speed significantly. For ordinary resolution, say, $512 \times 512 \times 512$, a flooding operation can be done almost in real time. The pseudocode for the 3D volume flooding algorithm is given as follows and the parameters ($s_i$, $s_j$, $s_k$) are the coordinates of the given seed voxel.

**VolumeFlooding**(int $s_i$, int $s_j$, int $s_k$)
```
1.   FloodingXYZ(s_i, s_j, s_k);
2.   loop = 1;
3.   while(loop)
4.       while (GRAYSTACK is not empty)
5.           (i, j, k) = GRAYSTACK.Pop();
6.           FloodingXYZ(i, j, k)
7.       loop = 0;
8.       for(i = 0; i < M_1; i++)
9.           for(j = 0; j < M_2; j++)
10.              for(k = 0; k < M_3; k++)
11.                  if ( Voxel (i, j, k) is white and is 6-adjacent with a gray voxel)
12.                      FloodingXYZ(i, j, k);
13.                      loop = 1;
```

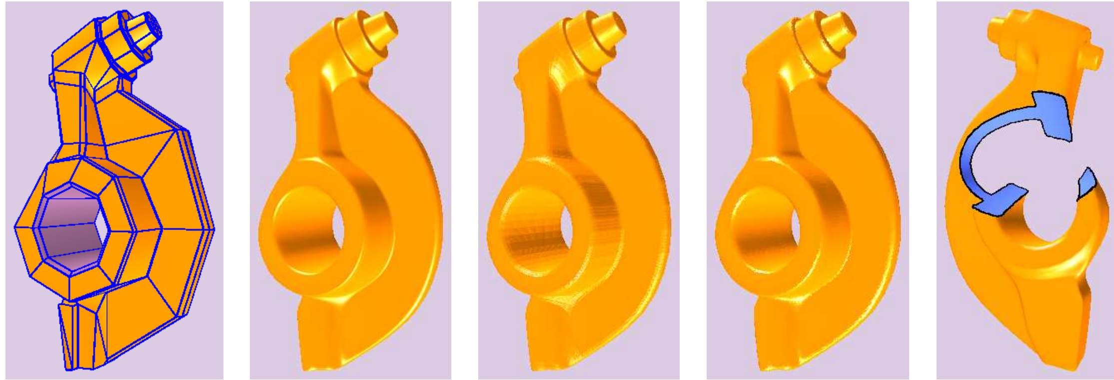
# 7 Applications

## 7.1 Visualization of Complex Scenes

Ray tracing is a commonly used method in the field of visualization of volume graphics. This is due to its ability to enhance spatial perception of the scene using techniques such as transparency, mirroring and shadow casting. However, there is a main disadvantage for ray tracing approach: large computational demands. Hence rending using this method is very slow. Recently, surface splatting technique for point based rendering has become popular [23]. Surface splatting requires the position and normal of every point to be known, but not their connectivity. With explicit position and exact normal information for each voxel in our voxelization results, now it is much easier for us to render discrete voxels using surface splatting techniques. The rendering is fast and high quality results can be obtained. For example, Fig. 4(a) is the given mesh, Fig. 4(b) is the corresponding limit surface. After the voxelization process, Fig. 4(c) is generated only using basic point based rendering techniques with explicitly known normals to each voxel. While Fig. 4(d) is rendered using splatting based techniques. The size of cubic frame buffer used for Fig. 4(c) is $512 \times 512 \times 512$. The voxelization resolution used for Fig. 4(d) is $256 \times 256 \times 256$. Although the resolution is much lower, we can tell from Fig. 4, that the one using splatting techniques is smoother and closer to the corresponding object surface given in Fig. 4(b).

## 7.2 Integral Properties Measurement

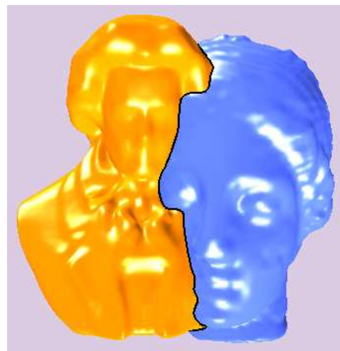
Another application of voxelization is that it can be used to measure integral properties of solid objects such as mass, volume and surface area. Without discretization, these integral properties are very difficult to measure, especially for free-form solids with arbitrary topology.



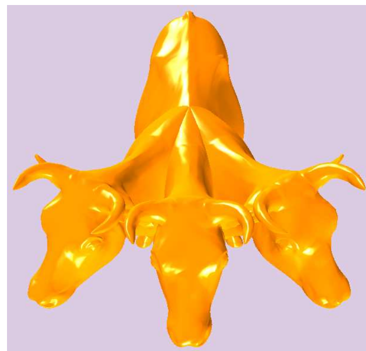
(a) Given Mesh (b) Object Surface (c) Point Based (d) Splatting Based (e) Difference

(f) Union

(g) Difference



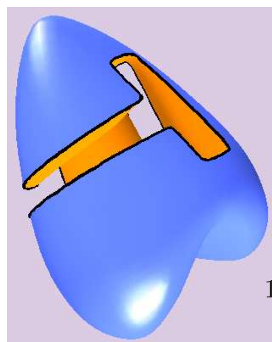
(h) Intersection Curve



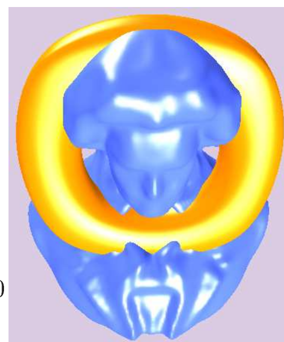
(i) Boolean Operations

(j) Boolean Operations

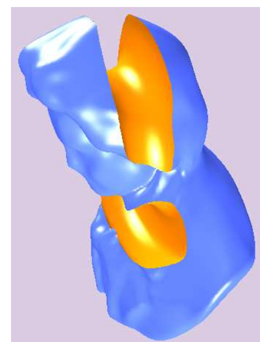
(k) CSG Operations



(l) Difference



(m) Union



(n) Difference

Volume can be measured simply by counting all the voxels inside or on the surface boundary because each voxel is a unit cube. With efficient flooding algorithm, voxels inside or on the boundary can be precisely counted. But the resulting measurement may not be accurate because boundary voxels do not occupy all the corresponding unit cubes. Hence for higher accuracy, higher voxelization resolution is needed. Once the volume is known, it is easy to measure the mass simply by multiplying the volume with density. Surface area can be measured similarly. But using this approach would lead to big error because we do not know how surfaces pass through their corresponding voxels. Fortunately, surface area can be measured much more precisely in the voxelization process. As we know, during the recursive voxelization process, if the recursive process stops, all the marked parameter points of a patch or subpatch (See Fig. 2) are points used for final voxelization. Hence all these quadrilaterals corresponding to these marked parameter points can be used for measuring surface area after these marked parameter space points are mapped to 3D space. The flatness of these quadrilaterals is required to be tested if high accuracy is needed. The definition of patch flatness and the flatness testing method can be found in [6].

### 7.3 Performing Boolean and CSG Operations

The most important application of voxelization is to perform Boolean and CSG operations on free-form objects. In solid modeling, an object is formed by performing Boolean operations on simpler objects or primitives. A CSG tree is used in recording the construction history of the object and is also used in the ray-casting process of the object. Surface-surface intersection (including the in-on-out test) and ray-surface intersection are the core operations in performing the Boolean and CSG operations. With voxelization, all of these problems become much easier set operations. For instance, Fig. 4(e) is generated by subtracting a heart model shown in Fig. 4(l) from the rocker arm model shown in Fig. 4(b). And Fig. 4(l) is the difference of a heart model and the rocker arm model shown in Fig. 4(b). While Fig. 4(f) and Fig. 4(g) are the union and difference results of the cow model and the rocker arm model shown in Fig. 4(b). And Fig. 4(m) and Fig. 4(n) are generated by uniting/subtracting a torus model from the blue model. Note that all these union and difference pairs are positioned the same way when Boolean operations are performed. Examples of performing multiple Boolean operations on models are shown in Fig. 4(i) and Fig. 4(j). A difference operation is first performed to remove some portions from each of these cows and a union operation is then performed to join them together. A mechanical part is also generated in Fig. 4(k) using CSG operations. Intersection curves can be similarly generated by searching for common voxels of objects. The black curves shown in Fig. 4(h), Fig. 4(e) and Fig. 4(l) are the intersection curves generated from two different objects.

## 8 Summary

A method to convert a free-form object from its continuous geometric representation to a set of voxels that best approximates the geometry of the object is presented. Unlike traditional 3D scan-conversion based methods, the new method does the voxelization process by recursively subdividing the 2D parameter space and sampling 3D surface points only at selected 2D parameter space positions. Because of the capability to calculate every 3D point position explicitly and accurately, uniform sampling on surfaces with arbitrary topology is not a problem for the approach at all. Moreover, the new method guarantees that discretization of 3D closed objects is leak-free when a 3D flooding operation is performed. This is ensured by proving that voxelization results of the new method satisfy the properties of separability, accuracy and minimality. In addition, a 3D volume flooding algorithm using dynamic programming techniques is presented which significantly speeds up the volume flooding process. Hence the new method is suitable for visualization of complex scenes, measuring object volume, mass, surface area, determining intersection curve of multiple surfaces and performing accurate Boolean/CSG operations.

**Acknowledgement.** Data sets for Figs. 4(h), 4(m) and 4(n) are downloaded from the web site:
research.microsoft.com/∼hoppe.
The data set for the cow model in Fig. 4 is downloaded from the web site:
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] Stam J, Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values, *Proceedings of SIGGRAPH* 1998:395-404.

[3] Stam J, Evaluation of Loop Subdivision Surfaces, *SIGGRAPH'99 Course Notes*, 1999.

[4] Zorin D, Kristjansson D, Evaluation of Piecewise Smooth Subdivision Surfaces, *The Visual Computer*, 2002, 18(5/6):299-315.

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

[6] 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.

[7] Sederberg TW, Zheng J, Sewell D, Sabin M, Non-uniform recursive subdivision surfaces, *Proceedings of SIGGRAPH*, 1998:19-24.

[8] Cohen Or, D., Kaufman, A., Fundamentals of Surface Voxelization, *Graphical Models and Image Processing*, 57, 6 (November 1995), 453-461.

[9] A. Kaufman, D. Cohen. Volume Graphics. *IEEE Computer*, Vol. 26, No. 7, July 1993, pp. 51-64.

[10] D. Haumont and N. Warzee. Complete Polygonal Scene Voxelization, *Journal of Graphics Tools*, Volume 7, Number 3, pp. 27-41, 2002.

[11] M.W. Jones. The production of volume data from triangular meshes using voxelisation, *Computer Graphics Forum*, vol. 15, no 5, pp. 311-318, 1996.

[12] S. Thon, G. Gesquiere, R. Raffin, A low Cost Antialiased Space Filled Voxelization Of Polygonal Objects, *GraphiCon 2004*, pp. 71-78, Moscou, Septembre 2004.

[13] Jian Huang, Roni Yagel, V. Fillipov and Yair Kurzion, An Accurate Method to Voxelize Polygonal Meshes, *IEEE Volume Visualization'98*, October, 1998.

[14] Kaufman, A., An Algorithm for 3D Scan-Conversion of Polygons, *Proc. EUROGRAPHICS'87*, Amsterdam, Netherlands, August 1987, 197-208.

[15] Kaufman, A., Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes, *Computer Graphics*, 21, 4 (July 1987), 171-179.

[16] S. Fang and H. Chen. Hardware accelerated Voxelisation. *Volume Graphics*, Chapter 20, pp. 301-315. Springer-Verlag, March, 2000.

[17] Beckhaus S., Wind J., Strothotte T., Hardware-Based Voxelization for 3D Spatial Analysis *Proceedings of CGIM '02*, pp. 15-20, August 2002.

[18] N. Stolte, A. Kaufman, Efficient Parallel Recursive Voxelization for SGI Challenge Multi-Processor System, *Computer Graphics International*, 1998.

[19] Nilo Stolte, Graphics using Implicit Surfaces with Interval Arithmetic based Recursive Voxelization, *Computer Graphics and Imaging*, pp. 200-205, 2003.

[20] T. Duff, Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry, *SIGGRAPH*, pp. 131–138, July, 1992.

[21] Lee, Y. T. and Requicha, A. A. G., Algorithms for Computing the Volume and Other Integral Properties of Solids: I-Known Methods and Open Issues; II-A Family of Algorithms Based on Representation Conversion and Cellular Approximation, *Communications of the ACM*, 25, 9 (September 1982), 635-650.

[22] S. Fang and D. Liao. Fast CSG Voxelization by Frame Buffer Pixel Mapping. *ACM/IEEE Volume Visualization and Graphics Symposium 2000 (Volviz'00)*, Salt Lake City, UT, 9-10 October 2000, 43-48.

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