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
{slai2, cheng}@cs.uky.edu
www.cs.uky.edu/∼cheng

Abstract. A voxelization technique and its applications for objects with arbitrary topology are presented. It 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, our voxelization method is performed by recursively subdividing the 2D parameter space and sampling 3D points from selected 2D parameter space points. Moreover, our voxelization of 3D closed objects is guaranteed to be leak-free when a 3D flooding operation is performed. This is ensured by proving that our voxelization results satisfy the properties of separability, accuracy and minimality.

1   Introduction

Volume graphics [5] represents a set of techniques aimed at modeling, manipulating and rendering of geometric objects, which have proven to be, in many aspects, superior to traditional computer graphics approaches. The main 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.

Subdivision surfaces have become popular recently in graphical modeling, visualization and animation because of their capability in modeling 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. In this paper we propose a voxelization method for free-form solids represented by Catmull-Clark subdivision surfaces. Instead of direct sampling of 3D points, the new method is based on recursive sampling of 2D parameter space points of a surface patch. Hence the new method is more efficient and less sensitive to numerical error.

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. Two voxels are said to be *26-adjacent* [4] 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* [4], and edge-sharing and face-sharing voxels are said to be *18-adjacent* [4].

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 number of faces in the uniformly refined meshes increases exponentially with respect to subdivision depth. Hence it is not practical to sample 3D points directly on subdivided surfaces. Fortunately, parametrization techniques have become available recently [2,3]. Therefore efficient and accurate sampling for voxelization is not a problem any more. Recent parametrization techniques show that every 3D point (its position, normal and partial derivatives) on the limit surface can be explicitly and accurately calculated [2,3].

2   Related 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. Most work on voxelization focused on voxelizing 3D polygon meshes [6,7,8,9] by using 3D scan-conversion algorithm. Although this type of methods can be extended to voxelize parametric curves, surfaces and volumes, 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 [10]. 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.

3   Voxelization of Solids Represented by CCSSs

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.

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 [2,3]. Consequently, any point in the surface can be explicitly calculated. On the other hand, for any given parameter space point (u, v), a surface point $S(u, v)$ corresponding to this parameter

space point can be exactly computed as well. 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 all the voxels generated from it are the same as the voxels generated using its four corners:

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

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 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. To make it smaller, we perform a *midpoint subdivision* on the corresponding parameter space by setting $u_{12} = (u_1+u_2)/2$ and $v_{12} = (v_1+v_2)/2$ to get four smaller subpatches:

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

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. For simplicity, 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, 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

$$i = \lfloor \mathbf{S}(u, v).x + 0.5 \rfloor, j = \lfloor \mathbf{S}(u, v).y + 0.5 \rfloor, k = \lfloor \mathbf{S}(u, v).z + 0.5 \rfloor. \qquad (1)$$

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 patch basis.

The above voxelization method, based on recursive subdivision of the parameter space, is summarized into the following algorithms: *Voxelization* and *VoxelizeSubPatch*. The parameters of these algorithms are defined below. S: control mesh of a CCSS which represents the given object; N: an integer that specifies the N-adjacent 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 $[0, M_1 - 1] \times [0, M_2 - 1] \times [0, M_3 - 1]$
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. $(i_1, j_1, k_1) = \text{Voxelize}(S(pid, u_1, v_1))$; $(i_2, j_2, k_2) = \text{Voxelize}(S(pid, u_2, v_1))$;
2. $(i_3, j_3, k_3) = \text{Voxelize}(S(pid, u_2, v_2))$; $(i_4, j_4, k_4) = \text{Voxelize}(S(pid, u_1, v_2))$;
3.   if(the size of this subpatch is smaller than a voxel)   return;
4. $\Delta_i = \max\{|i_a - i_b|\}$, with a and $b \in \{1, 2, 3, 4\}$;
5. $\Delta_j = \max\{|j_a - j_b|\}$, with a and $b \in \{1, 2, 3, 4\}$;
6. $\Delta_k = \max\{|k_a - k_b|\}$, with a and $b \in \{1, 2, 3, 4\}$;
7.   if($N = 6$ & $\Delta_i + \Delta_j + \Delta_k \leq 1$)   return;
8.   if($N = 18$ & $\Delta_i \leq 1$ & $\Delta_j \leq 1$ & $\Delta_k \leq 1$ & $\Delta_i + \Delta_j + \Delta_k \leq 2$)   return;
9.   if($N = 26$ & $\Delta_i \leq 1$ & $\Delta_j \leq 1$ & $\Delta_k \leq 1$)   return;
10. $u_{12} = (u_1 + u_2)/2$; $v_{12} = (v_1 + v_2)/2$;
11. VoxelizeSubPatch($N, pid, u_1, u_{12}, v_1, v_{12}$);
12. VoxelizeSubPatch($N, pid, u_{12}, u_2, v_1, v_{12}$);
13. VoxelizeSubPatch($N, pid, u_{12}, u_2, v_{12}, v_2$);
14. VoxelizeSubPatch($N, pid, u_1, u_{12}, v_{12}, v_2$);

In algorithm 'VoxelizeSubPatch', corresponding surface points for the four corners are directly evaluated using parametrization techniques in [2,3], where pid tells us which patch we are currently working on. The routine 'Voxelize' voxelizes points by using eq. (1). Lines 7, 8 and 9 are used to test if voxelizing the four corners of a subpatch is enough to generate a 6-, 18- and 26-adjacent voxelization, respectively.

4   Separability, Accuracy and Minimality

Let $\mathbf{S}$ be a C^1 continuous surface in R^3. We denote by $\bar{\mathbf{S}}$ the discrete representation of $\mathbf{S}$. $\bar{\mathbf{S}}$ is a set of black voxels generated by some digitalization method. There are three major requirements that $\bar{\mathbf{S}}$ should meet in the voxelization process. First, *separability* [4,9], which requires preservation of the analogy between continuous and discrete space and to guarantee that $\bar{\mathbf{S}}$ is not penetrable since $\mathbf{S}$ is C^1 continuous. Second, *accuracy*. This requirement ensures that $\bar{\mathbf{S}}$ is the most accurate discrete representation of $\mathbf{S}$ according to some appropriate error metric. Third, *minimality* [4,9], which requires the voxelization does 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 [4,9].

First we can see that the voxelization results generated using our recursive subdivision method satisfy the minimality property. The reason is that voxels

are sampled directly from the object surface. The termination condition of our recursive sampling process (i.e., Line 8, 9, 10 in algorithm 'VoxelizeSubPatch') and the coordinates transformation in eq. (1) guarantee that every point in the surface has one and only one image in the resulting voxelization. In other words,

$$\forall \ \mathbf{P} \in \mathbf{S}, \ \exists \ \mathbf{Q} \in \bar{\mathbf{S}}, \ \ \text{such that} \ \ \mathbf{P} \in \mathbf{Q}. \tag{2}$$

Note that here $\mathbf{P}$ is a 3D point and $\mathbf{Q}$ is a voxel, which is a unit cube. On the other hand, because all voxels are mapped directly from the object surface using eq. (1), we have

$$\forall \ \mathbf{Q} \in \bar{\mathbf{S}}, \ \exists \ \mathbf{P} \in \mathbf{S}, \ \ \text{such that} \ \ \mathbf{P} \in \mathbf{Q}. \tag{3}$$
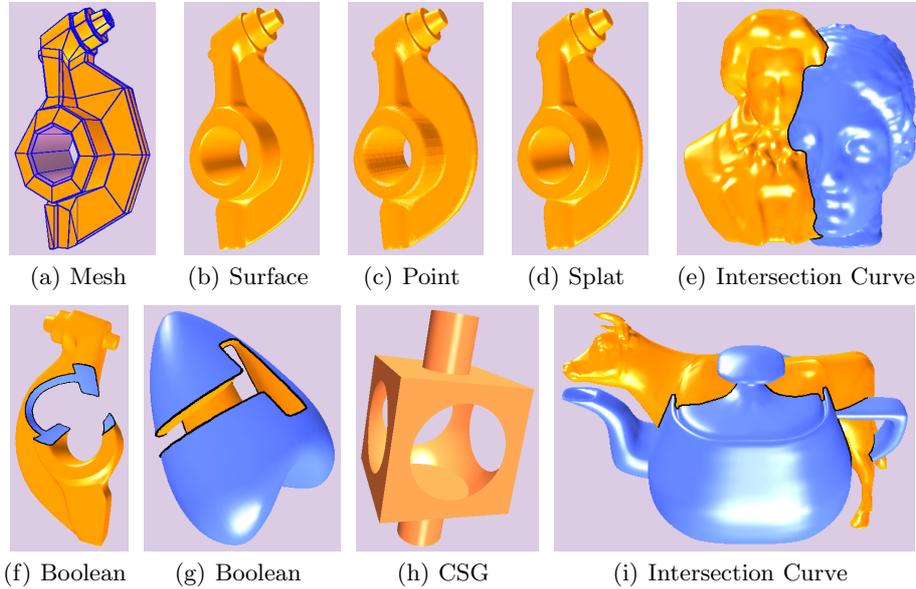
Hence no voxel can be removed from the resulting voxelization, i.e., the property of minimality is satisfied. In addition, from eq. (2) and eq. (3) we can also conclude that the resulting voxelization is the most accurate one with respect to the given resolution. Hence the property of accuracy is satisfied 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) $\mathbf{Q}$ in the resulting voxelization that is not included int $\bar{\mathbf{S}}$ but is intersected by $\mathbf{S}$ and, there must also exist two 6-adjacent neighbors of $\mathbf{Q}$ that are not included in $\bar{\mathbf{S}}$ either and are on opposite sides of $\mathbf{S}$. Because $\mathbf{S}$ intersects with $\mathbf{Q}$, there exist at least one point $\mathbf{P}$ on the surface that intersects with $\mathbf{Q}$. But the image of $\mathbf{P}$ after voxelization is not $\mathbf{Q}$ because $\mathbf{Q}$ is a hole. However, the image of $\mathbf{P}$ 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, $\mathbf{P}$ can only be voxelized into voxel $\mathbf{Q}$ because of eq. (1). Hence $\mathbf{Q}$ cannot be a hole, contradicting our assumption. Therefore, we conclude that $\bar{\mathbf{S}}$ is 6-separating.

5   Applications

5.1   Visualization of Complex Scenes

Ray tracing is a commonly used method in the field of visualization of volume graphics [6]. However, ray tracing is very slow due to its large computational demands. Recently, surface splatting technique for point based rendering has become popular [11]. 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 being available, now it is quite easy for us to render discrete voxels using surface splatting

(a) Mesh   (b) Surface   (c) Point   (d) Splat   (e) Intersection Curve

(f) Boolean   (g) Boolean   (h) CSG   (i) Intersection Curve

Fig. 1. Applications of Voxelization

techniques. The rendering is fast and high quality results can be obtained. For example, Fig. 1(a) is the given mesh, Fig. 1(b) is the corresponding limit surface. After the voxelization process, Fig. 1(c) is generated only using basic point based rendering techniques with explicitly known normals to each voxel. While Fig. 1(d) is rendered using splatting based techniques. The size of cubic frame buffer used for Fig. 1(c) is $512 \times 512 \times 512$. The voxelization resolution used for Fig. 1(d) is $256 \times 256 \times 256$. Although the resolution is much lower, we can tell from Fig. 1, that the one using splatting techniques is smoother and closer to the corresponding object surface given in Fig. 1(b).

5.2   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.

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 as well.

5.3   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 simply become easier set operations. Examples of performing Boolean operations on two objects are presented in Fig. 1(f) and Fig. 1(g), respectively. Fig. 1(f) is the difference of a rocker arm shown in Fig. 1(b) and a heart shown in Fig. 1(g), while Fig. 1(g) is the difference of a heart and a rocker arm shown in Fig. 1(b). A mechanical part is also generated in Fig. 1(h) using CSG operations. Intersection curves can be similarly generated by searching for common voxels of objects. The black curves shown in Fig. 1(f), Fig. 1(g), Fig. 1(i) and Fig. 1(e) are the intersection curves generated from two different objects, respectively.

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. Shuhua Lai, Fuhua (Frank) Cheng, Parametrization of General CCSSs and its Application, *Computer Aided Design & Applications*, 3, 1-4, 2006.
4. Cohen Or, D., Kaufman, A., Fundamentals of Surface Voxelization, *Graphical Models and Image Processing*, 57, 6 (November 1995), 453-461.
5. A. Kaufman, D. Cohen. Volume Graphics. *IEEE Computer*, Vol. 26, No. 7, July 1993, pp. 51-64.
6. D. Haumont and N. Warzee. Complete Polygonal Scene Voxelization, *Journal of Graphics Tools*, Volume 7, Number 3, pp. 27-41, 2002.
7. M.W. Jones. The production of volume data from triangular meshes using voxelisation, *Computer Graphics Forum*, vol. 15, no 5, pp. 311-318, 1996.
8. S. Thon, G. Gesquiere, R. Raffin, A low Cost Antialiased Space Filled Voxelization Of Polygonal Objects, *GraphiCon 2004*, pp. 71-78, Moscou, Septembre 2004.
9. Jian Huang, Roni Yagel, V. Fillipov and Yair Kurzion, An Accurate Method to Voxelize Polygonal Meshes, *IEEE Volume Visualization'98*, October, 1998.
10. Nilo Stolte, Graphics using Implicit Surfaces with Interval Arithmetic based Recursive Voxelization, *Computer Graphics and Imaging*, pp. 200-205, 2003.
11. Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, Markus Gross, Surface Splatting, *SIGGRAPH 2001*, pp. 371-378.