Shadow Generation for Objects Represented by Catmull-Clark Subdivision Surfaces

Shuhua Lai
Department of Computer Information Systems, Virginia State University, Petersburg, VA 23806
Fuhua (Frank) Cheng
Department of Computer Science, University of Kentucky, Lexington, KY 40506

Abstract. A method for generating good quality shadows for objects represented by Catmull-Clark subdivision surfaces (CCSSs) is presented. The new method is based on combining a *vexolization* technique and a *projection* technique. Like the *shadow mapping method*, one can easily tell that, after the projection, the voxels that are closest to the light are not in shadow. But it is difficult to tell if the remaining voxels are in shadow without other information. In this paper this problem is solved by augmenting the projection phase with information obtained during the voxelization phase. Two main approaches are presented: *offset-based* that uses position and normal information, and *topology-based* which uses voxel connectivity information. The shadow determination process is also facilitated by storing voxelization results in a *cubic framebuffer* directly. Since our voxelization process is performed in the parameter spaces of the CCSSs instead of the object space, the process is very fast and efficient. As a result, the overall shadow generation process is fast, efficient and robust. The new method is presented for CCSSs only, but the concept works for any subdivision schemes whose limit surfaces are parametrizable.

1 Introduction

A shadow is an area of relative darkness in an illuminated region caused by an object totally or partially occluding the light [23]. Shadows provide clues about the shapes, relative positions and surface characteristics of the objects. They can also indicate the approximate location, intensity, shape and size of the light source(s). Hence, the presence of shadows in a scene helps convey realism and aids depth perception.

Shadow determination is intrinsically a visibility determination process, except it is done with respect to the light source instead of the view point. *Shadow mapping* is a popular method in shadow generation. The concept was first introduced by Lance Williams [18] in 1978. It has since been used in many applications, including commercial softwares such as Pixar's *RenderMan* and high-end PC games. This concept is also supported by some commercial GPUs, like *nVidia*.

In shadow mapping, shadows are created by testing if pixels visible from the view point are visible from the light source. This is usually done as a two-phase process: a z-buffering with respect to the light source and a z-buffering with respect to the view point. Visible pixels identified in the second phase are compared with the corresponding entries of the z-buffer (or, *depth image*) obtained in the first phase to determine if they are in shadow. The depth image is usually stored in the form of a *texture*. A typical problem with this approach is, because of numerical error, a pixel with larger z value might actually be in light. Consider, for example, the three points A, B and C of the plane P in Figure 1. Since A, B and C are not blocked by anything when viewed from the light source, they should all be in light. However, because of the round-off error, when projected onto the projection plane, they all fall onto the same entry on the projection plane. Hence, according to the algorithm, A and B should be in shadow. As a result, most part of the plane P would be rendered in shadow. Hence the rendering result in this case would convey an incorrect perception of the scene. Solving this type of problem has always been a challenging task for shadow generation process involving discretization and projection.

In this paper, we study the problem of shadow generation for objects represented by Catmull-Clark subdivision surfaces (CCSSs) [1]. Our approach also involves discretization and projection. But the discretization process is not done through the z-buffering process, but by *voxelizing* the CCSSs. Unlike the *scan conversion process* in z-buffering, the voxelization process is not part of the rendering process [10, 11, 12], but a way to build a representation for an object (even

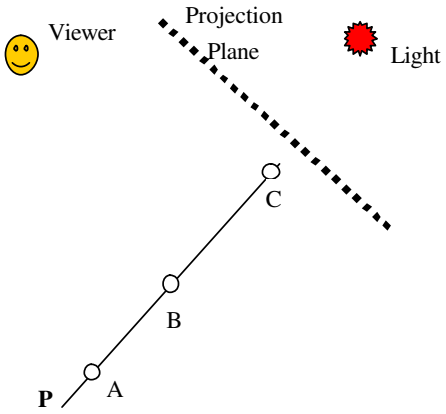


Figure 1: Scenario in which the result of shadow mapping is not correct (Intersection View)

though such a representation can make the rendering of an object easier in some cases. But a main reason of voxelization is for modeling purpose). Therefore, separate rendering (and shadow generation) process has to be developed for the results of the voxelization process. The shadow generation process in this paper is done through a projection process, similar to the shadow mapping method. Therefore, one would face the same problem as the one depicted in Figure 1. We solve this by augmenting the projection phase with information obtained during the voxelization phase. Two approaches are presented: an *offset-based* approach and a *topology-based* approach. The first approach uses position and normal information while the second approach uses voxel connectivity information. Test results show that both approaches work well.

The remaining part of the paper is arranged as follows. A brief review of background and previous works in this area are given in Section 2. A description of our voxelization technique is given in Section 3. Two approaches for shadow determination are presented in Section 4. Rendering issues critical to the shadow generation process are discussed in Section 5. Implementation details and test results are shown in Section 6. The concluding remarks are given in Section 7.

2 Background & Related Work

2.1 Catmull-Clark 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. The control mesh of a subdivision surface can contain vertices whose *valences* (numbers of adjacent edges) are different from four. Those vertices are called *extra-ordinary vertices*. Popular subdivision surfaces include Catmull-Clark subdivision surfaces (CCSSs) [1], Doo-Sabin subdivision surfaces [2] and Loop subdivision surfaces [3].

Subdivision surfaces can model/represent complex shape of arbitrary topology because there is no limit on the shape and topology of the control mesh of a subdivision surface. Subdivision surfaces are intrinsically discrete. Recently it was proved that subdivision surfaces can also be parametrized [4, 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 most general surface representation scheme. They include non-uniform B-spline and NURBS surfaces as special cases [9]. In this paper we only consider objects represented by CCSSs. But our approach works for any subdivision scheme whose limit surfaces are parametrizable.

2.2 Voxelization

Like pixelization of 2D items, voxelization of 3D surfaces [10, 11] is a powerful technique for representing and modeling complex 3D objects. This is proved by many successful applications of volume graphics techniques reported recently. For example, voxelization can be used for visualization of complex objects or scenes [12, 14, 15] and shadow determination [16, 17, 20, 21]. It can also be used for measuring integral properties of solids, such as mass, volume and surface area. It can be used for Boolean operations of free-form objects as well [13].

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

separability and accuracy. The mathematical definitions of these criteria can be found in [10, 11].

The widely used approach in voxelizing free-form solids is *spatial enumeration algorithms* which employ point or cell classification methods in an exhaustive fashion or by recursive subdivision. However, 3D space subdivision techniques for models decomposed into cubic subspaces are computationally expensive and thus are inappropriate for medium or high resolution grids. Our voxelization technique [12] also uses recursive subdivision. The difference is that our method performs recursive subdivision in 2D parameter space, not in 3D object space. Hence expensive distance computation between 3D points is avoided. It has been proven in [12] that our voxelization method satisfies the above three requirements. Hence, the result of our voxelization method is leak-free.

2.3 Shadow Generation

Shadow generation is an important area of computer graphics and has been extensively studied [23]. The most popular methods are *shadow mapping* [18], *shadow volume* [19] and *ray tracing* [20]. The shadow volume method is object space based. For complex scene, it takes longer time to render and generate shadows. The ray tracing method is very floating point intensive and, consequently, is expensive and numerical error prone. The shadow mapping method is an image space based method. The rendering of a shadowed scene in this case involves two steps. The first step produces a *shadow map* (depth map) by rendering the scene from the light source. The shadow map is often stored as a *texture* in the graphics card memory. The second step applies the shadow map to the scene by drawing the scene from the usual camera viewpoint. The tricky part of this step is the depth map test.

A main advantage of the shadow mapping method is that no knowledge or processing of the scene geometry is required. The accuracy of a shadow map, however, is limited by its resolution. Aliasing, especially when using small shadow maps, is the major disadvantage of this technique. For real-time shadows, shadow mapping is less accurate than shadow volume, but the shadow mapping method is much faster than shadow volume and ray tracing when dealing with complex scenes.

Shadow generation for objects represented by discrete voxels has also been studied [16, 17, 20, 21]. For example, a shadow determination accelerator for ray tracing, built on top of a uniform voxel traversal grid structure, is presented in [16]. An efficient shadow detection algorithm for ray tracing is proposed in [17]. The algorithm can be used for direct voxel rendering as well. A hardware implementation of shadow generation for voxels is reported in [21], which presents a novel approach to use graphics hardware to dynamically calculate a voxel-based representation of a scene.

However, as far as we know, no method has been proposed for upgrading shadow mapping method using knowledge of the scene geometry. This is probably because it would slow down the shadow generation process significantly. Voxelization based shadow generation methods [16, 17, 20, 21] do not use any information on the scene geometry either. But this is because the geometry information on voxels is unknown. In this paper, we present a method which is similar to shadow mapping but uses voxels in determining shadows. The major difference is that the scene geometry is utilized in the process of shadow determination. As a result, better quality of shadows can be generated for an even low voxelization resolution than shadow mapping. Because normal resolution ($256 \times 256 \times 256$) would lead to acceptable quality of shadows, our method can deal with complex scenes interactively.

3 Voxelization based on Recursive Parameter Space Subdivision

Given a free-form object represented by a CCSS and a *cubic frame buffer* of resolution $M_1 \times M_2 \times M_3$, the goal is to convert the CCSS represented free-form object (a continuous geometric representation) into a set of voxels that best approximates the geometry of the object [12] . We assume each face of the representation's control mesh is a quadrilateral and each face has at most one *extra-ordinary vertex*. If this is not the case, simply perform one Catmull-Clark subdivision on the control mesh of the CCSS [1].

We first consider voxelization of a single patch of the CCSS representation. Given a patch $\mathbf{S}(u, v)$ defined on $[u_1, u_2] \times [v_1, v_2]$, we voxelize it by recursively subdividing its parameter space until each subpatch is small enough (hence, flat enough) so that voxelization of that subpatch can be done simply by voxelizing its four corners [7]. It is easy to see that if the voxels corresponding to the four corners of a subpatch are not N-adjacent ($N \in \{6, 18, 26\}$) to each other, then there exist holes between them [10, 11, 12]. In this case, the subpatch is considered not small enough yet. A *midpoint subdivision* is performed on the parameter space to get four smaller subpatches and repeat the testing process on each of the subpatches. This

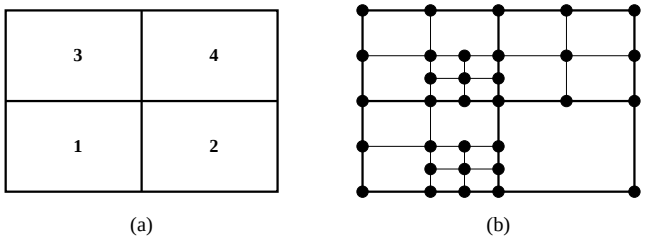


Figure 2: Basic idea of parameter space based recursive voxelization.

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

The above process guarantees a shared boundary (vertex) of adjacent subpatches will be voxelized to the same voxels (voxel). This is true for adjacent patches as well. Hence, voxelization of the entire CCSS representation can be performed on a patch based approach. To make the process of writing voxels into the cubic frame buffer simpler, the control mesh of the CCSS representation is normalized to be of dimension $[0, M_1-1] \times [0, M_2-1] \times [0, M_3-1]$ first. Patches of the CCSS representation are then voxelized one at a time, and the resulting voxels are written into corresponding entries of the cubic frame buffer. Result of this voxelization process satisfies the criteria of *separability*, *accuracy* and *minimality* with respect to the given N-adjacency connectivity requirement ($N \in \{6, 18, 26\}$) [10, 11, 12].

4 Shadow Determination

The process of determining if a voxel is in shadow is done through a projection process. Different from the Z-buffer based method, where the scene is first rendered with respect to the light to obtain a shadow map and then rendered with respect to the view point, our method projects the voxels onto a projection plane only once. Hence, it requires one traversal of the voxels only. Note that after the projection, for each entry of the projection plane, the voxel that is closest to the light source is in light. But the voxels with bigger distance from the light source are not necessarily in shadow. They could only be partially in shadow or even not in shadow at all, such as points A and B in Figure 1. So the main task here is to do shadow testing for those voxels with bigger distance from the light source. Voxels of this type will be called *uncertain voxels*.

In the projection process, to ensure a relatively good distribution of the projected voxels, the projection plane should be as much perpendicular to the light beams shot from the light to the scene as possible. The resolution of the projection plane should be the same as the resolution of the voxelization process. This is because a lower resolution of the projection plane would cause too many voxels to be projected onto the same entries of the projection plane, while a higher resolution could lead to unfilled entries between adjacent voxels. None of these situations is desired: the first situation would leave too many *uncertain voxels* to be tested and the second situation would generate illegal holes (gaps) in the final shadows. In the following, two approaches are presented for shadow testing of uncertain voxels. The testing process uses geometric and topological information inherited from the voxelization process [12].

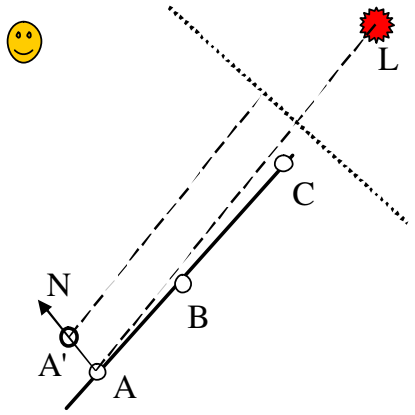


Figure 3: Offset based shadow determination

4.1 Offset-based Shadow Determination

In this approach, only those voxels that are closest to the light are stored during the projection process. The basic idea of this approach is to use coherence property to determine if a voxel is in shadow. If a voxel is blocked by another voxel, we move the voxel along its

normal direction outwards slightly and test if it is still blocked or not in the new position. For example, in Figure 3, voxel A is moved to a new location A' and then repeat the test. The new location of the voxel should be far enough to ensure this movement creates a different voxel, but not too far away to exceed the N-adjacency range ($N \in \{6, 18, 26\}$). A reasonable choice for the distance is $\sqrt{2} \cdot r$, where r is the size of a voxel. If the voxel in question is called A and the voxel at the new location is called A', then A is considered to be in shadow if A' is in shadow. Otherwise, A is considered to be in light. In order to avoid A' being projected onto the same entry as A, a different type of projection has to be used. In our implementation, we project voxel A' alone the line which starts from A' and is parallel to the beam emitted from the light to A (see Figure 3). In this way, voxel A' and A will be projected to different entries on the projection plane. After the projection, clear decision can be made if voxel A is in shadow according to our assumption. Of course, if $N \cdot (L - A) \leq 0$, then A is definitely in shadow, and the above process is not needed at all.

4.2 Topology-based Shadow Determination

In this approach, when a voxel, say A, is projected, the subpatch ID(s) that A belongs to in the surface is also stored in the corresponding entry of the projection plane. The subpatch ID of a voxel is explicitly known when we perform the voxelization process. Hence the projection process can be done simultaneously with the voxelization process. Because there are possibly more than one voxel falling onto the same entry of the projection plane, a list is needed for each entry of the projection plane so that IDs of the voxels that fall onto the same entry are linked and sorted by the distance from the voxel to the light. Therefore, after the projection process, we will know which voxel is possibly blocked by which subpatch(es). So it is straightforward to determine if a voxel A is in shadow by simply testing if the ray from A to the light intersects any subpatches linked to the same entry of the projection plane. Because the subpatches generated in the voxelization process are very small and are replaced with quadrilaterals in the rendering process, the intersection testing process becomes testing of a ray with two triangles. There are many methods for this testing process. The following approach is followed here [22] which only uses dot products in the testing process.

Given a ray R emitted from P_0 to P_1, and a triangle T with vertices V_0, V_1 and V_2. Suppose the intersection point of R and the plane of T is P_I. Note that P_I could be inside of T or outside of T. P_I is very easy to find and if P_I does not exist, R does not intersect T either. Hence if P_I exists and if $s >= 0$, $t >= 0$, and $s + t <= 1$, then R intersects with T, where

$$s = \frac{(u \cdot v)(w \cdot v) - (v \cdot v)(w \cdot u)}{(u \cdot v)^2 - (u \cdot u)(v \cdot v)}$$
$$t = \frac{(u \cdot v)(w \cdot u) - (u \cdot u)(w \cdot v)}{(u \cdot v)^2 - (u \cdot u)(v \cdot v)}$$

The definitions of u, v and w are as follows.

$$u = V_1 - V_0$$
$$v = V_2 - V_0$$
$$w = P_I - V_0$$

With 5 distinct dot products and no cross product at all, the computation is very efficient.

For a given voxel, if the ray intersects any subpatch in front of it, the voxel is in shadow. Therefore, even if there is a long list of subpatch IDs that might block this voxel, the computation actually is very fast for a voxel in shadow. This is because the computation stops once an intersection has been detected and this usually occurs on the first subpatch in font of it. Hence for a voxel in shadow, only one intersection computation is needed. If there is no subpatch being in front of a voxel, this voxel is obviously in light. For other voxels, if we go through all the subpatches in its linked list of blocking subpatches, and still are not able to find an intersection, then they should be lit. Fortunately, such voxels are always not many in an ordinary scene and usually linked lists of blocking subpatches are not long either (in our tests, they are less that 10). Hence, the overall testing does not require a lot of computation and much better quality of shadow can obtained.

5 Scene Rendering

Once shadow determination for each voxel is done, shadow generation is simply a process of rendering all voxels such that shadowed voxels are only lit with ambient light. Different from previous volumetric rendering techniques [14, 15] which render the voxels directly, our rendering process does not render the voxels, but the subpatches whose vertices are the voxels to be rendered [12]. Note that connectivity of voxels is known. Hence the resulting rendering is smooth and seamless. The rendering process is simply another process of voxelization of the surface. But this time we do not write any values into the cubic frame buffer. Instead, when we reach subpatches whose four corners are mapped to adjacent voxels, the subpatches are tessellated and sent to the rendering pipeline. However, before sending the tessellation result to the rendering pipeline, each subpatch has to be checked to see

if the corresponding voxels of its four corners are in shadow. If the voxels are all in shadow, the subpatch is only lit with ambient light. If n ($n < 4$) voxels are in shadow, then the subpatch will be lit with ambient light and $\frac{1}{2^n}$ reflection intensity, but no specular highlight. This approach provides a smooth transition between the shadowed area and the lit area.

The overall process seems simple. But the subprocess of checking if a voxel is in shadow is actually quite tricky. This particular subprocess affects the overall performance significantly. In the following two subsections we will discuss two methods to find out the lighting property of each voxel in the process of rendering.

5.1 Shadow Generation without Cubic Frame Buffer

As we can see from the above description, a cubic frame buffer is not a must in our shadow generation method. This is because we can project voxels onto the projection plane directly during the voxelization process, and consequently avoid the process of writing voxels into the cubic frame buffer. Although the projection of voxels only needs to be done once, without a cubic frame buffer, the result of shadow determination cannot be kept and has to be re-calculated each time the scene needs to be shown. Therefore this method is good only for static scenes, such as some CAD applications and scientific simulations, where one image is all that is needed. A major advantage of using this method is that the resolution of the voxelization process can be very high because we do not need a mass of memory to hold the voxels. Consequently, very high quality rendering results, with shadows, can be obtained. Of course, this is at this cost of much longer rendering time.

5.2 Shadow Generation with Cubic Frame Buffer

Without a cubic frame buffer, shadow determination has to be performed again each time the scene needs to be shown. An alternative is to have the result of shadow determination saved in a cubic frame buffer. One cubic frame buffer is enough for the entire scene, no matter how many subdivision surfaces are involved. With this approach, each voxel is either with a 0 (empty) a 1 (lit) or a 2 (shadowed) in the shadow determination process. Now we have two sets of data about the scene. One is a set of the continuous parameterized surfaces, and the other one is a set of discrete and shadow marked voxels. We can use the discrete voxel information to render the scene with some volumetric rendering methods, such as the *splatting method* [14]. We can use both of them to generate better results by rendering continuous quadrilaterals [12]. With this method, smooth and seamless rendering can be obtained. Note that connectivity of voxels is known in the voxelization process. Hence the rendering process is simply another process of voxelization of the continuous surface. But shadow determination does not need to be re-done because it is saved in the cubic frame buffer. When a subpatch is tessellated, the shadow and light properties of each voxel corresponding to each corner vertex of the quadrilateral are fetched and sent to the rendering pipeline as well.

Note with a cubic frame buffer, there is no need to re-generate the shadows when only the position of eye changes (this is most the cases seen in game designs or simulations where lights usually do not move). In addition, shadow generation with a cubic frame buffer is much faster than without one, even though it requires a lot of memory for the cubic frame buffer, especially for high voxelization resolution. Fortunately, for most of the interactive scenes, like some games, high quality shadows usually are not necessary, a relatively low voxelization resolution is acceptable most of the time. 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 would care more about the efficiency of the algorithm.

5.3 Crack Elimination

Another issue with the rendering process is *crack elimination* [8]. 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 4, 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.

Fortunately cracks can be eliminated simply by replacing each boundary of a patch or a subpatch with the one that contains all the evaluated points for that boundary. For example, in Figure 4, all the dot-

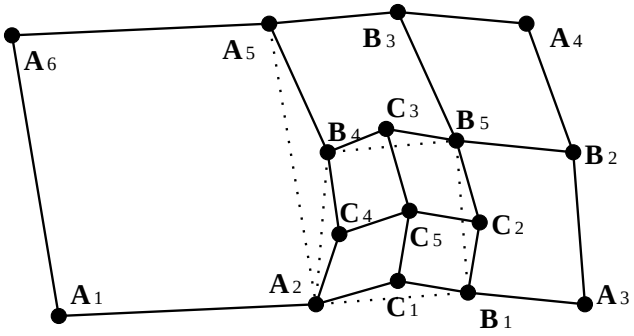


Figure 4: Crack elimination.

ted 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, the polygon $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$ in Figure 4, 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, through a simple zigzag technique, triangulation of those polygons is actually a simple and very fast process. It should be pointed out that the above polyline replacement strategy could cause small inaccuracy in the rendering process. But, the error can be precisely estimated [8]. Hence accurate rendering can still be obtained.

6 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 5. Resolution of global voxelization is $512 \times 512 \times 512$ for Figures 5(b), 5(c), 5(e), 5(f) and 5(h), and $1024 \times 1024 \times 1024$ for Figure 5(i). The rest cased in Figure 5 are voxelized with resolution $256 \times 256 \times 256$. The ground of each scene, either planar or curved, is also represented with a subdivision surface and is involved in the voxelization and shadow determination process as well.

From Figure 5 one can easily see that the quality of a result is determined by the resolution of the voxelization process. For example, Figure 5(i) uses the highest resolution, hence it has shadows of the highest quality. This is demonstrated by the smooth boundary of the lit and shadowed areas. However, with the same resolution, different approaches used for shadow determination could lead to different quality in shadow generation, especially in the area of self shadow. For example, Figures 5(a), 5(d) and 5(g) use the same voxelization resolution. But Figure 5(g) is generated using the topology-based shadow determination approach to determine shadow of each voxel while Figures 5(a) and 5(d) are generated using the offset-based shadow determination approach. As one can obviously tell that Figure 5(g) has a better quality than Figures 5(a) and 5(d).

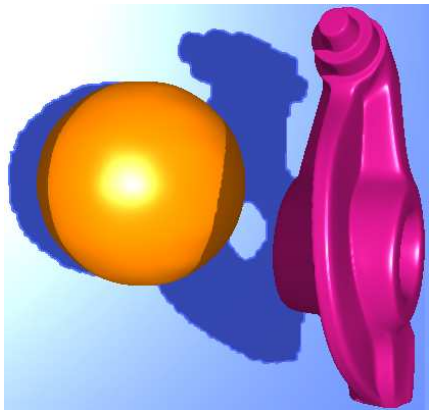
The rendering technique used for shadow generation in our experiment is cubic frame buffer based and rendering of the voxels is based on tessellation of quadrilaterals. Figure 5(i) is the only case in which cubic frame buffer is not involved in the rendering process. As we can tell there is no obvious visual difference in the generated results. However, the times used by the two approaches are very different; cubic frame buffer based approach is much faster. For example, all the test cases in Figure 5, except Figure 5(i) can be visualized interactively.

7 Summary

A shadow generation method for objects represented by CCSSs is presented. The objects are voxelized first and the resulting voxels are then projected onto a projection plane to determine which voxels are in shadow and which voxels are in light, similar to the concept of the shadow mapping method. The main difference between our method and the shadow mapping method is, in our method, uncertain cases can be resolved using geometric or topological information obtained during the voxelization phase. Therefore, better quality shadows can be generated without the need of using a high resolution in the voxelization process. A resolution of $256 \times 256 \times 256$ for the voxelization process is enough to generate shadows with acceptable quality.

Another advantage of the new approach is, since our voxelization process is performed in the parameter spaces of the CCSSs instead of the object space, the process is very fast and efficient. As a result, the overall shadow generation process is fast, efficient and robust. Therefore, our method has the capability of generating good quality shadows for complex scenes in real time without the need of using a high resolution in the voxelization process. The new method is presented for CCSSs only, but the concept works for any subdivision schemes whose limit surfaces are parameterizable.

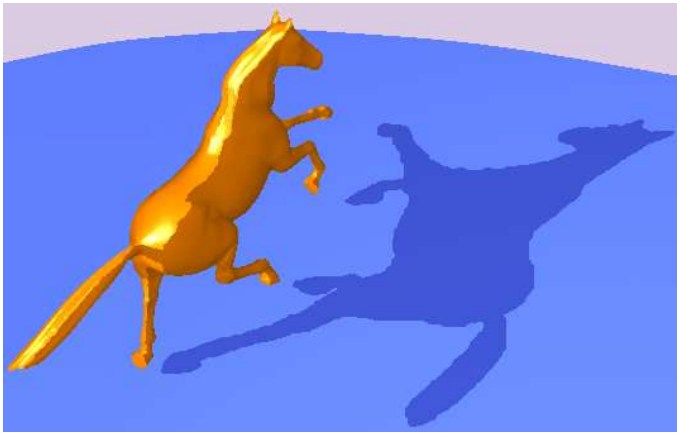
Acknowledgement. Research work reported in this paper is supported by NSF under grants DMS-0310645 and DMI-0422126. Data sets of Figures 5(d)



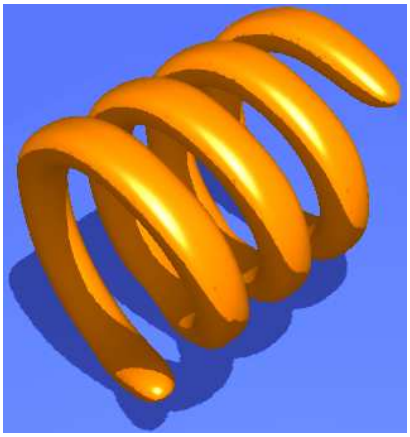
(a) Low resolution (256)

(b) Medium resolution (512)

(c) Medium resolution (512)

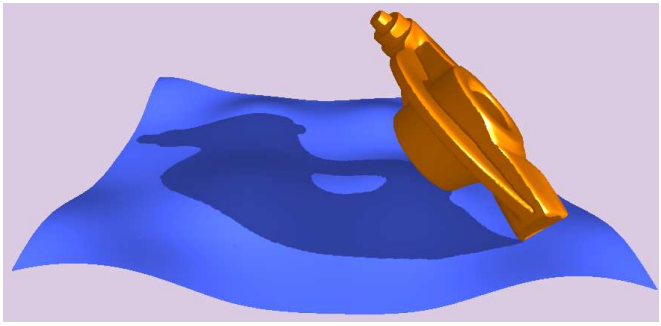


(d) Low resolution (256)

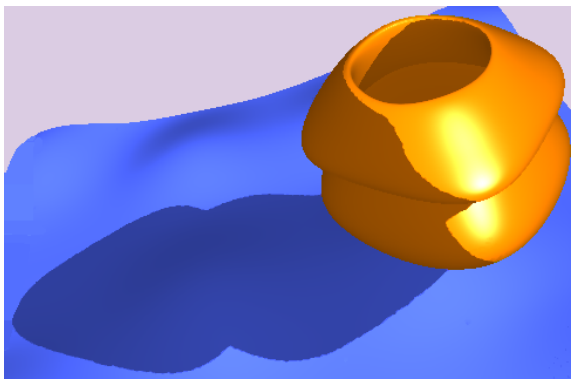


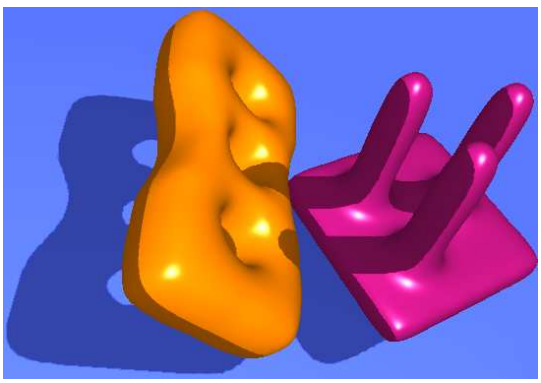
(e) Medium resolution (512)

(f) Medium resolution (512)



(g) Low resolution (256)





and 5(f) are downloaded from the web site: graphics.cs.uiuc.edu/~garland/research/quadrics.html. We also thank the anonymous reviewers for their many helpful comments.

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, 513-522.

[8] Shuhua Lai, Fuhua (Frank) Cheng, Inscribed Approximation based Adaptive Rendering of Catmull-Clark Subdivision Surfaces, *International Journal of CAD/CAM* 6,1 (2006), 1-16.

[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, *Lecture Notes in Computer Science* (GMP2006), Vol. 4077, Springer, 2006, 595-601.

[13] Shuhua Lai, Fuhua (Frank) Cheng, Robust and Error Controllable Boolean Operations on Free-form Solids Represented by Catmull-Clark Subdivision Surfaces, *Computer Aided Design & Applications*, 4, 1-4, 2007.

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

[15] Grossman, J. P., Dally, W. J. Point sample rendering. *Eurographics Rendering Workshop 1998*, 181-192.

[16] Andrew Woo and John Amanatides, Voxel occlusion testing: a shadow determination accelerator for ray tracing, in *Proceedings on Graphics interface*, 1990, 213-220.

[17] Mark W. Jones, An efficient shadow detection algorithm and the direct surface rendering volume visualisation model, in *Proc. Eurographics*, UK, 1997, 237-244.

[18] Lance Williams, Casting Curved Shadows on Curved Surfaces, in *Computer Graphics (Proc. SIGGRAPH 78)*, 12(3), August 1978, 270-274.

[19] F. Crow, Shadow Algorithms for Computer Graphics, *Computer Graphics*, 11(3), August 1977, 242-248.

[20] Andrew Pearce, A recursive shadow voxel cache for ray tracing, in *Graphics Gems II*, San Diego, 1991, 273-274.

[21] Elmar Eisemann, Xavier Décoret, Fast scene voxelization and applications, *Proc. 2006 symposium on Interactive 3D graphics and games*, 2006, 71-78.

[22] Didier Badouel, An Efficient Ray-Polygon Intersection, in *Graphics Gems*, 1990.

[23] Andrew Woo, Pierre Poulin and Alain Fournier, A Survey of Shadow Algorithms, *IEEE Computer Graphics and Applications*, Vol. 10(6), 1990, 13-32.