Geometry-shader-based real-time voxelization and applications

Article *in* The Visual Computer · March 2014

DOI: 10.1007/s00371-013-0858-5

6 authors, including:

Yu-Chi Lai
National Taiwan University of Science and Te…
18 PUBLICATIONS **94** CITATIONS
SEE PROFILE

Kai-Lung Hua
National Taiwan University of Science and Te…
55 PUBLICATIONS **63** CITATIONS
SEE PROFILE

Yuzhen Niu
Fuzhou University
34 PUBLICATIONS **214** CITATIONS
SEE PROFILE

Feng Liu
Carnegie Mellon University
222 PUBLICATIONS **3,097** CITATIONS
SEE PROFILE

Vis Comput
DOI 10.1007/s00371-013-0858-5

ORIGINAL ARTICLE

Geometry-shader-based real-time voxelization and applications

Hsu-Huai Chang · Yu-Chi Lai · Chin-Yuan Yao · Kai-Lung Hua · Yuzhen Niu · Feng Liu

Abstract

This work proposes a new voxelization algorithm based on newly available GPU functionalities and designs several real-time applications to render complex lighting effects with the voxelization result. The voxelization algorithm can efficiently transform a highly complex scene in a surface-boundary representation into a set of voxels in *one GPU pass* using the geometry shader. Newly available 3D textures are used to directly record the surficial and volumetric properties of objects such as opaqueness, refraction, and transmittance. In the first, the usage of 3D textures can remove those strenuous efforts required to modify the encoding and decoding scheme when adjusting the voxel resolution. Second, surficial and volumetric properties recorded in 3D textures can be used to interactively compute and render more realistic lighting effects including the shadow of objects with complex occlusion and the refraction and transmittance of transparent objects. The shadow can be rendered with an absorption coefficient which is computed according to the number of surfaces drawing in each voxel during voxelization and used to compute the amount of light passing through partially occluded complex objects. The surface normal, transmittance coefficient and refraction index recorded in each voxel can be used to simulate the refraction and transmittance lighting effects of transparent objects using our multiple-surfaced refraction algorithm. Finally, the results demonstrate that our algorithm can transform a dynamic scene into a set of voxels and render complex lighting effects in real time without any pre-processing.

Keywords Voxelization · Hardware voxelization · Transparent shadow · Volume ray-tracing

1 Introduction

Realism is important for human perception but photorealistic rendering is very time consuming and therefore, there are a large number of research to approximate these realistic lighting effects for interactive applications. Among these important lighting effects, shadow is important for human perception because shadow gives the sense of existence. Traditional shadow map [36] is the simplest interactive shadow algorithm but it cannot handle transparent objects and scenes with complex occlusion. In general, the refraction and transmittance effects of transparent objects are beautiful and intriguing. Although many research look for hardware-accelerated approximation [1, 12, 29, 38], there is still room for improvement. Thus, this paper proposes several techniques based on a volumetric representation to render the shadow of complex objects and refraction and transmittance of transparent objects for interactive applications.

H.-H. Chang · Y.-C. Lai (✉) · C.-Y. Yao · K.-L. Hua
National Taiwan University of Science and Technology, Taipei, ROC
e-mail: cheeryuchi@gmail.com

H.-H. Chang
e-mail: td458193@hotmail.com

C.-Y. Yao
e-mail: cyuan.yao@gmail.com

K.-L. Hua
e-mail: hua@mail.ntust.edu.tw

Y. Niu
College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China
e-mail: yuzhen@gmail.com

F. Liu
Portland State University, Portland, USA
e-mail: fliu@cecs.pdx.edu

Published online: 05 July 2013

Generally speaking, volumetric data stores properties of an object in a set of regular 3D grids or a non-uniform-sized volume structures such as a kd-tree [41] and a BVH tree [22]. A voxelization algorithm is used to transform a surface-boundary representation to a volumetric representation for volumetric applications. Generally, non-uniform-sized representations can be more memory efficient and robust to aliasing artifacts but the construction is more complex and because the storage cannot be directly mapped to 3D volume textures, the usage requires specific mapping techniques to efficiently extract recording information. All these make and limit the usage of these non-uniform-sized volumetric structures in the GPU-based applications. To the contrary, uniform-sized grids are regular, simple, fast and easy to extract information and propagate rays through the voxel space for real-time applications. Therefore, this work chooses uniform-sized grids for our volumetric representation. In order to achieve interactivity, real-time slicing-based GPU voxelization algorithms [3, 6, 10, 20, 24] are proposed to slice models into a set of voxels. However, their algorithms must run with multiple GPU passes which are equal to the number of slices and each pass must render all primitives. Therefore, encoding slice-based algorithms [6, 7, 11, 20] are proposed to reduce the number of passes by examining the intersection of each primitive with each voxel grid only once with a special encoding mechanism. Unfortunately there are several limitations including the usage of triangles as the represented primitives and the strenuous process of changing encoding and decoding mechanism when applications change their voxel resolution. As a result, this work proposes a new GPU-based voxelization algorithm to overcome these limitations using the geometry shader to slice the geometric models with the clipping plane algorithm [10] in *one GPU pass*. The usage of the geometry shader to voxelize the models relieves the need of multiple passes and reduces the times of primitive rendering from the number of slices to the depth range of a primitive. Furthermore, the adjustable 3D volume textures are used to store the slicing result with other surficial and volumetric information. Since the size of 3D textures can be easily adjusted according to the need of applications and the limitation of graphics hardware, this can ease the burden of changing encoding and decoding mechanism when adjusting the voxel resolution for a general encoding voxelization method. Results show that our algorithm is efficient enough to render the desired lighting effects for interactive applications.

Eisemann et al. [7] use the voxelization result to estimate the traversal length of light passing through transparent and partially occluded objects and then convolves the length with a homogeneous scene-based absorption coefficient for rendering the shadow of complex objects. The assumption of a single coefficient for the entire scene limits the ability to simulate different degrees of occlusion in complex partially occluded objects which are commonly seen in daily life. Therefore, our algorithm overcomes this limitation by computing the absorption coefficient according to the type of the object and the density of small geometries in each voxel. Then the shadow of the object can be rendered by attenuating the light using these absorption coefficients along the traversal path. The shadow generated is closer to our perception with negligible extra cost.

Two-surfaced refraction methods [29, 38] use image-space algorithms to approximate the refraction of a transparent object and they can provide realistic results in a high interactive frame rate. However, there are still limitations in image-space methods including the requirement of an extra image map per object for estimating the light traversal distance, the assumption of a non-self-occluded object and the disability to simulate multiple-refraction and total-reflection effects. This work proposes to use the volumetric representation to overcome these limitations. GPU-based voxelization first slices the entire scene into a set of unit-sized voxels stored in 3D volume textures. Our algorithm takes advantage of the flexibility and adjustability of 3D volume textures to compute and store the surficial and volumetric information including the normal, refraction index and transmittance coefficient for better approximation. And our multiple-surfaced refraction algorithm first traces a view ray from the camera into the scene. Then, when intersecting with the surface-boundary voxels, the normal and refraction index is used to compute the new propagation direction. The view ray is marched through the set of voxels to estimate the transmittance attenuation. The process continues until the ray shoots out of the scene and the color indexed by the ray direction and the attenuation accumulated along the path are used to compute the color of the ray. This voxel-based refraction algorithm gives applications the abilities to represent the entire scene with a single representation without the need of extra support data and simulate the multiple-refraction and total-reflection effects of possibly self-occluded transparent objects. The results show that our algorithm performs better than image-space methods on rendering a scene with complex deformable and transparent objects.

The main contributions of this paper are as follows: first, an efficient slicing-based voxelization is proposed based on the newly available geometry shader to reduce the number of GPU passes down to one and the number of primitive rendering from the number of slices down to the depth range of a primitive; second, newly available 3D textures are used to ease the burden of modifying the encoding and decoding mechanism when changing the voxel resolution and to record the surficial and volumetric information for better estimation of several complex lighting effects; third, inhomogeneous absorption coefficients are computed based on degrees of occlusion and used to create a more realistic shadow for objects with complex occlusion; finally, a

multiple-surfaced refraction algorithm is designed to overcome the limitations of image-space refraction algorithms in order to render the refraction and transmittance effects of transparent objects without the need of an extra image map per object and with the abilities to simulate multiple-refraction and total-reflection effects.

2 Related works

This section will review previous works in the topics of voxelization and rendering complex lighting effects including the transmittance, refraction, and shadow of complex objects.

2.1 Voxelization

Generally, voxelization strategies can be classified based on how to describe the existence of a model as surface voxelization [16, 17, 33] which uses the existence of a voxel to describe the boundary and solid voxelization [8, 13, 32] which both describes the existence of the boundary and interior of the entire model. Another common classification is based on how the existence of a voxel is represented and can be described as binary [6, 7, 10] and non-binary [10, 14, 16, 31, 32, 34, 35] voxelization approaches. The latter can be further divided into filtered voxelization [32, 35], multi-valued voxelization [10, 14], object identification voxelization [16], distance transform voxelization [31, 34] and classification voxelization [19]. A major drawback of these algorithms is efficiency and they are mainly used in a pre-processing step by assuming a static scene.

Slicing-based GPU voxelization algorithms [3, 6, 10, 20, 24] set appropriate clipping planes to decompose the model into slices to form the volumetric representation. These slicing-based algorithms suffer a serious limitation in computational efficiency because they require multiple GPU passes which are equal to the number of slices and each pass must render all primitives. Encoding slice-based algorithms [6, 7, 11, 20, 40] are proposed to reduce the number of passes by examining the intersection of each primitive with each voxel grid only once with a special encoding mechanism. Although the encoding algorithm can be really efficient, unfortunately the algorithm is limited to use triangles as the represented primitive and the performance is influenced by a dynamic update of the sorted triangles required for voxelizing deformable objects. Additionally these grid encoding methods have other limitations: first, the encoding method must determine the resolution of voxelization and design the encoding mechanism accordingly and this makes resolution adjustment highly strenuous; second, the surficial and volumetric properties of a model are hard to directly encode into the representation; finally, the number of GPU passes for a high-resolution representation still has chances to be more than one. As a result, this work uses the newly available geometry shader to slice the geometric models in *one GPU pass* and reduce the number of primitive rendering from the number of slices to the depth range of a primitive. In addition, adjustable 3D volume textures are used to store the slicing result with other surficial and volumetric information and ease the burden of changing encoding and decoding mechanism.

2.2 Transparent shadow

A standard shadow map [36] is a multi-pass method. First, a pass per light computes the depth of the closest occluder from the light. Second, when viewing from the camera, the depths for all lights of the first intersection point from the camera along the view is compared against the depths in the maps to determine whether a pixel is in shadow. However, the assumption of opaque objects puts a limit on applying the shadow map to transparent objects and objects with complex partial occlusions. In other words the amount of light passing through the objects is affected by the distance of light passing and the amount of occlusion. Later, a deep shadow map [25] is proposed to store a one dimensional light intensity function along ray for each ray. This technique achieves realistic self shadowing for very complex volumetric structures like hair. Works in [18, 39] further extend to render different lighting effects based on the existing volumetric representation. These algorithms must accompany some pre-processing steps for estimating the volume or the density of the subject and may not be unsuitable for dynamic scenes. Eisemann et al. [7] use the voxelization result to estimate the passing distance for rendering transparent shadow but their method assumes either transparent objects with homogeneous material or no partial occlusion by using a fixed absorption coefficient for occluded voxels and this limits the possibility of better precision. Our algorithm can compute the absorption coefficient according to the type of the object and the density of small geometries in each voxel automatically. Then the shadow is generated according to the passing length and the accumulated absorption along the traversal path.

2.3 Refraction

Single-surfaced refraction is proposed to compute refracted or pseudo-refracted directions through visible front facing polygons. The lighting of the refracted ray is computed by using a distant environment map [23] or a perturbed texture describing nearby geometry [1, 28]. These methods do not consider the multiple refractions and absorption of light passing through transparent objects. Several researchers propose multi-sided refraction algorithms. Kay et al. [21] use “thickness” to simulate the two-sided refraction

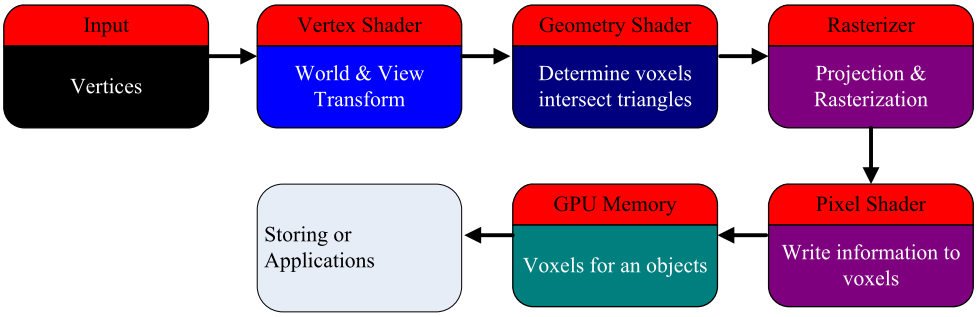


Fig. 1 This illustrates the concept of the proposed voxelization algorithm. The input is a set of vertices and topological information and the output is a set of voxels recording the properties of the scene. The output can be directly used for rendering the shadow of complex objects in Sect. 4 and the refraction and transmittance effects of transparent objects in Sect. 5

but the assumption breaks down when objects have variations in thickness. An interactive multi-pass hardware algorithm is proposed by Diefenbach et al. [5] to simulate the refraction through planar objects but the usage is limited by only allowing planar objects. Ohbuchi et al. [27] suggest a vertex tracing preprocess to approximate interactive refractions on a prototype multimedia processor. The interpolation from vertex tracing has issues in correctness and the requirement of pre-processing limits the usage in dynamic scenes. Guy et al. [12] propose an analytic method to compute refracted vertices and update the resulting facet tree every frame for interactively simulating the refraction of simple convex gemstones. However, this analytic method may not scale well to real world objects which are generally complex. Heidrich et al. [15] use a light field representation to trade higher memory consumption for interactivity. Wyman et al. [38] propose a two-surfaced refraction method to simulate the refraction of light passing through transparent objects. The second refraction is an approximation to the real path. Oliveira et al. [29] propose an improvement to remove the pre-processing step. However, there are still limitations in image-space refraction methods: first is the need of a penetration depth map per object, second is the requirement of objects with non-self-occlusion, third is the second refraction direction which may not be correct and finally is the disability to simulate multiple-refraction and total-reflection effects. Additionally, depth-peeling algorithms [2, 4, 9, 26] can be used to fast render non-refractive transparency by transforming surface-boundary geometries into a set of layer depth images (LDIs) [30]. Then LDIs are used to estimate the non-refractive transmittance effects. The transformation can reduce the number of GPU passes to the depth complexity of the scene. However, when using the depth-peeling techniques for estimation of refraction and transmittance effects, the techniques have the following limitations: first, because the depths in each layer do not maintain the spatial coherence in the direction of depth, extra operations must be added for ray marching or propagation; second, when the material is not homogeneous, the depth peeling may have issues in peeling objects; finally, the depth complexity can be large in objects with complex occlusion such as trees and hairs and the number of passes is large. Therefore, the slicing-based hardware voxelization is chosen to overcome the limitations of image-space and depth-peeling refraction algorithms. Our algorithm first computes and stores the surfacial and volumetric information including the normal, refraction index and transmittance coefficient during voxelization and then a multiple-surfaced refraction method is used to simulate the multiple-refraction and total-reflection effects of possibly self-occluded transparent objects using a single representation for the entire scene.

3 Geometry-shader-based voxelization

The newly available geometry shader is used to slice a surface-boundary scene into a set of voxels. Before developing our voxelization algorithm, we must decide how to store the volumetric representation of a model. A uniform-sized voxel structure is schematically similar to a 3D volume texture. Thus, using a texel in a 3D volume texture is the simplest way to store the volumetric data in a voxel. The ability to adjust the memory size of a texel gives our algorithm the flexibility of computing and storing extra surficial and volumetric information such as transmittance coefficients, normals, and colors for generating more realistic lighting effects as described in Sects. 4 and 5. The computation of voxelization is conducted with the steps shown in Fig. 1: First, the vertices of a primitive is sent to the vertex shader and their positions in the camera coordinate are computed. Second, the geometry shader determines which slices along the z-axis direction have the chance to intersect the rasterized primitive and duplicate the primitive for rasterization according to the depth range of the primitive as described in Fig. 2. The geometry shader is perfect for this task because it handles the assembly and construction of the triangle and issues the rasterization command to the pixel shader. Finally,

the geometry shader sets proper clipping planes and issues the rasterization of the triangle to the pixel shader for filling in boundary voxels. At the same time the pixel shader also computes and stores extra surficial voxel information for the rasterized pixel.

The pseudo code of slicing a triangle is shown in Fig. 2. Generally, a surface-boundary primitive is stored as n vertices with their position, normal, and other information and this paper chooses triangles as represented primitives. When vertices of a triangle are queued into the graphics pipeline, the position of a vertex is first transformed into the camera coordinate. Our voxelization algorithm computes which slices from the 3D volume texture have the chance to intersect the triangle. The range of slices which possibly intersect the triangle can be calculated with the depth of all three vertices using Step 3, 4, and 5 listed in Fig. 2. For all possible slices, our algorithm sets up the far and near clipping planes according to the index of the destined slice. The clipping planes are used to correctly determine those boundary voxels in this destined layer of the 3D volume texture in the rasterization process as shown in Fig. 3. At the end, the geometry shader duplicates the triangle and issues the rasterization of the duplicated triangle to the pixel shader for filling in boundary voxels as described in Step 9 and 10 in Fig. 2 and computing and storing extra surficial voxel information. Figure 3 shows a simple example of the process.

Slice a triangle to fill in boundary voxels

```
1   For each triangle, Tri
2     z0 = Z(Tri.V0), z1 = Z(Tri.V1), z2 = Z(Tri.V2)
3     max_slice = max(z0, z1, z2)/thickness
4     min_slice = min(z0, z1, z2)/thickness
5     For i = min_slice to max_slice
6       Plane_near = i * thickness
7       Plane_far = Plane_near + thickness
8       Set Plane_near and Plane_far to projection matrix
9       If (Intersect(Tri))
10          Rasterize(Tri) into the slice
```

Fig. 2 This is the pseudo code for computing the boundary voxels of a triangle, *Tri*. *V* denotes a vertex of a triangle, *thickness* is the voxel size which is a user-specified value, **Z**() is a function to extract the depth value of a vertex after transforming the position of the vertex into the camera coordinate, **max**() / **min**() computes the maximum/minimum value among the set of input values, **Intersect**() is a function to test whether the triangle is valid after being culled by the clipping planes and **Rasterize**() duplicates the triangle and issues the rasterization to the pixel shader to fill in boundary voxels and compute and store extra surficial voxel information. The HLSL shader code is in the supplemental material of this paper

4 Transparent shadow map

Shadows give the sense of existence and is an important cue for human perception. Traditional shadow map is a simple method to render shadows when all objects in the scene are opaque. When there are transparent objects in the scene, the partial occlusion of a light ray passing through these transparent objects makes the situation more complex. The distance of a light ray passing through the object and the absorption along the traversal affect the amount of absorption. Eisemann et al. [7] use the voxelization result to estimate the passing distance for rendering transparent shadows but their method did not take different degrees of occlusion and material absorption into account. Our algorithm makes an improvement in computing shadows by attenuating the light along the traversal path by accumulating the absorption with varying degrees of occlusion and material absorption as an example shown in Fig. 4 with the following steps:

1. The voxelization camera is set at the position of a light source and aligned with the light direction.

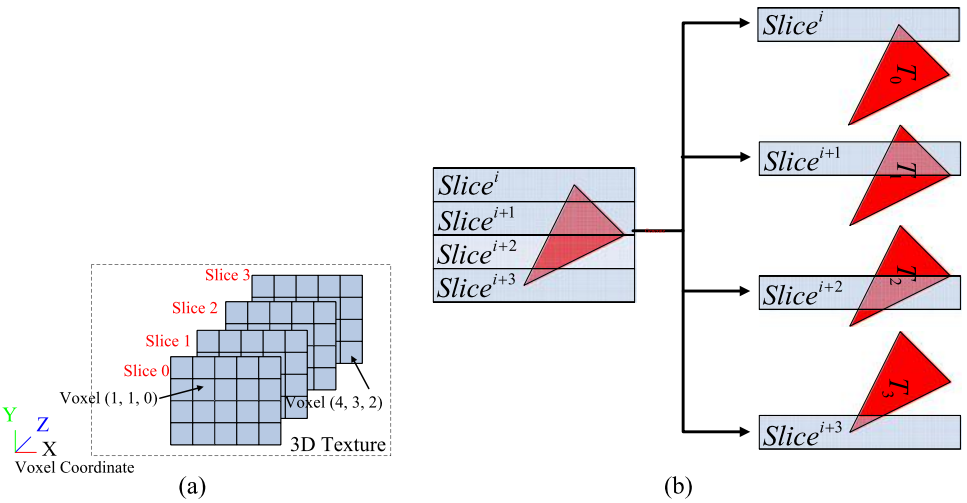


Fig. 3 (**a**) This shows a simple scheme of index for voxels in a 3D volume texture of $5 \times 5 \times 4$. Each pixel in the 3D volume texture corresponds to a voxel in the volumetric data such as $(1, 1, 0)$ and $(4, 3, 2)$. The center of each voxel is easily computed by single scale transformation of the texture coordinate. (**b**) The *triangle* is intersected with four slices. The clipping plane algorithm has to run four times and each sets the clipping planes for the corresponding slice. The *triangle* is also duplicated four times for the corresponding test in the clipping plane algorithm

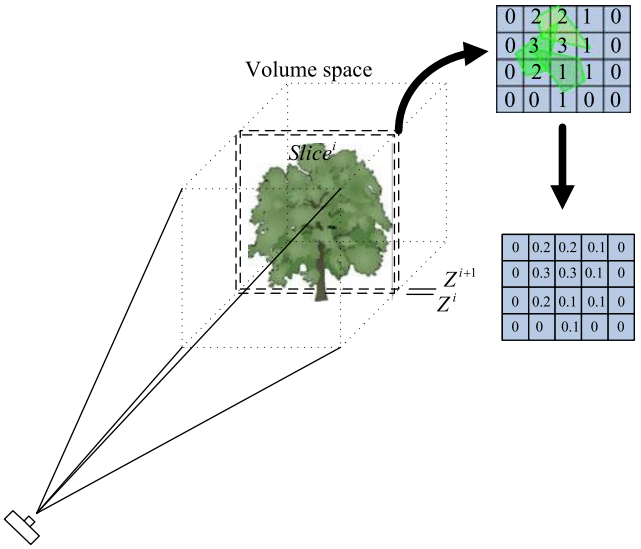


Fig. 4 The camera is aligned with the scene and the tree is drawn and sliced into the voxels. During slicing, each voxel will count the number of drawings as one example slice in the top right. Then the count will be transformed to the absorption coefficient in the bottom right

2. Our algorithm voxelizes the scene and computes and stores the absorption coefficient of each voxel. The absorption coefficient is computed by accumulating the number of writing in each voxel and then this number is multiplied by a user-specified constant to get the absorption coefficient because the number of drawing reflects the degree of partial occlusion in the voxel.
3. During the rendering process, the voxel position, (x, y, s), of the first intersection point from the view is computed.
4. The amount of occlusion for light rays can be computed using the following equation:

$$\sum_{i=0}^{s} \alpha(x, y, i) E(x, y, i) \tag{1}$$

where $\alpha()$ describes the light absorption in this voxel and $E()$ is an occupation flag for which 1 represents that the voxel is occupied by some object.

5 Refraction and transmittance

Refraction is the change in the propagation direction of a light ray when it transports from one medium to another and the change in the light propagation direction can be described by Snell's Law. Additionally, transmittance describes the phenomenon that the amount of energy decreases when a light ray passes through a transparent object. But single-surfaced refraction algorithms are not enough to describe the light transport through a transparent object because generally a light ray enters and exits an object in a pair and it is a multiple-refraction phenomenon.

Compute refraction

```
1  For each pixel
2     T1 = Refract(I, Voxel(P1).N)
3     d = FindThickness(Voxel(P1), I)
4     P2 = P1 + d * T1
5     N2 = Voxel(P2).N
7     T2 = Refract(T1, N2)
```

Fig. 5 **I** is the incident light direction, **Voxel**() is a function to locate the voxel with the location, **N** is the surface normal stored in the voxel, P_1 is the position of the rendering point, d is the transmittance distance, P_2 is the second refraction position, $\mathbf{T}_1$ and $\mathbf{T}_2$ are the first and second refraction directions, **FindThickness**() estimates the thickness with position and ray direction and **Refract**() calculate the refraction direction according to Snell's Law. The HLSL shader code is in the supplemental material of this paper

Furthermore, the refraction path can be used to better estimate the transmittance effect inside objects. Therefore, this section will describe methods to simulate multiple-refraction and transmittance effects inside objects with a complex compound of material based on our voxelization results.

5.1 Two-surfaced refraction

Wyman et al. [37] propose that multiple-refraction effects may be simplified to two-refraction effects: one happens when light enters the object and the other happens when light exits. The first refraction can use the normal of the intersection point and the incident direction to compute the first refraction direction, $\mathbf{T}_1$. If the traversal distance, d, between the first and the second refraction point can be estimated, the second refraction position can be estimated with the following equation:

$$P_2 = P_1 + d\mathbf{T}_1 \tag{2}$$

where P_1 and P_2 are the first and second refraction position, and $\mathbf{T}_1$ is the refraction direction after the first refraction. Wyman et al. [37] design an image-space method to estimate d without considering the first refraction direction. We realize that our voxelization result can find a better estimate of d and our two-surfaced refraction algorithm can estimate $\mathbf{T}_2$ in the following steps:

1. The voxelization camera is aligned with the rendering camera.
2. Our algorithm voxelizes the scene and computes and stores the surface normals and refraction indices.
3. After voxelization, $\mathbf{T}_1$ is computed with the view direction, position, surface normal and refraction index at the first scene intersection point of the view ray.
4. The voxelization result is used to estimate the transmittance distance, d, with a similar manner described in Sect. 4.

5. P_2 can be computed using Eq. (3) and projected into the voxel space to extract the surface normal, $\mathbf{N}_2$ and refraction index.
6. The second refraction direction, $\mathbf{T}_2$, can be computed with $\mathbf{T}_1$, $\mathbf{N}_2$ and refraction index.

Figure 5 lists the pseudo code for the two-surfaced refraction algorithm.

5.2 Multiple-surfaced refraction

The two-surfaced refraction method cannot simulate all transmittance lighting effects when light hits a transparent object in a scene. And it also has some limitations in the allowable models and simulating transmittance effects. Thus, a multiple-surfaced refraction algorithm is proposed to simulate the refractions and reflections inside a scene. The same voxelization process described in the two-surfaced refraction method is used. When rendering the scene, the view initiates a view ray passing through the center of a pixel. Then, the ray propagates inside the voxel space and every time when the ray hits a boundary voxel, the ray is refracted according to Snell's law. In order to properly locate the boundary voxel for refraction, the propagating distance must be set properly to prevent missing the boundary voxel during the traversal procedure and wasting efforts in extra propagation. Our implementation chooses the physical distance to propagate through a voxel as the propagation step distance. Then, the position where the next refraction event happens can be computed as follows:

$$P_i = P_{i-1} + \mathit{thickness} \times T_{\text{voxel}}(\mathbf{T}_i)/\cos\theta \tag{3}$$

where P_{i-1} is the current position, *thickness* represents the physical size of the voxel, $\mathbf{T}_{i-1}$ is the current ray propagation direction, $T_{\text{voxel}}()$ is a function to transform the ray into the voxel coordinate for locating the voxel which records the normal and transmittance information and θ is the angle between the ray and the dominant component axis of the ray. The next step computes the refracted view ray direction according to the following equation:

$$\mathbf{T}_i = \begin{cases} \mathit{ref}(\mathbf{T}_{i-1}, N_v(P_i), T_v(P_i)) & \text{if } (E(P_i)) = 1 \\ \mathbf{T}_{i-1} & \text{otherwise} \end{cases} \tag{4}$$

where $\mathit{ref}()$ computes the new ray direction according to Snell's law, $E(P_i)$ is a flag which indicates whether the voxel at P_i is a boundary voxel or not, $N_v(P_i)$ extracts the normal stored in the voxel at P_i and $T_v(P_i)$ extracts the refraction index stored in the voxel at P_i. The process continues to find the intersection and refracted ray direction until the ray shoots out of the scene.

However, when applying the multiple-surfaced refraction algorithm described in the previous paragraph, several situations may happen to induce serious artifacts into the rendering result. The one happens most frequently is multiple boundary voxels along the traversal path which is shown in Fig. 6(a). This happens because the boundary voxels may occupy a certain volume in the voxel space. Therefore, even when the ray passes through the voxel without really intersecting the surface, our algorithm may misjudge the hitting of the boundary voxel by the ray and initiate the refraction computation. This leads to unwanted artifacts. Our algorithm uses the locality of surfaces to relieve this issue. We observe that when the angle among the normals of surfaces is small, the chance of the consecutive refractions happens is small and the possibility of misjudge is high. Thus, our algorithm uses a threshold to determine whether the refraction mechanism initiates or not and this can reduce a large amount of artifact in this type.

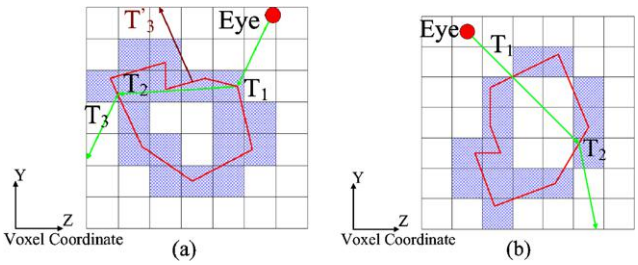


Fig. 6 (**a**) This demonstrates the problematic scheme during the ray traversal. The *green line* segments demonstrate the correct view path and the brown line segments show the possibly problematic view path. The reason of this is because $\mathbf{T}_2$ intersects with multiple boundary voxels and the closest one is not the desired one along the traversal path. (**b**) Because of the hole in the voxelization result, the first surface information is neglected and thus the refraction path cannot be currently tracing which is similar to the problematic ray marking

When tracing view rays, there is another condition as shown in Fig. 6(b) which may cause the failure of refraction computation. The missing boundary voxel problem happens because the voxelization resolution limitation induces a hole on the water-tight boundary surface and the ray may pass through the corner and edge of the boundary voxels and miss the hit when tracing the ray through the voxel space. This problem is similar to the hole problem when solid voxelizing a model. Generally a low pass filter should be able to reduce this problem. In addition, the filter technique can also reduce the multiple-intersection issue described previously. Our algorithm proposes another relief to this missing issue based on the observation that a missing issue is much harder to handle properly than a multiple-intersection issue. Thus, when slicing the scenes, our voxelization algorithm extends the clipping region of the slice to make the extent of a voxel overlap with others' to increase the chance of multiple-intersection situations and reduce the chance of missing situations. Then the angle threshold discussed in the previous paragraph can be used to get a good rendering result.

5.3 Transmittance

When light passes through a medium, the amount of energy passing through decreases and this phenomenon can be described by transmittance. The simplest method [7] to estimate transmittance uses a parameter, transparency, which depends on the traversal length of the light ray through the transparent object. Then, the transmittance is used to determine the amount of transparent blending between the object and the background. However, the method [7] is limited to an object with homogeneous material. Our voxelization result contains the surface normal, transmittance coefficient and refraction index. This allows our rendering method to compute transmittance with higher precision by both considering the length and the different transparent attenuation of the traversal voxels along the path. The traversal distance between refraction points can be estimated using $d_i = \|thickness \times T_{\text{voxel}}(\mathbf{T}_i)/\cos\theta\|$ which is a byproduct of Eq. (3). Then the transparent attenuation can be computed by the following equation:

$$transparency = \prod_{i=0}^{n} e^{-\alpha' c d_i} \tag{5}$$

where n is the total number of refraction points along the traversal path, α' represents the absorbance and c represent the density of the material [7].

6 Results

Due to the paper length and space limitation, this section will only present several representative results. Complete results rendered with our algorithm are shown in http://graphics.csie.ntust.edu.tw/pub/Voxelization/main.html.

6.1 Voxelization

All the results in this paper are rendered and measured using a computer with ATI HD5850, Intel Core 2 duo E6750 and 2 GB main memory. Our voxelization algorithm is implemented with DirectX 11 but the same program is also compatible to DirectX 10. The vertex, geometry, and pixel shaders are written using HLSL 4.0. A 3D volume texture is implemented with a 2D texture array which is a set of 2D textures with the same format in each pixel and the same resolution for each texture. The array provides the required properties to totally support the need of our algorithm and gives our algorithm more freedom in setting the format of each pixel during the implementation process. And a 32-bit RGBA floating point format is chosen to record the voxel information in our current implementation.

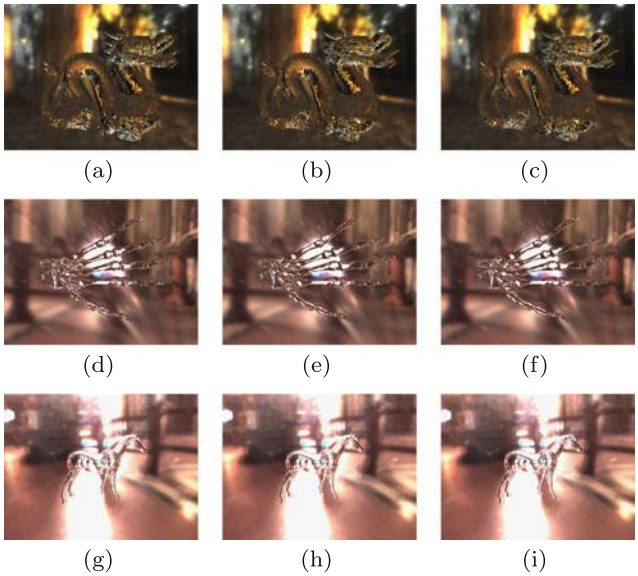


Fig. 7 The *first column* are results rendered with the multiple-surfaced refraction algorithm using the voxelization result in the resolution of $128 \times 128 \times 128$. The *second column* are results rendered with the multiple-surfaced refraction algorithm using the voxelization result in the resolution of $256 \times 256 \times 256$. The *third column* are results rendered with the multiple-surfaced refraction algorithm using the voxelization result in the resolution of $512 \times 512 \times 512$. The refraction index is set to be 1.14 for the models

Additionally, each graphics hardware device has a different limitation in the allowable voxelization resolution and the limitation depends on the available GPU memory space. For example, a resolution of $256 \times 256 \times 256$ with 32-bit information per voxel requires a memory space of 64 MB to store the data. According to the graphics card used in the test, our voxelization algorithm is tested on voxelizing different models with various triangle counts under three different resolution settings which are $128 \times 128 \times 128$, $256 \times 256 \times 256$ and $512 \times 512 \times 512$. Figure 7 shows the rendering results of simulating refraction effects using the voxelization result of these three different resolution settings. Most portion of the transparent objects is rendered the same with the three different resolutions. The statistics is listed in Table 1. In addition Table 1 also shows the time required to voxelize the same set of models under these three resolutions using the algorithm proposed by Ignacio et al. [24]. The main reason to compare against their algorithm is due to being the derivative of the slicing-based algorithm and the same usage of 3D textures to store the voxelization result. However, their algorithm processes the primitives in a model once per slice and all passes must go through all primitives once. Thus, the amount of computation increases with the increase in the voxelization resolution as shown in Table 1. Obviously, our algorithm can get much better efficiency when voxelizing models in a higher resolution.

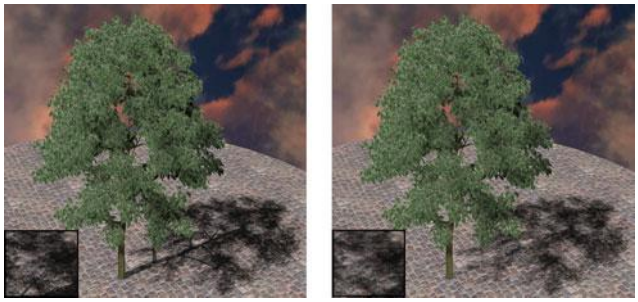


Fig. 8 Rendering the shadow of a surface-boundary tree model using ray tracing is too time consuming and thus the transparent shadow map technique is a better choice. (**a**) Is a tree rendered with different voxel absorption coefficients computed according to the material and the degree of occlusion. The trunk shadow is dark and the shadow of the leaves varies according the degree of occlusion. (**b**) Is a tree rendered with a uniform absorption coefficient for the entire tree

However, because their algorithm can process the slices in a sequential order to set up the proper stencil buffer in the voxelization process, their algorithm can get interior information with higher precision for relieving aliasing artifact when using the results.

Table 1 demonstrates that the main factor of efficiency is the triangle count of the model and the voxelization resolution. In addition our algorithm is also affected by the size of the model and the viewing angle of the voxelization process. When analyzing the contribution of these other factors, we find that the difference between the best and worst performance is roughly 10 ms. In addition to the cost factors from the voxelization process, there are other cost factors when applying the voxelization result to rendering the lighting effects proposed in Sects. 4 and 5. The main one is the amount of voxels accessed through the process. Because the rendering process has to run through a set of slices, when the number of processing slices increases, the amount of texture access also increases. It is even worse that the cost to access the texture data in the GPU memory space is much higher than the cost to access CPU memory. Thus, a general practice is to reduce the number of slices with the increase in the slice resolution to reduce the number of processed slices with an acceptable rendering quality.

6.2 Transparent shadow

A tree rendered with its transparent shadow is shown in Fig. 8. The trunk is opaque and the leaves are partially occluded and we model these partial occlusions using different absorption coefficients in each voxel. As shown in the right picture of the figure, Eisemann's method renders the shadow of the tree with a uniform absorption coefficient and thus the shadow of the trunk is not dark enough to show the solidness and the leaf shadow does not show the degree of the occlusion along the light paths. Our algorithm computes absorption coefficients according to the material and the degrees of occlusion. Thus, this gives large absorption coefficients to the trunk voxels and larger absorption coefficients to highly occluded leaf voxels and smaller absorption coefficients to lowly occluded leaf voxels. This makes the trunk shadow dark and the leaf shadow in the blow-up image in the left of the figure demonstrates different degree of darkness according to the degree of occlusion. Because our absorption coefficient takes the density of occlusion into account, the rendered shadow looks more realistic. This realism comes with almost negligible cost because the number of drawing in each pixel is the byproduct of voxelization. Additionally, the supplemental video also shows an animation of the tree scene with the movement of a light and the camera when rendering with our transparent shadow algorithm. The temporal coherence of the rendered shadow is well maintained as shown is the accompanied video.

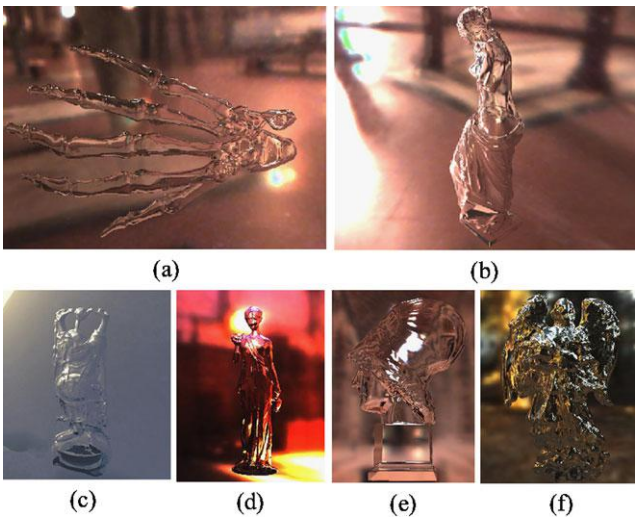


Fig. 9 These are the results rendered with the multiple-surfaced refraction with our voxelization algorithm. The refraction index is set to be 1.14 for the models used in (**a**), (**b**), (**c**) and (**d**) and 1.3 in (**e**) and (**f**)

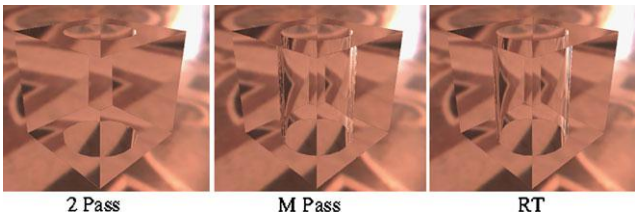


Fig. 10 The *left*, *middle*, and *right* are rendered using the two-surfaced refraction [37], multiple-surfaced refraction and ray-tracing methods

6.3 Refraction and transmittance

The multiple-surfaced refraction algorithm can simulate the multiple-refraction and total-reflection effects to generate realistic refraction effects in real time. It is generally more

Table 1 This shows the time needed to voxelize models with different triangle counts with different resolution settings using our voxelization algorithm marked with *Ours* and the voxelization algorithm proposed by Ignacio et al. [24] marked with *Grid*. The performance is measure in ms

Name	Alg.	# Tris	128^3_{ms}	256^3_{ms}	512^3_{ms}
Torus	Ours	800	2.08	4.46	10.75
	Grid		5.3	9.90	76.9
Box	Ours	1380	2.1	4.73	13.51
	Grid		34.4	69.4	202
Teapot	Ours	2304	2.1	4.73	13.51
	Grid		34.4	69.4	202
Monster	Ours	14840	2.58	6.28	17.2
	Grid		14.9	30.3	120
Venusm	Ours	43357	2.51	5.43	15.2
	Grid		37	77.00	208.5
Horse	Ours	96966	3	6.4	20.4
	Grid		77.7	157.5	386.7
Dragon2	Ours	108588	3	5.23	16.1
	Grid		86.4	174	1.04e3
Hand	Ours	654666	9	11.3	22.2
	Grid		509.6	983.2	3.150e3
Dragon	Ours	871414	11.8	14.7	24.9
	Grid		675	1.35e3	2.7

Table 2 This shows the time in FPS (frames per second) needed to render the transparent models with different triangle counts using different methods including one-surfaced refraction, two-surfaced refraction, multiple-surfaced refraction and ray-tracing methods

Name	1-surf.	2-surf.	m-surf.	RT
Torus	4720	2499	111	6.2
Box	4182	2389	89	2.1
Teapot	3612	2041	84	1.29
Monster	2632	1369	69	0.13
Venusm	2685	1357	70	4.4e−2
Horse	1755	908	34	1.2e−2
Dragon2	1617	837	34	8.92e−3
Hand	358	181	27	5.48e−4
Dragon	272	137	15	1.62e−5

(a) (b)

Fig. 11 This demonstrates the strength of recording surface normal and transmittance. Our algorithm can render an object with multiple different materials

efficient than traditional ray tracing. This is because a fixed number of voxels can be used to represent a complex surface model. Thus, the traversal cost can be limited in a controllable amount and so is the efficiency of rendering. When comparing the shark animation rendered with our multiple-surfaced refraction algorithm with the animation rendered with the two-surfaced refraction algorithm, our result has less aliasing artifact. Both animations are provided in the supplement material with our submitted paper. This is because the multiple-refraction algorithm can get a more precise simulation to the real condition of light refracted inside the transparent object than the two-surfaced refraction method.

Figure 10 shows the comparison among Wyman's image-space 2-surfaced refraction method [37], our multiple-surfaced refraction method and the ray-tracing method when rendering the refraction of a glass vase. The image-space

Table 3 This tables shows the performance comparison among 1-, 2- and multiple-surfaced refraction algorithms and the effects of applying the two artifact-removal heuristics based on the correct rate which is computed according to Eq. (6) by setting the threshold to be 0.9962. H_1 stands for the heuristic 1 of thresholding the tracing with an angle and the number denotes the threshold angle. $H_{1,2}$ stands for the heuristic 1 and 2 where the angle is chosen to be optimal for heuristic 1 and the number denotes the degree of extending the range of the clipping region for each slice

Name	1-surf	2-surf	m-surf(no)	m-surf	$H_1(2.5)$	$H_1(5)$	$H_{1,2}(0\,\%)$	$H_{1,2}(5\,\%)$
Torus	0.617	0.203	0.174	0.772	0.599	0.661	0.760	0.772
Box	0.572	0.034	0.016	0.746	0.732	0.733	0.732	0.745
Teapot	0.763	0.325	0.176	0.841	0.550	0.687	0.830	0.840
Monster	0.459	0.1352	0.290	0.568	0.448	0.520	0.568	0.568
Venusm	0.676	0.371	0.242	0.762	0.551	0.615	0.761	0.762
Horse	0.570	0.415	0.360	0.711	0.619	0.687	0.709	0.711
Dragon2	0.616	0.347	0.220	0.675	0.435	0.512	0.676	0.675
Hand	0.717	0.409	0.295	0.744	0.504	0.580	0.737	0.740
Dragon	0.649	0.354	0.251	0.717	0.480	0.563	0.712	0.717

method renders the glass vase as light passing through two planar surfaces. Failure to render objects with self-occlusion is one of the major problems existing in the image-space refraction method. In addition, it cannot simulate the multiple-refraction and total-reflection effects, either. The results generated by our multiple-surfaced refraction method are more closed to the one generated by ray tracing. These results demonstrate that our method can overcome the limitation of convex models and possible transmittance values existing in the two-surfaced refraction method. The tradeoff for this correctness is the efficiency as shown in Table 2. The main cost is from voxelization and traversing the voxels. Figure 9 shows more rendering results using this multiple-surfaced refraction method. Since our multiple-surfaced refraction algorithm aims at better approximating the rays refracted inside the transparent objects. The percentage of transparent-object pixels whose view ray estimated by a refraction algorithm can correctly approximate the truly refracted view ray computed by the ray-tracing algorithm is used as an indicator to quantitatively compare the performance of different refraction algorithms. The ratio is computed as

$$Corr. = \frac{\sum_{j\in\Omega} \mathbf{Test}(j)}{\sum_{j\in\Omega} 1} \tag{6}$$

where Ω is the pixels of the transparent objects and **Test**() tests whether the ray approximation at the pixel j is good or not and is computed as

$$\mathbf{Test}(j) = \begin{cases} 1 & \text{if } \mathbf{Dot}(\mathbf{T}_{\text{est}}(j), \mathbf{T}_{RT}(j)) > th \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where th is a user defined constant to the allowable angle deviation, $\mathbf{T}_{\text{est}}$ is the approximated view ray direction at the pixel j using the 1-, 2-, and multiple-surfaced refraction algorithms and $\mathbf{T}_{RT}()$ is the ray direction at the pixel j traced by ray tracing. Table 3 shows the statistics of applying different artifact-removal heuristics in the multiple-surfaced refraction algorithms. Quantitatively, the angle heuristic leads to obvious improvement under the correct rate metric and the extension heuristic only induces very limited amount of improvement. However, the extension heuristics is important to remove those artifacts which is small and contributes little to the correct rate but are easily noticed by human. The same table also shows the performance comparison among the 1-, 2-, and multiple-surfaced refraction algorithms. Our rendered result can better approximate the refracted view rays when comparing with one-surfaced and two-surfaced refraction methods.

Figure 12 shows the results rendered with refraction and transmittance effects with a uniform transmittance coefficient and refraction index for the entire model. As discussed in Sect. 5, the multiple-surfaced refraction algorithm can better estimate refraction effects and compute the transparent attenuation along the traversal path and thus, it is very easy for our application to render an object that contains parts with different transparent materials as shown in Fig. 11, while other algorithms such as [7, 29, 37] can only render the refraction of the ball without considering the refraction effect of the interior models.

7 Conclusion

Newly available GPU functionalities enable more efficient and realistic voxel-based rendering. The geometry shader can reduce the number of GPU passes down to one time for the slicing-based voxelization algorithm. Additionally, it can also reduce the times of rendering each primitive down to its depth range instead of the number of slices. At the same

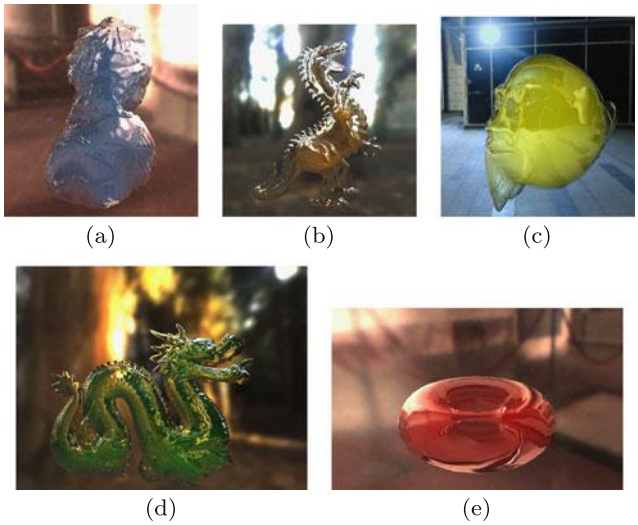


Fig. 12 These demonstrate the usage of multiple-surfaced refraction to render the refraction and transmittance effects of different models. When the traversal length of the view ray through the model becomes larger, the rendering of the ray becomes opaque i.e. the amount of blending with background is less. For example, the tier part in (**c**) is dark green and blends with no background component because the view ray cannot see through the model. All these examples use a homogeneous material and thus the thickness of the view path is the only affecting factor

time, 3D volume textures are used to compute and store surficial and volumetric information including the surface normal, absorption/transmittance coefficient and refraction index in a voxel. 3D textures give us the flexibility of adjusting the voxel resolution according to the hardware capability and the requirement of applications without the strenuous modification in the encoding and decoding process required by the grid encoding voxelization algorithms. Therefore, the voxelization efficiency is improved. After voxelization, the voxel-based information can also be used to generate more realistic lighting effects including the shadow of a complex object and the refractive view of transparent objects. The complex shadow is rendered by voxelizing the object to store an absorption coefficient in each voxel and using absorption coefficients to estimate the amount of light passing through different parts of the object. And when rendering the refraction and transmittance effects of a scene with transparent objects, the surficial refraction index and normal and volumetric transmittance are estimated and stored during the voxelization process and a multiple-surfaced refraction algorithm is proposed to trace the refraction and estimate the attenuation of light rays when passing through transparent objects. The results show that our voxelization is efficient for the proposed applications and the synthesized views look realistic.

There are still two future research directions. First, since different surficial and volumetric information can be recorded in each voxel, it would be interesting to design a user interface to specify and edit the material of each individual voxel in order to give glass artists the ability to simulate the fantastic color of glass artifacts. Second, it will be useful to use the voxelization result with global illumination algorithms such as photon mapping to generate other lighting effects such as caustics.

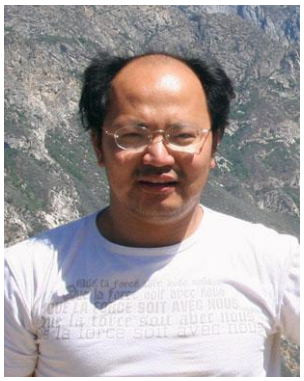
Acknowledgements This work is supported in part by National Science Council (NSC 101-2221-E-011-151, and 101-2221-E-011-153), Taiwan.

References

1. Adelson, S.J.: Simulating refraction using geometric transforms. Master's thesis, University of Utah (2003)
2. Bavoil, L., Myers, K.: Order independent transparency with dual depth peeling. Tech. rep., NVidia (2008)
3. Chen, H., Fang, S.: Fast voxelization of three-dimensional synthetic objects. J. Graph. Tools **3**(4), 33–45 (1998)
4. Diefenbach, P.J.: Pipeline rendering: Interaction and realism through hardware-based multi-pass rendering. PhD thesis, University of Pennsylvania (1996)
5. Diefenbach, P.J., Badler, N.I.: Multi-pass pipeline rendering: realism for dynamic environments. In: Proceedings of the 1997 Symposium on Interactive 3D Graphics, I3D '97, pp. 59–70 (1997)
6. Dong, Z., Chen, W., Bao, H., Zhang, H., Peng, Q.: Real-time voxelization for complex polygonal models. In: PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference, pp. 43–50 (2004)
7. Eisemann, E., Décoret, X.: Fast scene voxelization and applications. In: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 71–78 (2006)
8. Eisemann, E., Décoret, X.: Single-pass GPU solid voxelization and applications. In: GI '08: Proceedings of Graphics Interface 2008. ACM International Conference Proceeding Series, vol. 322, pp. 73–80 (2008)
9. Everitt, C.: Interactive order-independent transparency. Tech. rep., NVidia (2001)
10. Fang, S., Chen, H.: Hardware accelerated voxelization. Comput. Graph. **24**, 200 (2000)
11. Forest, V., Barthe, L., Paulin, M.: Real-time hierarchical binary-scene voxelization. J. Graph. Tools **29**(2), 21–34 (2009)
12. Guy, S., Soler, C.: Graphics gems revisited. ACM Trans. Graph. (Proceedings of the SIGGRAPH Conference) (2004)
13. Haumont, D., Warzee, N.: Complete polygonal scene voxelization. J. Graph. Tools **7**, 3 (2002)
14. Heidelberger, B., Teschner, M., Gross, M.: Real-time volumetric intersections of deforming objects. In: Proceedings of Vision, Modeling, and Visualization, pp. 461–468 (2003)
15. Heidrich, W., Lensch, H., Cohen, M.F., Seidel, H.-P.: Light field techniques for reflections and refractions. In: Rendering Techniques '99, pp. 187–196 (1999)
16. Huang, J., Yagel, R., Filippov, V., Kurzion, Y.: An accurate method for voxelizing polygon meshes. In: Proceedings of the 1998 IEEE Symposium on Volume Visualization, VVS '98, pp. 119–126 (1998)
17. Ix, F.D., Kaufman, A.: (2000). Incremental triangle voxelization
18. Kajiya, J.T., Kay, T.L.: Rendering fur with three dimensional textures. Comput. Graph. **23**(3), 271–280 (1989)
19. Kalaiah, A., Varshney, A.: Statistical geometry representation for efficient transmission and rendering. ACM Trans. Graph. **24**(2), 348–373 (2005)

20. Karabassi, E.A., Papaioannou, G., Theoharis, T.: A fast depth-buffer-based voxelization algorithm. J. Graph. GPU Game Tools **4**(4), 5–10 (1999)
21. Kay, D.S., Greenberg, D.: Transparency for computer synthesized images. Comput. Graph. **13**, 158–164 (1979)
22. Lauterbach, C., Garl, M., Sengupta, S., Luebke, D., Manocha, D.: Fast bvh construction on gpus. In: Proc. Eurographics '09 (2009)
23. Lindholm, E., Kilgard, M.J., Moreton, H.: A user-programmable vertex engine. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, New York, NY, USA, pp. 149–158 (2001)
24. Llamas, I.: Real-time voxelization of triangle meshes on the GPU. In: SIGGRAPH '07, ACM SIGGRAPH Sketches, p. 18 (2007)
25. Lokovic, T., Veach, E.: Deep shadow maps. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '00, pp. 385–392 (2000)
26. Mammen, A.: Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. IEEE Comput. Graph. Appl. **9**(4), 43–55 (1989)
27. Ohbuchi, E.: A real-time refraction renderer for volume objects using a polygon-rendering scheme. In: Proceedings of Computer Graphics International, 2003, pp. 190–195 (2003)
28. Oliveira, G.: Refractive texture mapping, part two. Online tutorial (2000). http://www.gamasutra.com/view/feature/3122/refractive_texture_mapping_part_.php
29. Oliveira, M.M., Brauwers, M.: Real-time refraction through deformable objects. In: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, I3D '07, New York, NY, USA, pp. 89–96 (2007)
30. Shade, J., Gortler, S., He, L.W., Szeliski, R.: Layered depth images. In: SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, pp. 231–242 (1998)
31. Sigg, C., Peikert, R., Gross, M.: Signed distance transform using graphics hardware. In: IEEE Visualization Conference, p. 12 (2003)
32. Sramek, M., Kaufman, A.: Alias-free voxelization of geometric objects. IEEE Trans. Vis. Comput. Graph. **5**(3), 251–267 (1999)
33. Stolte, N.: Robust voxelization of surfaces. Tech. rep., State University of New York at Stony Brook (1997)
34. Varadhan, G., Krishnan, S., Kim, Y.J., Diggavi, S., Manocha, D.: Efficient max-norm distance computation and reliable voxelization. In: SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, pp. 116–126 (2003)
35. Wang, S.W., Kaufman, A.E.: Volume sampled voxelization of geometric primitives. In: VIS '93: Proceedings of the 4th Conference on Visualization '93, pp. 78–84 (1993)
36. Williams, L.: Casting curved shadows on curved surfaces. In: Computer Graphics (SIGGRAPH '78 Proceedings), pp. 270–274 (1978)
37. Wyman, C.: An approximate image-space approach for interactive refraction. ACM Trans. Graph. **24**(3), 1050–1053 (2005)
38. Wyman, C., Davis, S.: Interactive image-space techniques for approximating caustics. In: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, I3D '06, pp. 153–160 (2006)
39. Xu, K., Ma, L.Q., Ren, B., Wang, R., Hu, S.M.: Interactive hair rendering and appearance editing under environment lighting. ACM Trans. Graph. **30**(6), 173:1–173:10 (2011)
40. Zhang, L., Chen, W., Ebert, D.S., Peng, Q.: Conservative voxelization. Vis. Comput. **23**, 783–792 (2007)
41. Zhou, K., Hou, Q., Wang, R., Guo, B.: Real-time KD-tree construction on graphics hardware. ACM Trans. Graph. **27**(5), 126:1–126:11 (2008)

Hsu-Huai Chang received the B.S. degree in Computer Science and Information Engineering from National Taiwan University of Science and Technology in 2011.He is currently working toward the M.S. degree in the Department of Computer Science and Information Engineering in National Cheng-Kung University, His research interest is computer graphics.



Yu-Chi Lai received the B.S. from National Taiwan University, Taipei, R.O.C., in 1996 in Electrical Engineering Department. He received his M.S. and Ph.D. degrees from University of Wisconsin–Madison in 2003 and 2009 respectively in Electrical and Computer Engineering and his M.S. and Ph.D. degrees in 2004 and 2010 respectively in Computer Science.
He is currently an assistant professor in NTUST and his Research interests are in the area of graphics, vision, and multimedia.

Chin-Yuan Yao is an Assistant Professor in the Department of Computer Science and Information Engineering at National Taiwan University of Science and Technology (NTUST), Taipei, Taiwan. He received his M.S. and Ph.D. degrees in Computer Science and Information Engineering from National Cheng-Kung University, Taiwan, in 2003 and 2010, respectively. His research interest is computer graphics, including mesh processing and modeling, and non-photorealistic rendering (NPR).

Kai-Lung Hua received the B.S. degree in electrical engineering from National Tsing Hua University in 2000, and the M.S. degree in communication engineering from National Chiao Tung University in 2002, both in Hsinchu, Taiwan. He received the Ph.D. degree from the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, in 2010. Since 2010, Dr. Hua has been with National Taiwan University of Science and Technology, where he is currently an assistant professor in the Department of Computer Science and Information Engineering. He is a member of Eta Kappa Nu and Phi Tau Phi, as well as a recipient of MediaTek Doctoral Fellowship. His current research interests include

digital image and video processing, computer vision, and multimedia networking.

Yuzhen Niu received the B.S. and Ph.D. degrees from Shandong University, Jinan, China, in 2005 and 2010, respectively, both in computer science. She is currently a Professor in the College of Mathematics and Computer Science at the Fuzhou University, China.
Her research interests are in the areas of graphics, vision and human-computer interaction.

Feng Liu received the B.S. and M.S. degrees from Zhejiang University, Hangzhou, China, in 2001 and 2004, respectively, both in computer science. He is currently a Ph.D. candidate in the Department of Computer Sciences at the University of Wisconsin–Madison, USA.
His research interests are in the areas of graphics, vision and multimedia.