Pacific Graphics (2011)
Bing-Yu Chen, Jan Kautz, Tong-Yee Lee, and Ming C. Lin (Editors)

Short Papers

Real-time Realistic Voxel-based Rendering

S.-H. Chang[1] and Y.-C. Lai[1†] and Y. Niu[2] and F. Liu[2] and K.-L. Hua[1]

[1]National Taiwan University of Science and Technology, Taiwan
[2] Portland State University, U.S.A.

Abstract

*With the advance of graphics hardware, setting 3D texture as render target is newly available to allow voxelization algorithms to record the existence, color and normal information in a voxel directly without specific encoding and decoding mechanism. In this paper two new voxel-based applications are proposed to take advantage of this new functionality for interactively rendering realistic lighting effects including shadow of objects with complex occlusion and refraction and transmission of transparent objects. An absorption coefficient is computed according to the number of surface drawing in each voxel during voxelization and used to compute the amount of light passing through partial occluded complex objects. The refraction and transmission of light passing through transparent objects is simulated by our multiple refraction algorithm using surface normal, transmission coefficient and refraction index in each voxel. All these applications can generate the result in real-time without any preprocessing step. Additionally, we also found that the newly available geometry shader can be used to transform a highly complex surface-represented scene into a set of high-resolution voxels in **only one** GPU pass. This possibly improve the efficiency of the voxelization process.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

1. Introduction

Realism is important for human perception but interactivity is more important for many applications. The refraction and transmission of light as it passes through different materials result in many beautiful and intriguing effects. The requirement of interactivity motivates many research to look for hardware-accelerated approximation to simulate refraction effects [DB97, HLFpS99, Ohb03, Oli, LKM01, Ade03, GS04, WD06, OB07, IZT*07, SZS*08, WZHB09, CO, LES09]. Among these, Eikonal rendering algorithms [IZT*07, SZS*08] which transforms edge boundaries to the gradient of refraction indices for simulating refraction and multiple refraction methods [WD06, OB07, CO, LES09] which use image-based algoirhtms to approximate the light transport through a transparent object are popular because they can provide realistic results in a highly interactive frame rate. However, there are still limitations in Eikonal methods including complex computation in tranformation and difficulties in adding absorption and in image-based methods including the requirement of an extra image map per object for estimating traversal distance, the assumption of a none-self-occluded object and disability to simulate multiple refraction and total internal reflection effects. In this paper we proposed to use volumetric representation to overcome these limitations. GPU-based voxelization first slices the surface models into a set of unit-sized voxels stored in a 3D volume texture. Then, the multiple-refraction method uses the voxelization results to render the refraction and transmission effect. Our algorithm takes advantage of the flexibility and adjustability of the 3D volume texture to compute and store the surface and volume information including normal, refraction index and transmission coefficient for better approximation of refraction and transmission. The information allows us to trace a view ray from the camera into the scene and when intersecting with the surface boundary voxels, the normal and refraction index is used to compute the new propagation direction. 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

[†] Corresponding author, NSC 99-2218-E-011-005-, Taiwan

DOI: 10.2312/PE/PG/PG2011short/001-006

is used to compute the color of the ray. 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 internal reflection effects of possibly self-occluded transparent objects are the advantages of our voxel-based algorithm. The results show that our algorithm perform better than image-based methods on rendering a scene with complex deformable objects.

In addition to the refraction and transmission effects, shadow is also important for human perception because shadow gives the sense of existence. Traditional shadow map [Wil78] is the simplest interactive shadow algorithm but it cannot handle transparent and partially-occluded objects. Transparent shadow proposed by Eisemann et al. [ED06] uses the voxelization result to estimate the traversal length of light passing through transparent and partially-occluded objects and then the attenuation can be computed with the homogeneous absorption coefficient of the entire scene. The assumption of a single coefficient for the entire scene limits the ability to simulate different degree of occlusion in a 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 automatically during voxelization process. 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.

In both applications described above volumetric representation is required. Generally, volumetric data stores properties of an object in a set of regular 3D grids. A voxelization algorithm is needed to transform the surface-boundary representation to a volumetric representation before applying volumetric applications and algorithms. In order to achieve interactivity, real-time slicing-based GPU voxelization algorithms [CF98, FC00, Lla07, KPT99, DCB*04] are proposed to slice models into a set of voxels but their efficiency is limited by the requirement of multiple GPU passes . Therefore, encoding slice-based algorithms [KPT99, DCB*04, ED06, FBP09] 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 primitive, the strenuous process of changing encoding and decoding mechanism when applications change voxel resolution and difficulties in recording surfacial and volumetric information. In addition the number of GPU passes for a high-resolution representation still has chance to be more than one. Through the development process of our voxel-based application, a new GPU-based voxelization algorithm are found to overcome these limitations by using the geometry shader to slice the geometry models using the clipping plane algorithm [FC00] in single GPU pass. The usage of the geometry shader to voxelize the model relieves the need of multiple passes in original slicing-based algorithms and enhances the voxelization efficiency. Then, the adjustable 3D volume texture is used to store the slicing result with other surfacial and volumetric information. Since the size of the 3D texture can be easily adjusted according to the need of application 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 can gain improvement in voxelization efficiency and render all three different lighting effects in real time for a high-resolution voxelization process.

2. Voxelization

The newly available geometry shader is used to slice the surface model 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 surfacial and volumetric information such as transmittance, normal and color for generating more realistic lighting effect as described in Section 3 and 4.

The computation of voxelization is conducted as shown in Fig. 1. Generally, a surface-represented triangle is stored as 3 vertices with their position, normal and other information. 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 and 4 listed in Fig. 1. The geometry shader duplicates the triangle according to the number of slices in the possible range. At the end of the process the slicing algorithm sets up the far and near clipping plane according to the index of the destined slice in order to correctly compute the boundary voxels for the triangle. Then a duplicated triangle is rendered for each proper set of clipping planes.

3. Transparent Shadow Map

Eisemann et al. [ED06] used the voxelization result to estimate the passing distance for rendering transparent shadow but their method did not take different degrees of occlusion and material absorption into account. Our algorithm makes an improvement in computing the shadow by attenuating the light along the traversal path by accumulating the absorption varying with the degree of occlusion and material in the following steps:

Detecting the voxels intersected with triangle

```
For each triangle, $Tri$
    $z_0 = \mathbf{Z}(Tri.V0)$, $z_1 = \mathbf{Z}(Tri.V1)$,$z_2 = \mathbf{Z}(Tri.V2)$
    $max_{slice} = \mathbf{max}(z_0, z_1, z_2)/thickness$
    $min_{slice} = \mathbf{min}(z_0, z_1, z_2)/thickness$
    For $i = min_{slice}$ to $max_{slice}$
        $Plane_{near} = i * thickness$
        $Plane_{far} = Plane_{near} + thickness$
        Set $Plane_{near}$ and $Plane_{far}$ to projection matrix
        If ($\mathbf{Intersect}(Tri)$)
            Rasterize $Tri$ into the slice
```

Figure 1: *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,* $\mathbf{Z}()$ *is a function to extract the depth value of a vertex after transforming the position of the vertex into the camera coordinate,* $\mathbf{max}()$ */* $\mathbf{min}()$ *computes the maximum/minimum value among the set of input values and* $\mathbf{Intersect}()$ *is a function to test whether the triangle is valid after being culled by the clipping planes.*

1. The voxelization camera is set at the position of the light source and aligned with the light direction.
2. Our algorithm voxelizes the scene and computes and stores the absorption 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-defined 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 can be computed using the following equation, $\sum_{i=0}^{s} \alpha(x, y, i) \times E(x, y, i)$ where $\alpha()$ describes the light absorption in this voxel and $E()$ is an occupation flag which 1 represents that the voxel is occupied by some object.

4. Refraction and Transmission

Refraction is the change in propagation direction of a light ray when it transports from one medium to another and the light propagation direction change can be described by the Snell's Law. But single refraction is 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.

4.1. Two-surface refraction

Wyman et al. [Wym05] proposed that multiple refraction may be simplified to a two-surface-refraction effect: 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, $\vec{T}_1$. If the traversal distance, d, between the first and the second refraction point can be estimated, the second refraction position can be estimated with $P_2 = P_1 + d\vec{T}_1$ where P_1 and P_2 are the first and second refraction position, and $\vec{T}_1$ is the refraction direction after the first refraction. Wyman et al. [Wym05] proposed an image-based method to estimate d without considering the first refraction direction. We realized that our voxelization result can find a better estimate of d with a similar manner described in Sec. 3, P_2 can be computed as described previously and projected into the voxel space to extract the surface normal, $\vec{N}_2$ and refraction index and $\vec{T}_2$ can be computed with $\vec{T}_1$, $\vec{N}_2$ and refraction index.

4.1.1. Multiple-surface refraction

The two-surface-refraction method cannot render all transmittance lighting effects when light hits a transparent object in a scene. In addition it also has some limit in the allowable models and transmittance. Thus, a multiple-surface-refraction algorithm is proposed to simulate the refractions and reflections inside a scene. The same voxelization process described in the two-surface-refraction method is used. When rendering the scene, the view initiates a view ray passing through the center of a pixel. Then, the ray is propagated inside the voxel space and every time when the ray hits a boundary voxel, the ray is refracted according to the 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 with $P_i = P_{i-1} + thickness \times T_{voxel}(\vec{T}_i)/cos\theta$ where P_{i-1} is the current position, $thickness$ represents the physical size of the voxel, $\vec{T}_{i-1}$ is the current ray propagation direction, $T_{voxel}()$ 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:

$$\vec{T}_i = \begin{cases} ref(\vec{T}_{i-1}, N_v(P_i), T_v(P_i)) & if(E(P_i)) = 1 \\ \vec{T}_{i-1} & otherwise \end{cases} \quad (1)$$

where $ref()$ computes the new ray direction according to the 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 hit the boundary of the volume.

However, there are still problems when applying the algorithm due to the finite resolution of voxels. The most frequent one is multiple boundary voxels along the traversal path. The surface locality information is used to relieve this issue. We observed that when the angle between the normals of surfaces where the consecutive refractions happens is small, 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. The second problem is when a ray hits the corner of two boundary voxels, the algorithm may not find the right refraction point. 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 intersecting voxel issues described previously. Our algorithm proposed another relief to this missing voxel issue based on the observation that a missing voxel issue is much harder to handle properly than a multiple intersected voxel 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 intersected voxel situations and reduce the chance of missing voxel situations. Then the angle threshold discussed in the previous paragraph can be used to get a good rendering result.

4.2. Transmittance

When light passes through a medium, the amount of energy passing through will decrease and this phenomenon can be described by transmittance. The simplest method [ED06] 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 [ED06] is limited to an object with homogeneous material. Our voxelization result contains the surface normal information, the 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 $\| thickness \times T_{voxel}(\vec{T_i})/cos\theta \|$ which is a byproduct of refraction ray estimation. Then the transparent attenuation can be computed by $transparency = \prod_{i=0}^{N} e^{\sigma(d_i)}$ where N is the total number of refraction points along the traversal path and $\sigma()$ calculates the transparent attenuation of the object [ED06].

5. Results and Discussions

5.1. 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. The 3D volume texture is implemented with the format of 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.

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 $64MB$ to store the data. According to the graphics card used in the test, our voxelization algorithm are 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 $512 \times 512 \times 512$. The results are shown in Table 1. In addition the time required to voxelize the same set of models under these three resolutions using the algorithm proposed by Ignacio et al. [Lla07] is also shown in Table 1. The main reason to compare against their algorithm is due to being the derivative of slicing-based algorithm and the same usage of 3D texture to store the voxelization result. However, their algorithm processes the primitives in a model once per slice. 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 higher resolution. However, because their algorithm can process the slices in sequential order to set up proper stencil buffer in 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 found that the difference between the best and worst performance is roughly $10ms$. In addition to the cost factors from the voxelization process there are other cost factors when applying the voxelization result to render the lighting effects proposed in Section 3 and 4. 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

Name	Alg.	# Tris	128^3_{ms}	256^3_{ms}	512^3_{ms}
Torus	Ours	800	1.66	3.73	10.91
	Grid		2.67	7.29	641.03
Venusm	Ours	43357	3.19	5.04	12.41
	Grid		12.41	73.02	746.27
Horse	Ours	96966	4.28	7.65	15.09
	Grid		15.09	156.99	925.93
Hand	Ours	654666	11.5	20.00	27.68
	Grid		27.68	632.91	2500
Dragon2	Ours	871414	16.54	23.85	31.06
	Grid		31.06	1282.05	3448.28

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 [Lla07] et al. marked with **Grid**. The performance is measure in **ms**.*

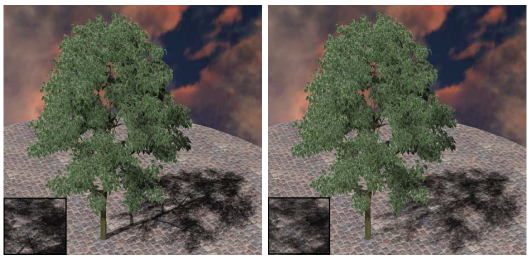

Figure 2: *Rendering the shadow of a surface-represented tree model using ray tracing is too time consuming and thus transparent shadow map is a better choice. The right is a tree rendered with a uniform absorption coefficient for the entire tree. The right is a tree rendered with different voxel absorption 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.*

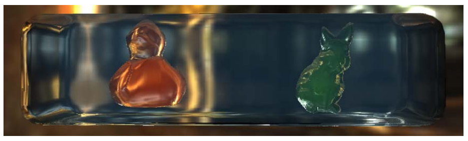

Figure 3: *This demonstrates the strength of recording surface normal and transmittance. Our algorithm can render an object with multiple different materials.*

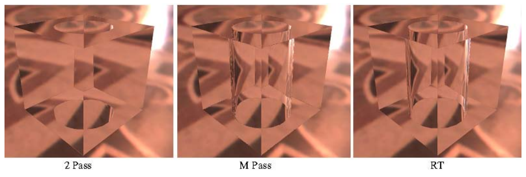

Figure 4: *The left, middle and right are rendered using the two-refraction [Wym05], multiple-refraction and ray tracing methods.*

will also increase. 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, general practice is to reduce the number of slices with the increase in the slice resolution to reduce the number of processed slices with acceptable rendering quality.

5.2. Transparent Shadow

A tree rendered with its transparent shadow is shown in Fig. 2. 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 degree 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 because the number of drawing in each pixel is the byproduct of voxelization.

5.3. Refraction and Transmission

The multiple-surface-refraction algorithm can simulate the multiple refraction and total internal reflection effects to generate realistic refraction in real time. It is generally more 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. Fig. 4 shows the comparison among Wyman's image-based 2-refraction method [Wym05], multiple-refraction and ray tracing methods when rendering the refraction of a glass vase. The image-based 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-based refraction method. In addition, it cannot simulate the multiple refraction and total internal reflection effects, either. The results generated by our multiple-refraction method are more closed to the one generated by ray tracing. These results demonstrate that our method can overcomes the limitation of convex models and possible transmittance values existing in the two-surface-refraction and image-based refraction methods.

It is very easy for our application to render an object that contains parts with different transparent materials as shown in Fig. 3 while other algorithms such as [ED06, Wym05, OB07] can only render the refraction of the ball without considering the refraction effect of the angel.

6. Conclusion

Newly available GPU functionalities enable more efficient and realistic voxel-based rendering. In this paper 3D volume texture is used to compute and store surfacial and volumetric information including the surface normal, attenuation/transmission coefficient and refraction index in a voxel. The information is used to generate more realistic lighting effects including the shadow of a complex object and the refractive view of a transparent model. Through the development of these new applications, we found that the geometry shader can duplicate the triangles during the voxelization process to reduce the GPU rendering pass down to one time. At the same time 3D volume texture also 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. The price paid in our voxelization method is the extra triangles drawn per voxelized triangle but the voxelization efficiency is still improved. Although we demonstrate the new voxel-based applications using our geometry-shader-based voxelization algorithm, the same concept can be easily incorporated with other voxelization algorithms by adding an extra process to compute the surface properties. Through our observation some of the refraction artifact may go away if conservative voxelization algorithms such as [SS10] are used. Finally, the results presented in this paper demonstrate that our voxel-based applications and geometry-shader-based voxelization is efficient and flexible for real-time voxelization and applications.

References

[Ade03] ADELSON S. J.: *Simulating Refraction Using Geometric Transforms*. Master's thesis, 2003. 1

[CF98] CHEN H., FANG S.: Fast voxelization of three-dimensional synthetic objects. *J. Graph. Tools 3*, 4 (1998), 33–45. 2

[CO] CHAUDHARI P., OLANOY M.: Real-time multiple refractions through deformable objects. 1

[DB97] DIEFENBACH P. J., BADLERT N. I.: Multi-pass pipeline rendering: Realism for dynamic environments. 59–70. 1

[DCB*04] 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* (2004), pp. 43–50. 2

[ED06] EISEMANN E., DÉCORET X.: Fast scene voxelization and applications. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006), pp. 71–78. 2, 4, 6

[FBP09] FOREST V., BARTHE L., PAULIN M.: Real-time hierarchical binary-scene voxelization. *Journal of Graphics Tools 29*, 2 (2009), 21–34. 2

[FC00] FANG S., CHEN H.: Hardware accelerated voxelization. *Computers and Graphics 24* (2000), 200–0. 2

[GS04] GUY S., SOLER C.: Graphics gems revisited. *ACM Transactions on Graphics (Proceedings of the SIGGRAPH conference)* (2004). 1

[HLFpS99] HEIDRICH W., LENSCH H., F M. C., PETER SEIDEL H.: Light field techniques for reflections and refractions. In *In Rendering Techniques ąę99* (1999), pp. 187–196. 1

[IZT*07] IHRKE I., ZIEGLER G., TEVS A., THEOBALT C., MAGNOR M., SEIDEL H.-P.: Eikonal rendering: Efficient light transport in refractive objects. *ACM Trans. on Graphics (Siggraph'07)* (Aug. 2007), to appear. 1

[KPT99] KARABASSI E.-A., PAPAIOANNOU G., THEOHARIS T.: A fast depth-buffer-based voxelization algorithm. *journal of graphics, gpu, and game tools 4*, 4 (1999), 5–10. 2

[LES09] LEE S., EISEMANN E., SEIDEL H.-P.: Depth-of-field rendering with multiview synthesis. *ACM Trans. Graph. 28* (December 2009), 134:1–134:6. 1

[LKM01] 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* (New York, NY, USA, 2001), SIGGRAPH '01, pp. 149–158. 1

[Lla07] LLAMAS I.: Real-time voxelization of triangle meshes on the gpu. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches* (2007), p. 18. 2, 4, 5

[OB07] OLIVEIRA M. M., BRAUWERS M.: Real-time refraction through deformable objects. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), I3D '07, pp. 89–96. 1, 6

[Ohb03] OHBUCHI E.: A real-time refraction renderer for volume objects using a polygon-rendering scheme. In *Computer Graphics International, 2003. Proceedings* (july 2003), pp. 190 – 195. 1

[Oli] OLIVEIRA G.: Refractive texture mapping, part two. 1

[SS10] SCHWARZ M., SEIDEL H.-P.: Fast parallel surface and solid voxelization on gpus. *ACM Trans. Graph. 29* (December 2010), 179:1–179:10. 6

[SZS*08] SUN X., ZHOU K., STOLLNITZ E., SHI J., GUO B.: Interactive relighting of dynamic refractive objects. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (2008), pp. 1–9. 1

[WD06] WYMAN C., DAVIS S.: Interactive image-space techniques for approximating caustics. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games* (2006), I3D '06, pp. 153–160. 1

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *In Computer Graphics (SIGGRAPH ąę78 Proceedings* (1978), pp. 270–274. 2

[Wym05] WYMAN C.: An approximate image-space approach for interactive refraction. *ACM Trans. Graph. 24*, 3 (2005), 1050–1053. 3, 5, 6

[WZHB09] WALTER B., ZHAO S., HOLZSCHUCH N., BALA K.: Single scattering in refractive media with triangle mesh boundaries. *ACM Transactions on Graphics 28*, 3 (aug 2009). 1