Deep Shadow Maps from Volumetric Data on the GPU

Adam J. Shook
University of Maryland Baltimore County

Abstract

A method of generating Deep Shadow Maps from a 3D data set is presented. This method uses ray tracing on the GPU to accumulate opacity and store them in a deep shadow map. The deep shadow map is then sampled based on view direction to determine how much light got to a particular fragment. The shadow maps can also be used to cast shadows onto other objects.

Introduction

Much research has gone into improving shadow mapping techniques, from completely eliminating shadow aliasing to allowing transparent objects to cast shadows. Deep Shadow Maps (DSMs) improve over the standard shadow mapping technique by storing a representation of the visibility of all possible depths. DSMs allow shadows from partially transparent surfaces and volumetric objects, such as hair and fog. DSMs are traditionally created by rendering the scene from the light's perspective to build the shadow map based on an object's material properties. This paper presents a method of generating DSMs from volumetric data stored in a 3D texture. These shadow maps are then used to self-shadow the volumetric data as well as cast shadows onto other objects.

Related Works

Traditional shadow maps [6] are rendered by placing a camera at the light source facing all of the objects that are going to be casting shadows. The scene is then rendered, storing the depth of each fragment into the buffer, which is sometimes referred to as a *depth map*. From here, the scene is rendered as usual. The depth of each fragment is compared with the value inside the shadow map. If the sampled depth is greater than the current fragment's depth, then the fragment is considered to be in shadow and is shaded as such.

Lokovic and Veach [5] introduced an extension of the traditional technique called Deep Shadow Maps. Instead of storing the depth of a fragment in the map, DSMs store a representation of the visibility through a pixel at all possible depths. While traditional shadow maps can only tell you if a fragment is occluded by another fragment, it cannot tell you anything about the fragment itself. Deep shadow maps attempt to remedy this by storing this representation. Because of this, we can shade fragments that are partially occluded by fog or sparse data like hair.

Amantatides and Woo [1] presented an algorithm to quickly traverse a voxelization of primitive data. A 3D grid is created to form smaller 3D cubes or voxels. From here, a scene is parsed and the primitives are stored in individual voxels. The presented traversal algorithm provides a way to quickly traverse this voxelization for a great performance enhancement.

Eisemann and Décoret [2] presented a hardware implementation of voxelization as well as many applications of their technique. A scene is voxelized in real time on the GPU into a 2D texture called a *slice map*. Each bit in a pixel of a 2D texture represents a voxel, where a value of 1 represents a fragment is present in the voxel. This allows for a 3D representation of a scene with a depth of 32 "voxels" using a single texture. Using multiple render targets will allow for greater depths. The related application of this technique was using this voxelization to create what they called Transmittance Shadow Maps. It allows for colored shadows due to semi-transparent objects such as glass.

Enderton et al. [3] offer a technique called Stochastic Transparency to render accurate shadows with many types of transparent geometry such as hair, smoke, foliage, etc. It utilizes a stochastic sampling approach that produces correct alpha-blended colors in a single render pass with no sorting, but introduces noise. Several smoothing methods were presented to reduce the noise in the image. Comparisons were made to other algorithms, specifically depth peeling which requires many passes to produce a correct image. The algorithm presented produces a correct image extremely similar to depth peeling but in a much shorter time.

Implementation

The chosen implementation uses 3D textures to store volumetric data sets. Cubes are physically rendered and then a hardware accelerated implementation of a ray tracer is used to cast rays through the 3D texture in order to both generate a deep shadow map and produce the final shadowed rendering.

First, cubes are rendered utilizing the light's view and projection matrix. The bulk of the ray tracing algorithm for the deep shadow map generation is in the fragment shader. The RGB channels of the deep shadow map store the depth of a chosen opacity, while the alpha channel stores the maximum opacity for the given fragment. The chosen opacities were 10%, 50%, and 90% respectively. These particular values were chosen based off a presentation by Kobayashi [4]. The following pseudocode segment will outline the ray tracing routine to generate a deep shadow map:

1. Determine the texture-space coordinates where a ray enters the cube.
2. Determine the normalized direction the ray will be cast through this cube based on the world-space vertex coordinates and the position of the light.
3. Choose an arbitrary scaling value to determine the length of each step. The smaller this value, the more samples but better results.
4. Begin tracing through texture space. Sample the 3D texture at each step and attenuate the opacity as you are tracing.
5. For 10% opacity, 50% opacity, and 90% opacity, record the depth that has been traversed so far in the RGB channels of the texture, respectively.
6. Once the entire cube has been traversed, store the maximum opacity into the alpha channel of the texture.

Our deep shadow map is now created. We have the depths at 10%, 50%, and 90% opacity and the maximum opacity value of the texture.

For Step 5 above, the previously calculated opacity and depth were stored as each voxel was processed. The previous opacity and current opacity were used to determine when the borders of 10%, 50%, and 90% were crossed. At each border, a smooth step was used to determine at what location in the interval [0, 1] the border was crossed. The previous depth and current depths are then linearly interpolated based on this value. This linearly interpolated value is then stored in the appropriate channel.

Another approach to storing the depths would be to just check if the current opacity is less than 10%, 50%, or 90% with a simple if-else if block. Store the current depth in the channel that falls into the particular "bucket". For example, once the border is crossed from 10% to 50%, the red channel will have the greatest depth that is less than or equal to 10%. This will produce less accurate results but increase performance. In most cases, the decrease in accuracy is not noticeable – especially with a large number of steps through the data.

Once the deep shadow map is created, the scene is rendered using the camera's view and projection. Ray tracing is used again to accumulate color through the data set to use as the base for our final rendering. A standard volume rendering technique is used in a similar manner of how the DSM was generated.

1. Determine the texture-space coordinates where a ray enters the cube.
2. Determine the normalized direction the ray will be cast through this cube based on the world-space vertex coordinates and the position of the camera.
3. Choose an arbitrary scaling value to determine the length of each step. The smaller this value, the more samples but better results.
4. Begin tracing through texture space. Sample the 3D texture and accumulate color as we traverse. The sampled value is used as each RGBA channel.
5. Return the final color once parsing is complete.

Once we have the color, we now need to sample our DSM along a ray to determine the appropriate opacity for the fragment. Figure 1 shows a 2D visual representation of the rays being used. The extension to 3D is trivial.

1. Determine the texture-space coordinates where a ray enters the cube.
2. Determine the normalized direction the ray will be cast through this cube based on the world-space vertex coordinates and the position of the camera.
3. Using this direction and origin, calculate where the ray intersects with the other side of the cube.
4. Calculate the length of this line segment and divide it by the desired number of samples.
5. Begin stepping through the cube. For each sample point in the cube, create a ray with this origin and a direction towards the light position.
6. Determine where this ray will exit the cube.
7. Calculate the length between these two points. This is the depth we will use to compare with our DSM.
8. Project the point in the cube into shadow map space using the light's WVP matrix and sample.
9. Determine the opacity for this point based on the DSM (explained below).
10. Average all the sampled opacities to determine the lighting.

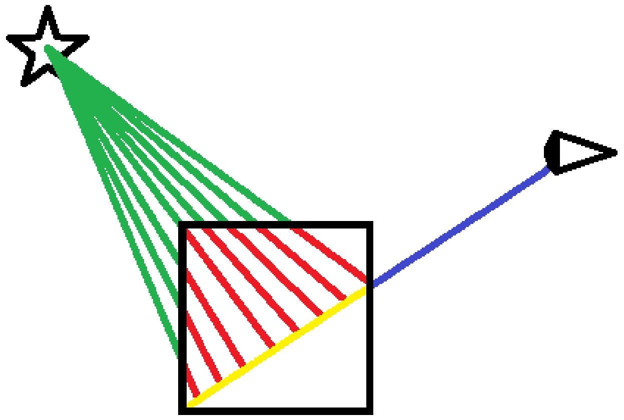

Figure 1: Yellow represents the ray inside the cube that is based on two points and a direction from the eye. Red are the projections of a sample point on the yellow line to their associated point on the cube. These points are then projected into light space texture coordinates and used to sample our deep shadow map.

You may recall the RGB channels are storing depths at 10%, 50% and 90% opacity, respectively. We use these stored depth values to find which two channels the current depth lies between. From here, the smooth step operation is used to determine a value between [0, 1] as to where the current depth lies between the two depths in the channels. This value is then used to linearly interpolate between the appropriate opacities (10%, 50%, or 90%). This interpolated value is then returned as the opacity at that particular location in the cube. If the depth is less than the closest depth (red channel), a value of 0 is returned, anything greater than the furthest recorded depth (blue channel) returns the maximum opacity stored in the alpha channel.

All of these opacities are averaged together for each ray to determine the final lighting at that fragment. The number of samples is up to the user to hit the balance between performance and quality.

The final lighting value is used in coherence with the color from the initial volume ray trace to darken the fragment.

Finally, the DSM can be used to cast shadows on arbitrary objects outside of the volumetric data set. The value stored in the alpha channel is the maximum opacity of each ray cast through the volume. This value can be retrieved in a fragment shader and used to shadow a plane where appropriate.

Results

The results presented in this paper were generated using DirectX 9 and HLSL 3.0 on an NVIDIA GeForce GTX 465, an Intel Core 2 Duo processor at 2.66 GHz, and 4 GB of DDR2 RAM. A maximum of 512 steps were used to create the DSM and the volume rendering, while 128 steps were used to sample the DSM. All images were produced in real time on the above hardware configuration – between 180 and 330 FPS based on the size of the data set.

Figure 2 shows renderings of deep shadow maps for four different volumetric data sets – a standard box that gets more opaque towards the center of the box (64x64x64), a set of bucky balls (32x32x32), crossed rods (64x64x64), and clouds (512x512x32). The DSM itself is shown on the far left, and the individual RGBA channel values are also represented. For the RGB channels, brighter pixels represent a greater depth until the 10%, 50%, or 90% opacity was met for a particular fragment. For the alpha channel, brighter values represent a higher maximum opacity for the fragment.

Figure 3 shows renderings of the box data set. Here, we can a straight accumulation of the data, the final lighting value accumulated from traversing the DSM, and the data darkened with this value.

Figure 4 shows shadows cast on a plane using the maximum opacity stored in the DSM.

Due to the nature of these volumetric sets, there is often noise that is sampled where usually there would be empty space. 3D textures usually contain a single value at each voxel. Often, 2D textures are sampled to return a color and opacity based on the sampled value. This will allow for volumetric data to contain colors and opacities determined by an artist. For this paper, the sampled value was used across the board for the RBGA channels, resulting in blurrier renderings of the data itself. These artifacts can be seen as streaks and blurred data in the volume renderings in Figure 4.

Future Work

The presented algorithm can be extended to allow for colored data as well as colored shadows. By using multiple render targets, the RGB channels can be used to store depth information related to the amount of red, green, and blue at each pixel. These values can then be attenuated in a similar manner to color the shadows instead of just darkening them. These color values can also be used to color shadows cast onto other objects.

Currently, two ray tracing steps are used during the final render stage to acquire the volume rendering and the opacity rendering. These two could be reduced into one, thus slightly increasing performance.

Further research can be done to extend this ray tracing approach to generate and utilize shadow maps for more than one volumetric data set.

Acknowledgments

Data sets initially retrieved from the University of Erlangen Volume Library. The provided utility was then used to turn the native PVM format into a raw binary format. The raw data set was then converted by hand to a 3D texture in a DDS format with DXT5 compression to be used for this paper.

References

[1] John Amanatides and Andrew Woo. 1987. "A Fast Voxel Traversal Algorithm for Ray Tracing.", In Proceedings of Eurographics '87. Eurographics Association, pp. 3-10.

[2] Elmar Eisemann and Xavier Décoret. 2006. "Fast Scene Voxelization and Applications." In Proceedings of I3D'06: the Symposium on Interactive 3D Graphics and Games. ACM, New York, NY, USA. 71-78.

[3] Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. 2010. "Stochastic Transparency." In Proceedings of I3D '10. ACM New York, NY, USA. 157-164.

[4] Mach Kobayashi. 2001. “Deep Shadows.” Stupid RenderMan Tricks.

[5] Tom Lokovic and Eric Veach. 2000. "Deep Shadow Maps." In Proceedings SIGGRAPH 2000. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. 385-392

[6] Lance Williams. 1978. "Casting Curved Shadows on Curved Surfaces." In Proceedings of SIGGRAPH '78. ACM, New York, NY. 270-274.

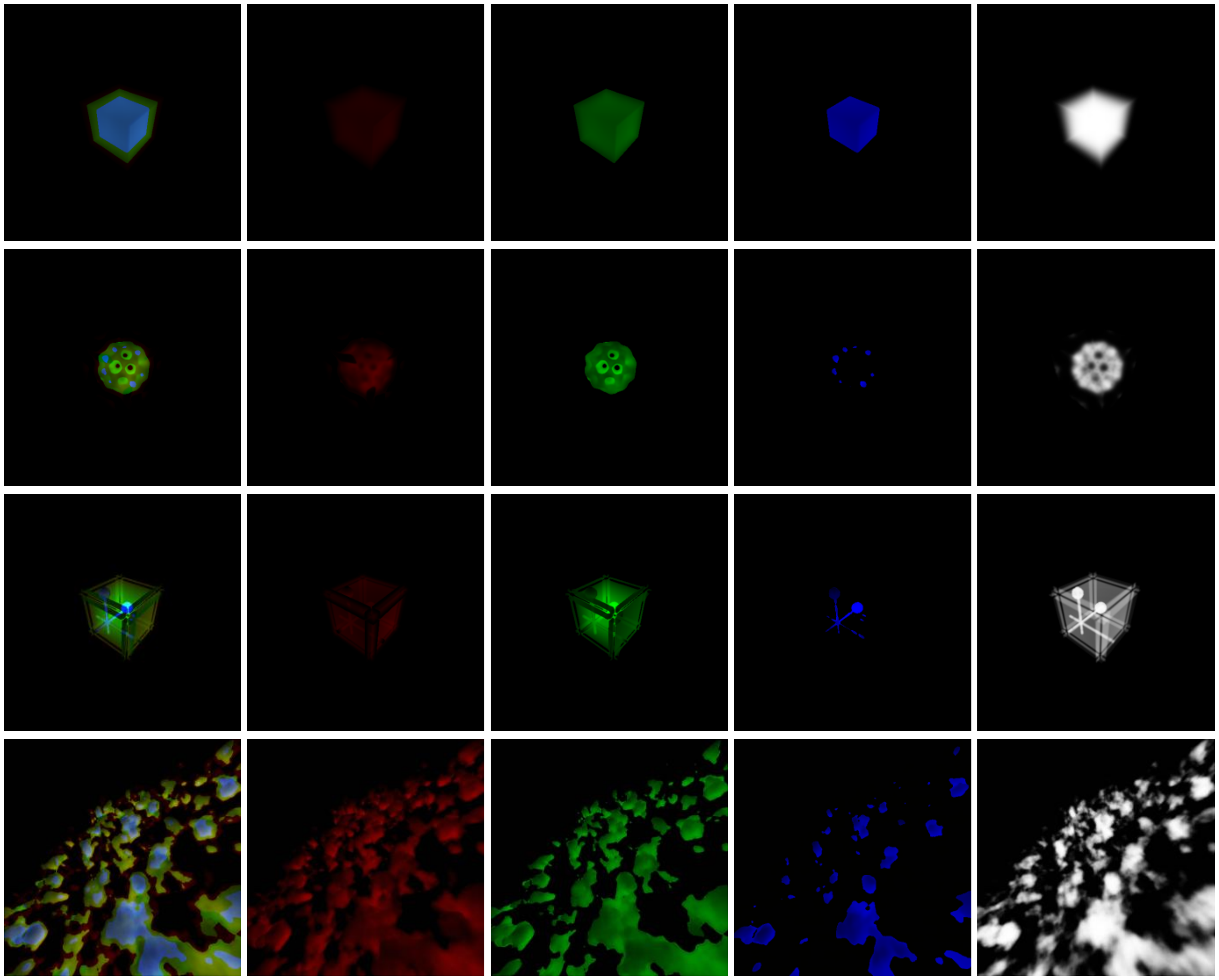
Figure 2: 512x512 Deep Shadow Maps of four volumes: Box, Bucky Ball, Crossed Rods, Clouds.
From left to right: 1) Entire Deep Shadow Map 2) Red Channel (depth at 10%) 3) Blue Channel (depth at 50%) 4) Green Channel (depth at 90%) 5) Alpha Channel (Maximum Opacity at each pixel).
Brighter values represent greater depths and a larger maximum opacity

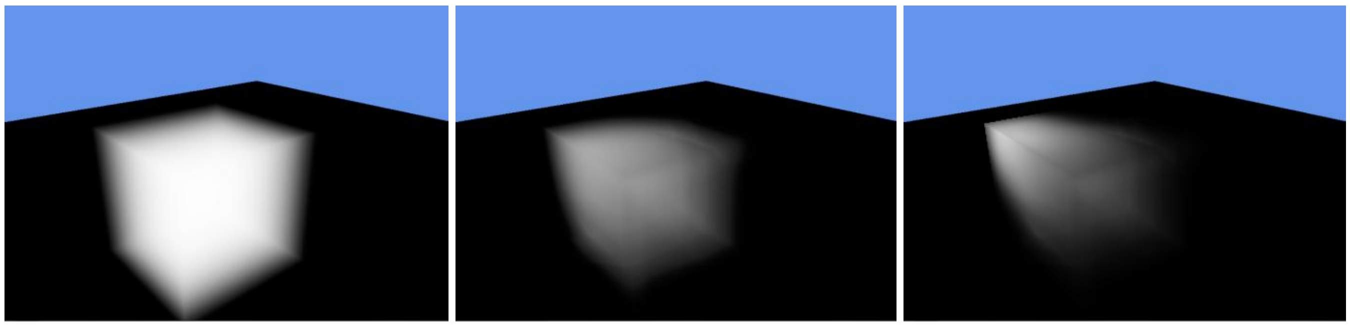
Figure 3: Renderings of a volumetric box that gets more transparent as the distance grows from the center. From left to right: 1) The volumetric rendering without any shadowing. 2) Combination of the volumetric rendering and the sampled opacity. 3) The opacity generated from sampling the DSM.

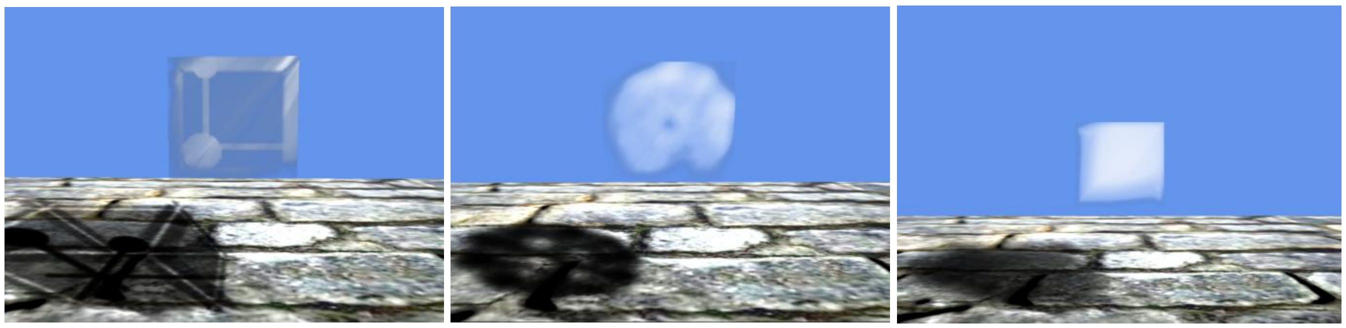
Figure 4: Renderings showing the shadow on a plane taken from a DSM.
From left to right: 1) Crossed rods 2) Bucky Balls 3) Box