Implementation of Shadows Using Splatting

Caixia Zhang, Roger Crawfis
Department of Computer and Information Science
The Ohio State University, Columbus, OH

ABSTRACT

This paper describes an efficient algorithm to model the light attenuation and implementation of shadows with more participating media and low albedo. The light attenuation is modeled by using the single splatting for both the viewer and the light source. During the rendering, a 2D shadow buffer attenuates the light for each pixel. When the contribution of a footprint is added to the image buffer as seen from the eye, we add the contribution to the shadow buffer as seen from the light source. We have generated shadows for point lights and parallel lights using this algorithm. The shadow algorithm has been extended to deal with multiple light sources and projective textured lights.


Keywords: visualization, volume rendering, shadows, illumination

1 INTRODUCTION

Volume rendering is the display of datasets sampled in three dimensions. There are four popular volume rendering algorithms: raycasting, splatting, shear-warp, and hardware-assisted 3D texture mapping. Based on the comparison and evaluation of the four algorithms [8], splatting can create high-quality images, and render efficiently in the case of sparse dataset. Splatting was proposed by Westover [17], and its basic principles are: (1) represent the volume as an array of overlapping basis functions with amplitudes scaled by the voxel values; (2) project these basis functions to the screen to achieve an approximation of the volume integral. A major advantage of splatting is that only relevant voxels are projected and rasterized. This can dramatically reduce the volume data that needs to be processed and stored.

A shadow is a region of relative darkness within an illuminated region caused by an object totally or partially occluding the light. Shadows are essential to realistic images. Earlier implementations of shadows focused on hard shadows, in which a value of 0 or 1 is multiplied with the light intensity. The shadow volume algorithm by Crow [4] introduces the concept of shadow volumes. A shadow volume is the polygonalized solid that models the volume of a shadow cast into space by the silhouette of an occluder. During the rendering, a visible point is first verified that it does not fall inside such a shadow volume before it is illuminated by the light source. In the 2-pass hidden surface algorithm by Nishita and Nakamae [13] and Atherton et al. [1], the first pass transforms the image to the view of the light source, and separates shadowed and unshadowed portions of the polygons. Then a new set of polygons is created, each marked as either completely in shadow or not. In the second pass, visible determination from the eye is done, and the polygons are shaded taking into account their shadow flag. This 2-pass hidden surface algorithm is only suitable for polygon primitives. Williams [20] uses a z-buffer depth map algorithm to generate shadows. A light source depth map is first created with respect to the light source. During the rendering, the z-buffer depth map is used to determine if an object point visible from the eye is also visible from the light source. This algorithm supports primitives other than just polygons, but it has aliasing problems due to discretized depth map cells. New graphics hardware can generate shadows without participating media. For example, NVIDIA GeForce4 video cards are used to do rendering calculation and implement shadows.

The shadow volume algorithm, 2-pass hidden surface algorithm and z-buffer depth map algorithm can only determine if an object point is in shadow or not, resulting in only binary values for the light intensity. These algorithms are not suitable for volume rendering. In volume rendering, as the light traverses the volume, the light intensity is continuously attenuated by the volumetric densities. Raytracing offers the flexibility to deal with the attenuation of the light intensity. Raytracing has been used to generate shadows for both surface representations [19] and volumetric datasets [16]. Here we investigate a new shadow algorithm that properly determines this light attenuation and generates the shadows for volumetric datasets, using a splatting paradigm for volume rendering.

Behrens [2] uses texture mapping hardware to add shadows to a texture-based volume renderer. A shadowed volume which contains the light attenuation information is first produced by the hardware using the original unshadowed volume and the light vector. The shadowed volume is then rendered using texture-based volume rendering. The resulting image has diffusely illuminated effects and the performance decreases by less than 50% when shadows are added. However, for high performance, it is limited to parallel light sources. Lokovic and Veach [7] proposed the concept of deep shadow maps to deal with light attenuation. A deep shadow map is a rectangular array of pixels in which every pixel stores a visibility function. The function value at a given depth is the fraction of the light beam's initial power that penetrates to that depth. The deep shadow map is equivalent to computing the approximate value of (1.0 - opacity) at all depths. They implemented deep shadow maps in a highly optimized scanline renderer. However their work gives us some ideas to deal with the light attenuation in volume rendering using splatting.
Nulkar and Mueller have implemented an algorithm to add shadows to volumetric scenes[14] using splatting. They use a two-stage splatting approach: in the first-stage, splatting is used to construct a light volume; the second stage is formed by the usual rendering pipeline (the only difference is that the light contributions are interpolated from the light volume). Since the algorithm needs a 3D buffer to store the light volume, it has the problem of high storage and memory cost. Here, we investigate a new algorithm to implement shadows using splatting that requires only a 2D buffer for each light source.

In this paper, we focus on generating shadows using image-aligned slicing algorithms, in particular image-aligned sheet-based splatting. The algorithm uses the same splatting for both the light attenuation and the rendering, as seen from the light source and from the eye respectively. In the following section, the image-aligned sheet-based splatting is reviewed and the motivation of this work is given. Section 3 describes the basic shadow algorithm for a single light source. Sections 4 and 5 are the extensions of the basic shadow algorithm: multiple light sources and projective textured lights. Section 6 discusses the accuracy issues and the conclusions are given in Section 7.

2 IMAGE-ALIGNED SHEET-BASED SPLATTING

In splatting, each voxel is represented by a 3D kernel weighted by the voxel value. The 3D kernels are integrated into a generic 2D footprint along the traversing ray from the eye. This footprint can be efficiently mapped onto the image plane and the final image is obtained by the collection of all projected footprints, weighted by the voxel values. This splatting approach is fast, but it suffers from color bleeding and popping artifacts due to incorrect volume integration.

In order to mitigate this problem, Westover proposed the sheet-buffer splatting method [18], in which the voxels are summed within volume slices most parallel to the image plane and stored in the sheet buffer. The sheets are then composited together to form the final image. But this improved splatting introduces a more substantial popping artifact when the orientation of the sheets changes as the viewpoint moves. Mueller et. al. [12] eliminates this popping drawback by aligning the sheets to be parallel to the image plane. This splatting method (as shown in Figure 1) is called image-aligned sheet-based splatting. All the voxel kernels that overlap a slab are clipped to the slab and summed into a sheet buffer. The sheet buffers are composited front-to-back to form the final image. While this significantly improves image quality, it requires much more compositing and several footprint sections per voxel to be scan-converted. Using a front-to-back traversal, this method can make use of the culling of occluded voxels by keeping an occlusion map and checking whether the pixels that a voxel projects to have reached full opacity [6].

The motivation of this paper is to implement shadows using the image-aligned sheet-based splatting to create more realistic and informative images.

3 BASIC SHADOW ALGORITHM FOR A SINGLE LIGHT SOURCE

3.1 Illumination Models

In splatting, we calculate per-pixel illumination at each sheet, then composite the sheet with its previous sheets by the following formula:

\[
I_o = I_c + (1 - A_c) * (I_n * A_n)
\]

\[
A_o = A_c + (1 - A_c) * A_n
\]

For a front-to-back traversal:

For a back-to-front traversal:

\[
I_o = ((1 - A_n) * I_c) + (I_n * A_n)
\]

\[
A_o = ((1 - A_n) * A_c) + A_n
\]

where \(I\) denotes the intensity, \(A\) denotes the opacity, \(o\) denotes the output, \(c\) denotes what is already in the image buffer, and \(n\) denotes the new point in the current sheet.

For the per-pixel illumination at each sheet, the illumination model we use is:

\[
C(x) = C_{obj}(x) * (k_d I_a + k_a I(x) * (N(x) \cdot L(x)))
+ k_s I(x) * (E(x) \cdot R(x))^s
\]

where \(k_a\) is the material’s ambient reflection coefficient, \(k_d\) is the diffuse reflection coefficient, \(k_s\) is the specular reflection coefficient, \(k_n\) is the Phong exponent, \(C_{obj}(x)\) is the diffuse color of the object at the location corresponding to the pixel at the sheet (determined by the transfer function), \(I_a\) is the intensity of the ambient light, \(I(x)\) is the intensity of the light which is the fraction of the original light intensity that penetrates to the location \(x\) from the light source, \(N(x)\) is the normal vector (determined by the gradient), \(L(x)\) is the light vector, \(E(x)\) is the eye vector, and \(R(x)\) is the reflection vector.

Here, \(k_a\), \(k_d\), \(k_s\), \(k_n\) and \(I_a\) are independent of the sample location. However, \(C_{obj}(x)\), \(I(x)\), \(N(x)\), \(L(x)\), \(E(x)\), and \(R(x)\) are not.
\( E(x) \) and \( R(x) \) are functions of the location \( x \). \( N(x) \) is calculated by estimating the gradient at each pixel using central differences. The object color \( C_{\text{obj}}(x) \) is determined from the transfer function using the value at the pixel. For the implementation of the shadows, the main work is to determine the intensity of the light \( I(x) \) arriving at the location \( x \). The intensity of the light is decreased due to light attenuation as light traverses the volume.

### 3.2 Implementation of Shadows Using Splatting

Visibility algorithms and shadow algorithms are essentially the same. The former determine the visibility from the eye, and the latter determine the visibility from the light source. However, it’s hard to implement shadows, especially accurate shadows, in volume rendering, because the light intensity is continuously attenuated as the light traverses the volume. We need to determine the light intensity arriving at the point being illuminated.

Nulkar and Mueller [14] use two-stage splatting to add shadows: first splat the volume with respect to the light source using the image-aligned splatting algorithm and store the intensity value at each pixel for each sheet; secondly, splat the volume with respect to the eye to render the volume. Due to the inconsistency between the two splattings, they construct a light volume to store the intensity values after the first-stage splatting. The advantages of this approach include pre-processing intensity calculation and view-independent light volume. But, accurate shadows are difficult to implement using this method due to the high requirement of resolution for the light volume.

In our shadow algorithm, we implement shadows by splatting the volume only once to generate per-pixel accurate shadows. The same splatting is used for both the viewer and the light source using sheet-based warping methods [18]. For each footprint, while adding its contribution to the sheet buffer as seen from the eye, we also add its contribution to a sheet shadow buffer as seen from the light source.

Here, we consider the case of a light source behind the viewer. In the image-aligned sheet-based splatting, the light passing through the front sheets will be attenuated and cause shadows on the back sheets along the light rays. This effect of front sheets on back sheets is shown in Figure 2.

For the opacity transfer function, we can use the same opacity transfer function for both the rendering and the shadow. The opacity with respect to the light source can also be accumulated using the formula:

For a front-to-back traversal:

\[
A_o = A_c + ((1 - A_c) * A_n)
\]

For a back-to-front traversal:

\[
A_o = ((1 - A_n) * A_c) + A_n
\]

During the rendering, when we calculate the illumination for a pixel at the current sheet, we determine the accumulated opacity for the pixel from the shadow buffer by mapping the pixel to the shadow buffer. The pixel at the current sheet is first transferred back to eye space, and it is then re-projected to the shadow buffer as seen from the light source (as shown in Figure 3). Here we take the orientation of the shadow buffer aligned with image plane. The shadow buffer can also be light aligned.

The pixel \((i, j)\) on the current sheet buffer can be mapped to the pixel \((i', j')\) on the shadow buffer using the following transformation:

\[
\begin{pmatrix}
  i' \\
  j'
\end{pmatrix} = M_2 M_1 \begin{pmatrix}
  i \\
  j
\end{pmatrix}
\]

where, \( M_1 \) is the matrix which transfers the pixel \((i, j)\) on the current sheet buffer to the point \(x\) in eye space, \( M_2 \) is the matrix which transfers the point \(x\) in eye space to the pixel \((i', j')\) on the shadow buffer.

Then the intensity of the light arriving at the pixel is:

\[
I(x) = (1.0 - \alpha(x)) * I_{\text{light}}
\]
where, $\alpha(x)$ is the accumulated opacity at the location $x$ in the shadow buffer, $I_{light}$ is the original intensity of the light source.

Now the illumination model becomes:

$$C(x) = C_{obj}(x) * (k_a I_a + k_d I_{light} * (1.0 - \alpha(x)) * (N(x) \cdot L(x))) + k_s I_{light} * (1.0 - \alpha(x)) * (E(x) \cdot R(x))^k$$

For a given point $x$, we get its $\alpha(x)$ by choosing its nearest pixel’s opacity value in the shadow buffer. For better shadow quality, we can also calculate its $\alpha(x)$ by interpolating the opacity values of its nearby pixels.

Compared to splatting without shadows, two more buffers are needed: a 2D shadow buffer to store the composited opacity from the light to the current sheet, and a 2D sheet shadow buffer to store the opacity caused by the current sheet from the transfer function with respect to the light. The sheet shadow buffer is composited into the shadow buffer and used for the next slice.

Using the above algorithm, we have implemented shadows for two different light sources: parallel lights and point lights.

Figure 4 shows the shadow of a robot which is a constructed volume geometry (CVG) model composed of cube and rectangular parallelepipeds. The shadow of the Olympic rings composed of torus primitives is shown in Figure 5. Figure 6 is a scene of a smoky room with a cube inside. Figure 7 shows a room scene including the robot, the Olympic rings and a smoke-like object constructed using a turbulence function.

Figure 8 is the scene for a HIPPI (High Potential Iron Protein) data set. The data set describes a one-electron orbital of a four-iron and eight-sulfur cluster found in many natural proteins. The data is the scalar value of the wave function ‘psi’ at each point. We render the data set using the absolute value of the data. By comparing the scenes with and without shadows, it’s obvious that shadows give us more spatial relationship information.

Our splatting algorithm has also been extended to support haptexures. Figure 9 shows the shadow of a haptexured object. Figure 10 shows the curved shadows on curved objects.

All these shadows show that this shadow algorithm can create accurate shadows with little additional storage requirements.
4 MULTIPLE LIGHT SOURCES

The shadow algorithm can be easily extended to multiple light sources by using multiple shadow buffers. Each light source has a shadow buffer. For each footprint, we add its contribution to multiple shadow buffers as seen from multiple light sources. Hence, we splat the footprint n+1 times for n light sources.

During the rendering, when we calculate the illumination of a pixel at the current sheet, we map the pixel to multiple shadow buffers and get multiple opacity values for the pixel. The intensity of the i\textsuperscript{th} light arriving at the pixel is:

\[ I_i(x) = (1.0 - \alpha_i(x)) \cdot (l_{\text{light}})_i \]

The illumination model becomes:

\[
C(x) = C_{\text{obj}}(x) \cdot k_a \cdot I_a + \\
\sum_i C_{\text{obj}}(x) \cdot k_d \cdot (I_{\text{light}})_i \cdot (1.0 - \alpha_i(x)) \cdot ((N(x) \cdot L_i(x))) + \\
\sum_i (k_i(I_{\text{light}})_i \cdot (1.0 - \alpha_i(x)) \cdot (E(x) \cdot R_i(x))_{x^k})
\]

This extension has one limitation: all the lights need to lie either behind the viewer or behind the volume, with respect to the viewer, so that we can render the scene from front to back or from back to front.

Figure 11 shows the shadow of a robot with two light sources. It can be seen that the intersection region of the two shadows with respect to the two lights is darker than the region that is only in the shadow of one light.
5 PROJECTIVE TEXTURED LIGHTS

Projective textures can be added for special effects. We use a light screen to get the effect of the “light window” or slide projector and map the light pattern to the scene. The range of the shadow buffer is determined by projecting the light screen to the shadow buffer plane. We give the light screen an initial image. So in the illumination model:

\[ C(x) = C_{\text{obj}}(x) \cdot (k_a I_a + k_d I_{\text{light}} \cdot (1.0 - \alpha(x))) \]

\[ (N(x) \cdot L(x)) \cdot k_I I_{\text{light}} \cdot (1.0 - \alpha(x)) \cdot (E(x) \cdot R(x)) \]

the intensity of the light \( I_{\text{light}}(x) \) should be treated as a vector (the color of the light). We calculate the R, G and B components separately. For example, the R component is calculated using the formula:

\[ R(x) = R_{\text{obj}}(x) \cdot (k_a I_a + k_d R_{\text{light}} \cdot (1.0 - \alpha(x))) \]

\[ (N(x) \cdot L(x)) \cdot k_R R_{\text{light}} \cdot (1.0 - \alpha(x)) \cdot (E(x) \cdot R(x)) \]

Similarly, the G and B components are calculated.

We warp the light pattern to a buffer, defining the initial distribution of the light intensity in the buffer. During the rendering, the corresponding values can be obtained from this buffer.

Figure 12 shows a scene where the light screen has a pattern of rings, while Figure 13 shows a scene where the light screen is an image of the logo of The Ohio State University. From the two figures, we can see the light window and the smooth transmission of the light intensity. Figure 14 shows the shadow on a cube, from which we can see how the shadow changes with the shape of the object. In Figure 15, a parallel area light with a grid texture casts the grid pattern on a CVG sphere composed of 2 spheres. It gives us some dimension information of the object in 3D space.

In the room scene (as shown in Figure 16), the effect of the “light window” is achieved, with no pattern added on the light. Figure 17 shows an image with light beams passing through a semi-transparent cube. We can see the light intensity is attenuated as the light traverses the cube.

6 ACCURACY ISSUES

One limitation of our algorithm for shadows using image-aligned sheet-based splatting is in dealing with light sources perpendicular to the eye vector. The image-aligned splatting makes it difficult to keep track of accurate opacities as seen from the perpendicular light source, especially for those slices with similar z-values with the light source (as seen in Figure 18).

We can solve this problem by non-image-aligned sheet-based splatting. Here, we calculate the half way vector between the eye vector and the light vector, then splat the volume in the direction of the half way vector. For the rendering, we warp the sheets to the image plane aligned with the eye; while for the shadow, we warp the sheets to the shadow buffer aligned with the light source (as shown in Figure 19). So it’s a single splatting followed by \( n+1 \) warps for \( n \) light sources. This splatting along the half way vector will not cause the popping problem.

Using this non-image-aligned splatting method, we generate shadows for a perpendicular light source: the cube is at (0,0,0), the eye at (0,0,100) and the light at (30,100,0) in the world space. The splatting is along the half way vector (0.3,1.0,1.0) and the sheets are warped to the image buffer and the shadow buffer. In this way, the light attenuation is accurately modeled and the shadow is generated (as shown in Figure 20).

7 CONCLUSIONS

In this paper, we have described an algorithm to model the light attenuation through a volume using the image-aligned sheet-based splatting. This algorithm uses the single splatting to model the light attenuation with respect to the light source and generate shadows. We need two additional 2D buffers to keep the accumulated opacity and the individual sheet opacity with respect to the light source. For the running time, the algorithm with shadows takes less than twice the time without shadows. This algorithm has the advantage of saving storage and running time.

We have used this algorithm to implement shadows for point lights and parallel lights. Projective textured lights are used to create images with special effects. Our work also includes the implementation of shadows with respect to multiple light sources, by keeping separate shadow buffers with respect to each light and getting the opacity value for each light at each pixel. Our future work is to extend this algorithm to deal with extended light sources and generate soft shadows with penumbra and umbra corresponding to the shape of the extended light.
Figure 12: A scene with the shadow for a light screen with ring pattern

Figure 13: A scene with the shadow for a light screen with an image of OSU logo

Figure 14: A cube with the shadow for a light screen with stripe pattern

Figure 15: A CVG sphere for a parallel area light with grid texture

Figure 16: A room scene with “light window”

Figure 17: A scene with beams of light that pass through the cube
8 ACKNOWLEDGEMENTS

We would like to thank the Department of Energy ASCI Program for generous support for this project. Additional support was provided through an NSF Career Award (#9876022).

References