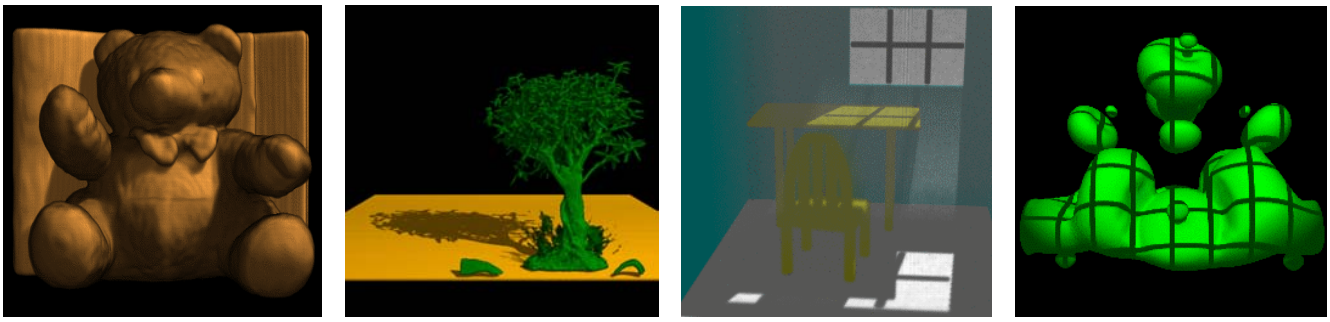
To appear in the Proceedings of the IEEE Visualization 2002 conference

Volumetric Shadows Using Splatting

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



(a) (b) (c) (d)

Figure 1. Some scenes with shadows. (a): Teddy bear. (b): Bonsai tree. (c): Room scene. (d): HIPIP with grid pattern.

Abstract

This paper describes an efficient algorithm to model the light attenuation due to a participating media with low albedo. The light attenuation is modeled using splatting volume renderer 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.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation – Display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Color, shading, shadowing and texture.
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 [19], 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 tremendously 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 [15] 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 [22] 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.

{zhangc, crawfis}@cis.ohio-state.edu, 2015 Neil Ave., 395 Dreese Lab, Columbus, OH 43210, USA

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 [21] and volumetric datasets [18]. 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 [9] 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 into how to deal with the light attenuation in volume rendering using splatting.

Nulkar and Mueller have implemented an algorithm to add shadows to volumetric scenes[16] using splatting. They use a two-stage splatting approach. In the first-stage, splatting is used to construct a three-dimensional 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.

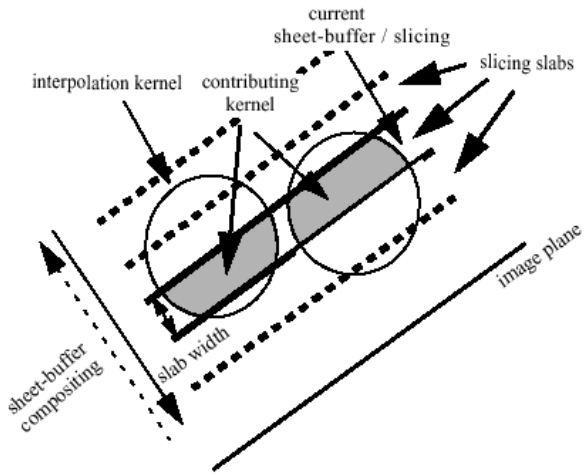


Figure 2. Image-aligned sheet-based splatting

In order to mitigate this problem, Westover proposed the sheet-buffer splatting method [20], 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. This improved splatting introduces a more substantial popping artifact when the orientation of the sheets changes. Mueller et. al. [14] eliminates this popping drawback by aligning the sheets to be parallel to the image plane. This splatting method (as shown in Figure 2) 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]. Splatting using post classification was proposed by Mueller et. al. [12] to generate images without blurry appearance.

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 [19]:

For a front-to-back traversal:

$$I_o = I_c + ((1 - A_c) * (I_n * A_n))$$
$$A_o = A_c + ((1 - A_c) * A_n) \quad (1)$$

For a back-to-front traversal:

$$I_o = ((1 - A_n) * I_c) + (I_n * A_n)$$
$$A_o = ((1 - A_n) * A_c) + A_n \quad (2)$$

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. I_o and A_o becomes I_c and A_c for the next sheet respectively.

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

$$C(x) = C_{obj}(x) * (k_a I_a + k_d I(x) * (N(x) \cdot L(x))) + k_s I(x) * (E(x) \cdot R(x))^{k_n} \quad (3)$$

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, corresponding to the fraction of the original light intensity that penetrates to the location $\boldsymbol{x}$ 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 functions of the location $\boldsymbol{x}$. $N(x)$ is calculated by estimating the gradient at each pixel using central differences [12]. The object color, $C_{obj}(x)$, and opacity, can be determined from a transfer function at each pixel. For the implementation of shadows, the main work is to determine the intensity of the light $I(x)$ arriving at each location $\boldsymbol{x}$. The intensity of the light is decreased due to light attenuation as light traverses the semi-transparent 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 is 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 [16] use a two-stage splatting algorithm to add shadows. They first splat the volume with respect to the light source using the image-aligned splatting algorithm and store the opacity values at each pixel for each sheet. Secondly, they splat the volume with respect to the eye to render the volume. They thus construct an entire light volume to store the intensity values after the first-stage splatting. The advantages of this approach include pre-processing the attenuation calculation for view-independent light volume. Accurate shadows are difficult to implement using this method, due to the limited resolution of the light volume.

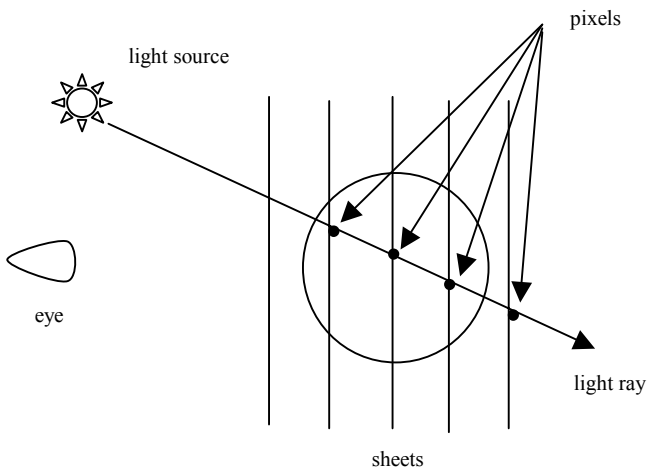


Figure 3. The light attenuation model
(Front sheets cause shadows to the back sheets along the light ray)

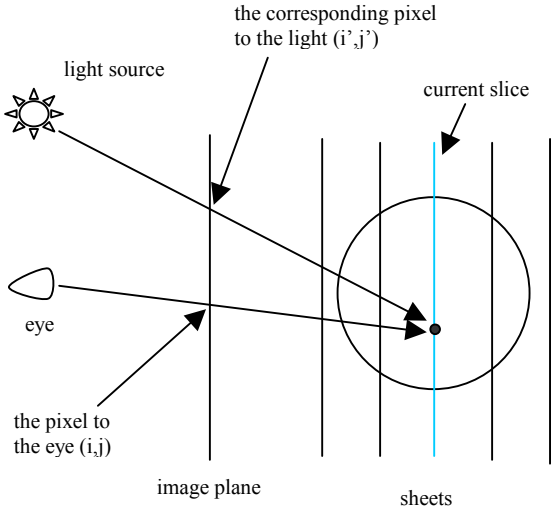


Figure 4. Determining the opacity value for the considered pixel

In our shadow algorithm, we implement shadows by traversing the volume only once to generate per-pixel accurate shadows. The same splatting algorithm is used for both the viewer and the light source. For each footprint, while adding its contribution to the sheet buffer as seen from the eye, we also add its contribution to a 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 3.

The opacity with respect to the light source can also be accumulated using the same formula (1) and (2) as before.

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 4). Here we take the orientation of the shadow buffer aligned with the image plane.

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:

$$\begin{pmatrix} i^{'} \\ j^{'} \end{pmatrix} = M_2 M_1^{-1} \begin{pmatrix} i \\ j \end{pmatrix} \tag{4}$$

where, M_1^{-1} is the matrix which transfers the pixel (i, j) on the current sheet buffer to the point $\boldsymbol{x}$ in eye space; M_2 is the matrix which transfers the point $\boldsymbol{x}$ in eye space to the pixel $(i^{'}, j^{'})$ on the shadow buffer.

Then, the intensity of the light arriving at the point $\boldsymbol{x}$ is:

$$I(x) = (1.0 - \alpha(x)) * I_{light} \tag{5}$$

where, $\alpha(x)$ is the accumulated opacity at $\boldsymbol{x}$, which is the value at $(i^{'}, j^{'})$ in the shadow buffer, I_{light} is the original intensity of the light source.

Now the illumination model becomes:

$$\begin{aligned} C(x) = C_{obj}(x) * (k_a I_a + \\ k_d I_{light} * (1.0 - \alpha(x)) * (N(x) \cdot L(x))) + \\ k_s I_{light} * (1.0 - \alpha(x)) * (E(x) \cdot R(x))^{k_n} \end{aligned} \tag{6}$$

For a given point $\boldsymbol{x}$, we get its $\alpha(x)$ by choosing its nearest pixel's opacity value in the shadow buffer. For better shadow quality, we can also interpolate the opacity values of 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 shadow sheet buffer to store the opacity caused by the current sheet from the transfer function with respect to the light. The shadow sheet buffer is composited into the shadow buffer and used for the next slice.

The shadow algorithm using the image-aligned sheet-based splatting is demonstrated with the pseudo code in Figure 5.

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

Figure 6 shows the shadow of a robot which is composed of cube primitives and rectangular parallelepiped primitives. The shadow of the Olympic rings composed of torus primitives is shown in Figure 7. Figure 8 is a scene of a smoky room with a volumetric cube inside. Figure 9 shows a room scene, which includes the robot, the Olympic rings and a smoke-like object constructed using a turbulence function.

Figure 10 is the HIPIP (high-potential iron-sulfur protein) dataset, which 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. Shadows provide spatial relationship information. Figure 11 shows the curved shadows on smooth objects.

```
1.  Transform each voxel to eye space;
2.  Bucket sort voxels according to the transformed z-values;
3.  Initialize occlusion map to zero opacity;
4.  Initialize the shadow buffer to zero;
5.  For each sheet in front-to-back order
6.      Initialize image sheet buffer;
7.      Initialize shadow sheet buffer;
8.      For each footprint
9.          Rasterize and add the footprint to the current image sheet
            buffer;
10.         Rasterize and add the footprint to the current shadow sheet
            buffer;
11.     End;
12.     Calculate the gradient for each pixel using central difference;
13.     Classify each pixel in the current image sheet buffer;
14.     Map pixel to the shadow buffer and get its opacity;
15.     Calculate the illumination to obtain the final color;
16.     Composite the current image sheet buffer to the frame buffer;
17.     Classify each pixel on current shadow sheet buffer and
          composite to accumulated shadow buffer;
18. End;
```

Figure 5. Pseudo code of shadow algorithm using the image-aligned sheet-based splatting

Figure 1(a) and 1(b) provides more results from our algorithm. Figure 12 is a uncBrain with shadows included.

The above images are generated using a front-to-back rendering. The room scene in Figure 1(c) is an example of back-to-front rendering: light comes into the room through the window from the back. A desk and a chair reside in the room filled with a light haze.

When light is attenuated, the running time is longer than the time without shadows, because footprint evaluation and shadow buffer compositing need to be done with respect to the light source. The algorithm with shadows takes less than twice the time without shadows. For the Bonsai tree (256*256*128) rendered to a 512*512 image, the running time with shadows is only about 56% slower, making the algorithm attractive for high-quality volume rendering.

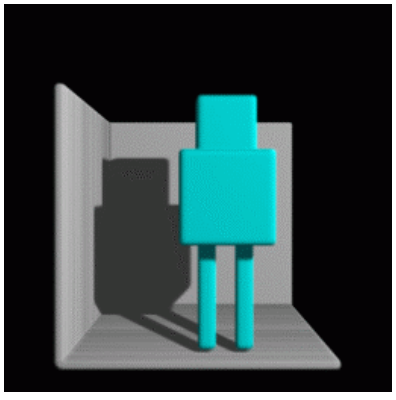


Figure 6. A robot with the shadow

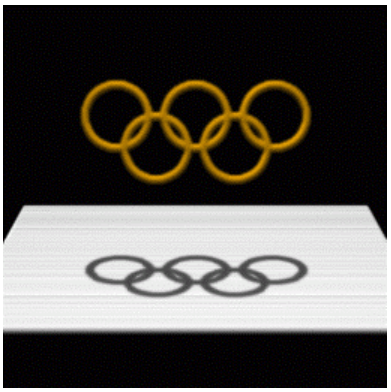


Figure 7. Shadows of Olympic rings

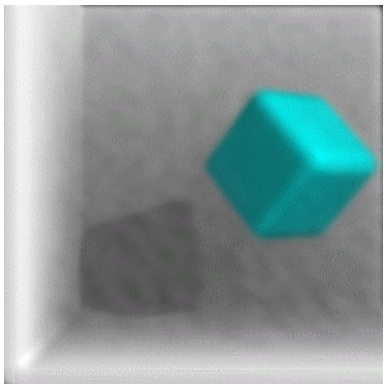


Figure 8. A smoky room with a cube inside

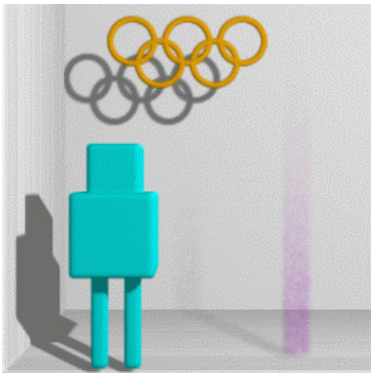


Figure 9. A room scene with shadows

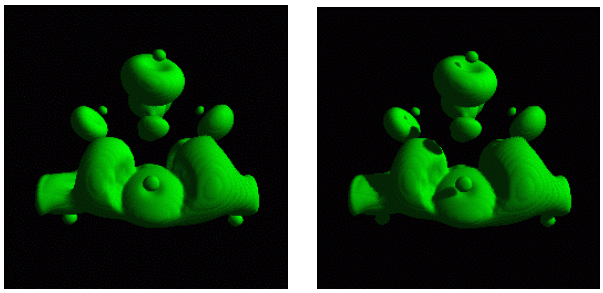


Figure 10. A scene of a HIPIP data set
(left: without shadow; right: with shadow)

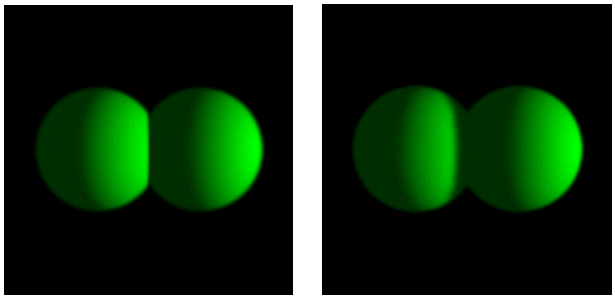


Figure 11. Curved shadows on smooth objects
(left: without shadow; right: with shadow)

Figure 12: uncBrain with shadows

4. MULTIPLE LIGHT SOURCES

The shadow algorithm can be easily extended to multiple light sources by using multiple shadow buffers. Each light source needs a separate accumulated 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^{th} light arriving at the point $\boldsymbol{x}$ corresponding to the pixel is thus:

$$I_i(x) = (1.0 - \alpha_i(x)) * (I_{light})_i \tag{7}$$

The illumination model becomes:

$$C(x) = C_{obj}(x) * k_a I_a + \sum_i (C_{obj}(x) * k_d (I_{light})_i * (1.0 - \alpha_i(x)) * (N(x) \cdot L_i(x))) + \sum_i (k_s (I_{light})_i * (1.0 - \alpha_i(x)) * (E(x) \cdot R_i(x))^{k_n}) \tag{8}$$

This extension has one limitation: all the lights need to lie either in front of the volume or behind the volume, with respect to the viewer. This is required in order to render the scene from front-to-back or from back-to-front.

Figure 13 shows the shadow of a robot with two light sources. It can be seen that the region shadowed by the two lights is darker than the region that is only in the shadow of one light.

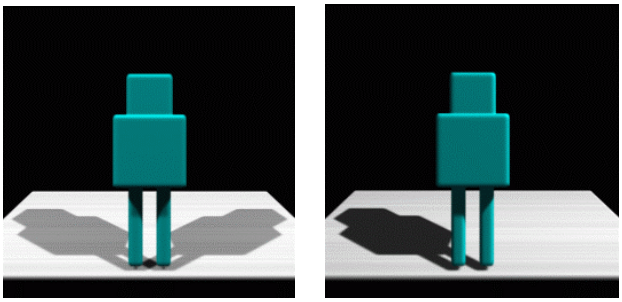


Figure 13. A robot with two light sources (left);
A robot with one light source (right)

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:

$$C(x) = C_{obj}(x) * (k_a I_a + k_d I_{light}(x) * (1.0 - \alpha(x)) * (N(x) \cdot L(x))) + k_s I_{light}(x) * (1.0 - \alpha(x)) * (E(x) \cdot R(x))^{k_n} \quad (9)$$

the intensity of the light $I_{light}(x)$ should be treated as a vector (the color of the light).

If the light is not aligned to the splat buffers, we need to warp the light pattern to the 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 14 shows a scene where the light screen has a pattern of red and green rings. A semi-transparent block is placed above a ground plane, which either sees the light or is partially attenuated by the block. The room scene in Figure 15 is lit by a light with an image of the logo of The Ohio State University. Shadows are generated by the robot and the rings which reside in the room.

In Figure 1(d), a parallel area light with a grid texture casts the grid pattern on a HIPIP scene. It gives us some dimension information of the object in 3D space.

Figure 16 shows images with light beams passing through a semi-transparent cube. Three light beams with red, green and blue colors enter the cube at the right top, traverse the cube and come out from the left bottom. The left image is without consideration of light attenuation, while the right one is with light attenuation. The light intensity exiting the cube is the same as the original intensity entering the cube in the left image, while the resulting light intensity is lower than the original light intensity due to attenuation as the light traverses the cube in the right image. In the middle area of the beams, the beam colors are partially blocked by front participating media.

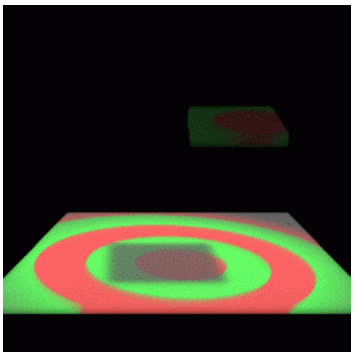


Figure 14. A scene with shadows for a light screen with ring pattern;

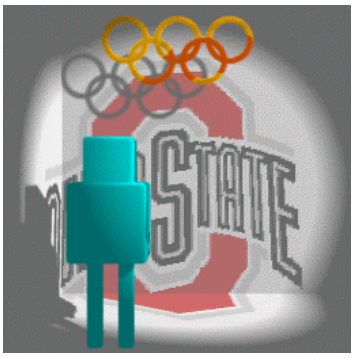


Figure 15: A room scene for a light screen with an image of OSU logo

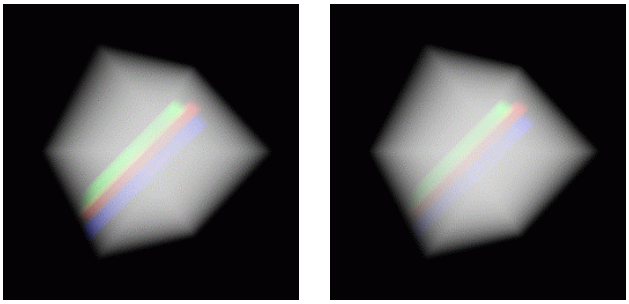


Figure 16. A scene with beams of light that pass through the cube (left: without attenuation; right: with attenuation)

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 as the light source (as seen in Figure 17).

We can solve this problem by using a new 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 each sheet, we add the footprint contribution to the image plane aligned with the eye for the rendering, and to the shadow buffer aligned with the light

source for the shadow (as shown in Figure 18). We use per-pixel classification. The pixel (*i,j*) at the current image buffer is first transferred back to the point ***x*** in eye space, and it is then projected to the shadow buffer aligned with the light source. The light intensity arriving at the point ***x*** is obtained from the accumulated opacity stored at the corresponding pixel (*i',j'*) on the shadow buffer, which has been calculated from front sheets. In this way, the light attenuation is accurately modeled. Kniss [7,8] recently has also proposed the idea of a half angle slice axis.

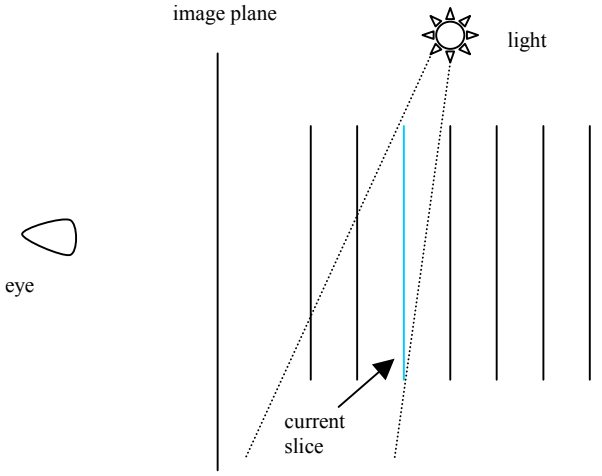


Figure 17. The problem of image-aligned splatting in dealing with perpendicular light sources

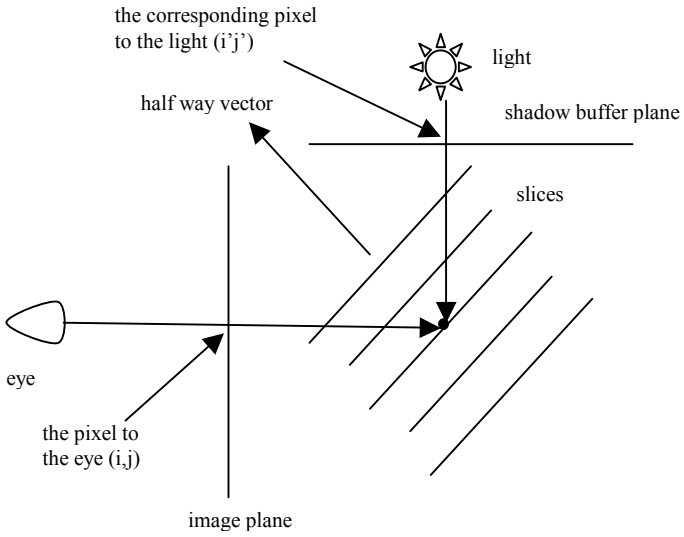


Figure 18. Non-image-aligned sheet-based splatting

Using this non-image-aligned splatting method, we generate shadows for a perpendicular light source (as shown in Figure 19): a semi-transparent cube is at (0, 0, 0), the eye at (0, 0, 120) and the light beam at (0, -1, 0) direction. The splatting is along the half way vector (0.0, 0.5, 0.5) and the footprints are individually warped to the image buffer and the shadow buffer. From the comparison of the two images in Figure 19, we can see that the light is attenuated.

The shadow algorithm for non-image-aligned splatting has been extended to generate shadows for multiple light sources. We use the average light vector of all the light sources to calculate the half way vector. Still, all the light sources should satisfy either $E \cdot L_i \geq 0$ or $E \cdot L_i \leq 0$ in the coordinate system defined by the half-way vector, so that the volume can be rendered either in front-to-back or back-to-front order. Here, *E* is the eye vector, and L_i are the light vectors. Figure 20 shows the Olympic rings with shadows implemented using this new non-image-aligned splatting for two light sources: one at (0, 320, 240), and the other at (0, 240, 320). The region shadowed by two lights is darker than the region that is only shadowed by one light.

In front-to-back or back-to-front rendering, if lights move with respect to the viewer, this non-image-aligned sheet-based splatting along the half way vector will not have the popping artifacts as mentioned for the sheet-based splatting in [20], since the splatting direction changes continuously with the eye vector and/or the light vectors. A consistent ray integration is generated with accurately reconstructed sheets. During the switch from front-to-back rendering to back-to-front rendering, there could be some popping problem for non-image-aligned splatting. Since we use small slice thickness, the popping artifacts are not discernable in our study.

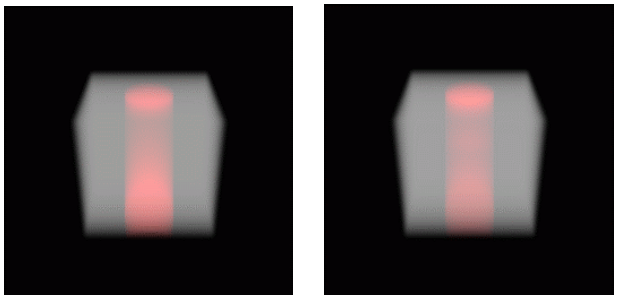


Figure 19. A scene with a beam of light that passes through the cube (left: without shadow; right: with shadow)

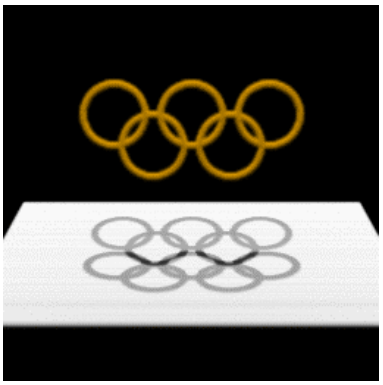


Figure 20: Shadows of Olympic rings for two light sources

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 models the light attenuation with respect to the light source and generates 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 or quantitative analysis. 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. Future work has progressed on extending this algorithm to deal with extended light sources to generate soft shadows with penumbra and umbra [24].

8. ACKNOWLEDGEMENTS

We would like to thank the NSF Career Award (#9876022) for support to this project and thank the University of Erlangen-Nuremberg for providing the Teddy bear and Bonsai tree datasets.

References

[1] P. Atherton, K. Weiler, D. Greenberg, "Polygon Shadow Generation", *Proc. SIGGRAPH'78*, pp. 275-281, 1978.

[2] U. Behrens and R. Ratering, "Adding Shadows to a Texture-based Volume Renderer", *1998 Symposium on Volume Visualization*, pp. 39-46, 1998.

[3] R. Crawfis, J. Huang, "High Quality Splatting and Volume Synthesis".

[4] F. Crow, "Shadow Algorithm for Computer Graphics", *Proc. SIGGRAPH'77*, pp. 242-248, 1977.

[5] F. Foley, A. Van Dam, S. Feiner, J. Huges, *Computer Graphics: Principles and practice*, Addison Wesley, 1996

[6] J. Huang, K. Mueller, N. Shareef, R. Crawfis, "FastSplats: Optimized Splatting on Rectilinear Grids", *Visualization'2000*, pp. 219-227, 2000.

[7] J. Kniss, G. Kindlmann, C. Hansen, "Multi-Dimensional Transfer Function for Interactive Volume Rendering", TVCG 2002.

[8] J. Kniss, S. Premoze, C. Hansen, D. Ebert, "Interactive Translucent Volume Rendering and Procedural Modeling", IEEE Visualization (2002 to appear).

[9] T. Lokovic, E. Veach, "Deep Shadow Map", *Proc. SIGGRAPH'2000*, 2000.

[10] M. Meissner, J. Huang, D. Bartz, K. Mueller, R. Crawfis, "A Practical evaluation of Popular Volume Rendering Algorithms", *2000 Symposium on Volume Rendering*, pp. 81-90, Salt Lake City, October 2000.

[11] K. Mueller, T. Moeller, J.E. Swan, R. Crawfis, N. Shareef, R. Yagel, "Splatting Errors and Antialiasing", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, No. 2, pp. 178-191, 1998.

[12] K. Mueller, T. Moeller, R. Crawfis, "Splatting Without the Blur", *Proc. Visuali-zation'99*, pp. 363-371, 1999.

[13] K. Mueller, N. Shareef, J. Huang, R. Crawfis, "High-quality Splatting on Rectilinear Grids with Efficient Culling of Occluded Voxels", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 5, No. 2, pp. 116-134, 1999.

[14] K. Mueller, R. Crawfis, "Eliminating Popping Artifacts in Sheet Buffer-based Splatting", *Proc. Visualization'98*, pp.239-245, 1998.

[15] T. Nishita, E. Nakamae, "An Algorithm for Half-Tone Representation of Three-Dimensional Objects", *Information Processing in Japan*, Vol. 14, pp. 93-99, 1974.

[16] M. Nulkar, K. Mueller, "Splatting With Shadows", *Volume Graphics 2001*.

[17] K. Perlin, E. M. Hoffert, "Hypertexture", Proc. SIGGRAPH'89, pp. 253-262, 1989.

[18] L. Sobierajski, A. Kaufman, "Volumetric Raytracing", *1994 Symposium on Volume Visualization*, pp. 11-18, 1994.

[19] L. Westover, "Interactive Volume Rendering", *Proceedings of Volume Visualization Workshop* (Chapel Hill, N.C., May 18-19), Department of Computer Science, University of North Carolina, Chapel Hill, N.C., 1989, pp. 9-16.

[20] L. Westover, "Footprint Evaluation for Volume Rendering", *Proc. SIGGRAPH'90*, pp. 367-376, 1990.

[21] T. Whitted, "An Improved Illumination for Shaded Display", *Communications of the ACM*, Vol. 23, No. 6, pp. 343-349, 1980.

[22] L. Williams, "Casting Curved Shadows on Curved Surfaces", *Proc. SIGGRAPH'78*, pp. 270-174, 1978.

[23] A. Woo, P. Poulin, A. Fournier, "A Survey of Shadow Algorithm", *IEEE Computer Graphics and Applications*, Vol. 10, No. 6, 1990.

[24] C. Zhang, "Implementation of Shadows Using Splatting", The Ohio State University Master thesis, 2002.