Shadows and Soft Shadows with Participating Media Using Splatting

Caixia Zhang and Roger Crawfis, *Member, IEEE Computer Society*

Abstract— **This paper describes an efficient algorithm to model the light attenuation due to a participating media with low albedo. Here, we consider the light attenuation along a ray, as well as the light attenuation emanating from a surface. The light attenuation is modeled using a splatting volume renderer for both the viewer and the light source. During the rendering, a 2D shadow buffer accumulates the light attenuation. We first summarize the basic shadow algorithm using splatting [30]. Then an extension of the basic shadow algorithm for projective textured light sources is described. The main part of this paper is an analytic soft shadow algorithm based on convolution techniques. We describe and discuss the soft shadow algorithm, and generate soft shadows, including umbra and penumbra, for extended light sources.**

Index Terms— **volume rendering, splatting, shadows, soft shadows, participating media, illumination.**

1 INTRODUCTION

VOLUME rendering is the display of datasets sampled in three dimensions. Splatting is one volume rendering algorithm, which can create high-quality images, and render efficiently in the case of a sparse dataset. The basic principles of a splatting algorithm 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 [6]. 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 and informative images. Earlier implementations of shadows focused on hard shadows, in which a value of 0 or 1 is multiplied with the light intensity. In volume rendering, as the light traverses the volume, the light intensity is continuously attenuated by the volumetric densities. This amounts to two separate volume renderings. Here, we investigate a new shadow algorithm that properly determines this light attenuation and generates shadows from volumetric datasets, using a splatting paradigm for volume rendering.

This paper is on the shadow algorithm using sheet-based splatting [19]. 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, background and previous work are reviewed and the motivation of this work is given. Section 3 summarizes the basic shadow algorithm using splatting [30]. In Section 4, we extend this approach for projective textured lights. This paper focuses on the soft shadow algorithm using splatting. Section 5 describes and discusses an analytic soft shadow algorithm for extended light sources using a convolution technique. The conclusions and future work are given in Section 6.

Manuscript received Nov. 15, 2002.

C. Zhang is with the Department of Computer and Information Science, The Ohio State University, 2015 Neil Ave., Columbus, OH 43210-1277. E-mail: zhangc@cis.ohio-state.edu.

R. Crawfis is with the Department of Computer and Information Science and the Advanced Computing Center for the Arts and Design, The Ohio State University, 2015 Neil, Ave., Columbus, OH 43210-1277. E-mail: crawfis@cis.ohio-state.edu.

2 PREVIOUS WORK

2.1 Shadow Algorithms

Earlier implementations of shadows focused on hard shadows from and onto strictly opaque objects. The algorithm by Crow [7] 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 surface or sample point is first checked to see whether or not it falls inside a shadow volume before it is illuminated by the light source. In the 2-pass hidden surface algorithm by Nishita and Nakamae [20] and Atherton et al. [1], the first pass transforms the image to the view of the light source, and decomposes the polygon into shadowed and unshadowed portions. A new set of polygons is created, each marked as either completely in shadow or visible from the light source. In the second pass, visibility 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 [28] 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

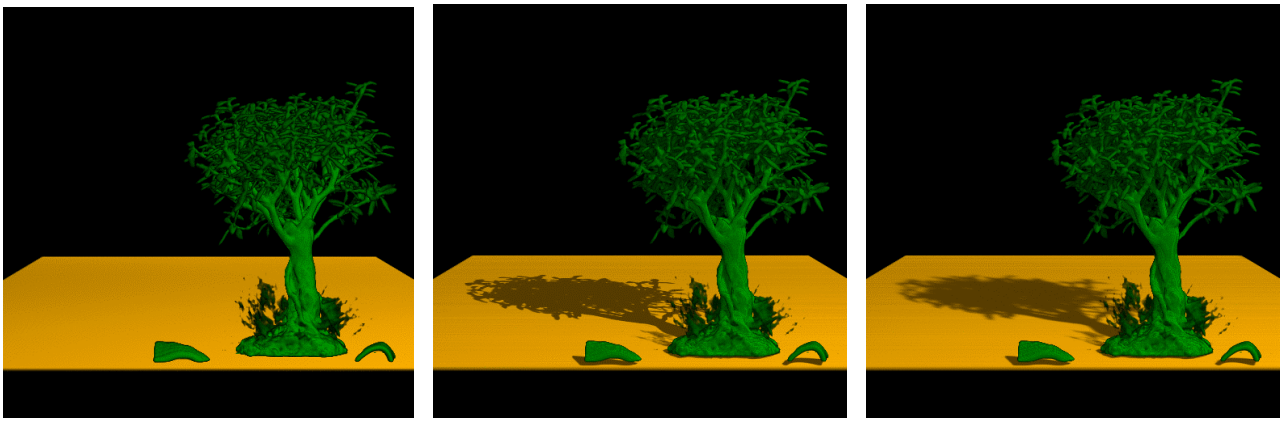


Fig. 1. Bonsai tree. (a) Without shadows. (b) With shadows. (c) With soft shadows.

primitives other than just polygons, but it has aliasing problems due to the discretized depth-map cells.

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 [27] and volumetric datasets [8, 11, 23]. To generate shadows for objects represented by densities, Kajiya et al. [11] store the contribution of each light source to the brightness of each point in space into a 3D array $I_i(x, y, z)$. Ebert and Parent [8] improve the calculation of the shadow table by storing the shadow values already calculated in a 3D table and calculating the shadow table values starting with the points closest to the light and proceeding to the points farthest from the light to avoid repeated calculations. Also, to make the approach feasible, they use a reduced-resolution shadow table. Thus, only a bilinear interpolation is needed to determine each value in the shadow table. However, the shadow algorithm using ray tracing is very costly computationally [15]. Here we investigate a new shadow algorithm that properly determines the light attenuation and generates the shadows for volumetric datasets, using a splatting paradigm for volume rendering.

Behrens, et al.[2] use 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. However, high performance is limited to parallel light sources. Also the pre-calculation of the light attenuation precludes post-classification. Lokovic and Veach [14] propose 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. 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 using splatting [21]. 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. Since the algorithm needs a 3D buffer to store the light volume, it has the problem of high storage and memory cost. Also, accurate shadows are difficult to implement using this method, due to the limited resolution of the light volume.

We investigate a new algorithm to implement shadows using splatting that requires only a 2D buffer for each light source [30]. Kniss, et al. [12] also utilize an off screen render buffer to accumulate the light attenuation. In this paper, we first give a summary of our basic shadow algorithm and Kniss et al.'s shadow algorithm. The main part of this paper is on the extensions of the basic shadow algorithm, and an analytic soft shadow algorithm using splatting.

2.2 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.

Mueller, et al. [19] eliminate these problems by aligning the sheets to be parallel to the image plane. This splatting method (as shown in Fig. 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

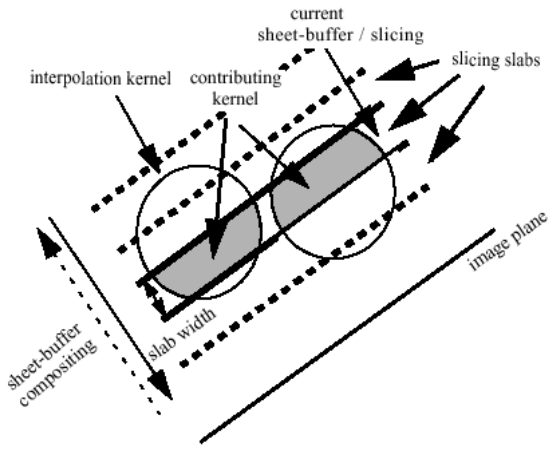


Fig. 2. Image-aligned sheet-based splatting.

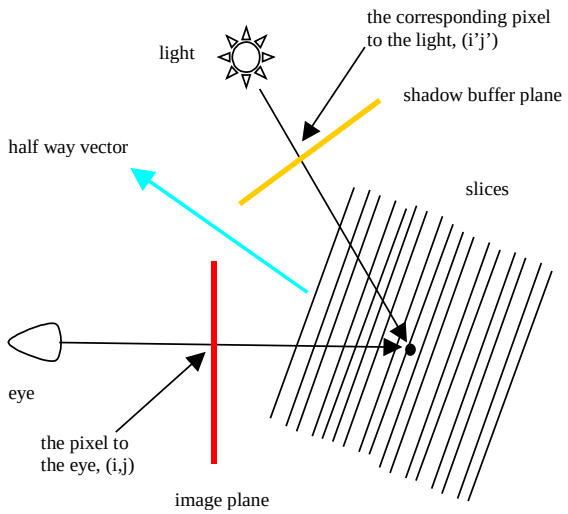


Fig. 3. Non-image-aligned sheet-based splatting.

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 [10].

Traditionally, splatting classifies and shades the voxels prior to projection. Projecting the fuzzy color balls leads to a blurry appearance of object edges. Splatting using post classification, which performs the color and opacity classification and shading process after the voxels have been projected onto the screen, was proposed by Mueller, et al. [17] to generate images with crisp edges and well-preserved surface details. In this paper, we use the post-classification to keep track of the per-pixel contribution to the light attenuation and generate per-pixel shadows.

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

3 BASIC SHADOW ALGORITHM

3.1 Summary of Shadow Algorithm 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, since the light intensity is continuously attenuated as the light traverses the volume. Our fundamental problem therefore is not determining whether a point is visible from the light, but rather to determine the light intensity arriving at the point being illuminated.

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. In the sheet-based splatting, the light passing through the front sheets will be attenuated and cause shadows on the back sheets along the light rays. At the current sheet, the light intensity is attenuated by all front sheets. If the light source resides behind the object, with respect to the viewer, then a back-to-front compositing order of the sheets is taken.

To generate shadows using splatting, we propose a new non-image-aligned sheet-based splatting to keep track of accurate light attenuation [30]. We first calculate the half way vector between the eye vector and the light vector. Rather than slicing the reconstruction kernels via planes parallel to the image plane, we chop the volume by slices perpendicular to the direction of the half way vector. We keep the image buffer aligned with the eye and the shadow buffer aligned with the light source (as shown in Fig. 3) to avoid sampling and resolution problems. This non-image-aligned sheet-based splatting along the half way vector will not have the popping artifacts as mentioned for the volume-aligned sheet-based splatting in [19], since the splatting direction changes continuously with the eye vector and/or the light vector. Therefore, a consistent ray integration is generated with accurately reconstructed sheets.

For high-quality rendering, we need to support per-pixel post classification and illumination. This implies the need to also support per-pixel shadowing. This is not possible with previous methods which store a 3D light buffer, such as Kajiya [11] or Nulkar [21]. During the rendering, when we calculate the illumination for a pixel at the current sheet, we look-up the accumulated opacity for the pixel from the shadow buffer by mapping the pixel to the shadow buffer. The pixel (i,j) at the current image buffer is first transferred back to the point $\boldsymbol{x}$ in the eye space using the current sheet's z-value. It is then projected to the pixel (i',j') at the shadow buffer, aligned with the light source (as shown in Fig. 3).

The light intensity arriving at the point $\boldsymbol{x}$ is calculated using the accumulated opacity stored at the corresponding pixel (i',j') on the shadow buffer:

$$I(x) = (1.0 - \alpha(x)) * I_{light} \quad (1)$$

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

This shadow buffer has accumulated the energy loss from all the sheets in front of the current sheet. In this way, the light attenuation is accurately modeled. For a given point $\boldsymbol{x}$, we get its $\alpha(x)$ by choosing its nearest pixel's opacity value in the shadow buffer, or using bilinear interpolation of the opacity values of nearby pixels in the shadow buffer. Since the shadow buffer is generated in lock-step with the image for each view, we can easily guarantee correct sampling of the shadow buffer.

Kniss, et al. [12] recently have proposed a similar idea of a half angle slice axis using 3D texture mapping for the volume rendering. Instead of creating a volumetric shadow map, they use an off screen render buffer to store the accumulated light attenuation. Also, they modify the slice axis to be the direction between the view and light directions. This allows the same slice to be rendered from both the eye and the light point of view. They use a p-buffer, an off screen render buffer, to store the accumulated light attenuation. Light is attenuated by simply accumulating the opacity for each sample using the over operator. Then, they copy the results to a texture, which is multiplied with the volume slice in the 3D volume renderer before it is blended into the frame buffer. In this way, they generate interactive shadows for volumes using 3D texture-mapping hardware.

Compared to image-aligned sheet-based splatting without shadows, two additional 2D buffers are needed in our shadow algorithm: a 2D shadow buffer to store the accumulated opacity from the light to the current sheet, and a working 2D sheet shadow buffer to which the current slab of voxel footprints are added into. The reason we need the sheet shadow buffer is that several voxel segments may contribute to a pixel within the current slab, and the contributions are evaluated voxels by voxels and added together to the sheet shadow buffer. In this way, the sheet shadow buffer keeps the opacity contribution of the current slab. Then, a per-pixel classification is applied to the sheet shadow buffer, which is then composited into the accumulated shadow buffer. Kniss, et al. [12] implement the shadows using graphics hardware. They use a p-buffer to store the accumulated light attenuation, equivalent to our 2D shadow buffer. The 3D texture rasterizer, reconstructs the function for the current slice, which is accumulated to the p-buffer using the over operator.

Our shadow algorithm using sheet-based splatting is demonstrated with the following pseudo code.

```
1.  Transform each voxel to the coordinate system having the half way
    vector as the z-axis;
2.  Bucket sort voxels according to the transformed z-values;
3.  Initialize opacity map to zero;
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 for;
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 it to the accumulated shadow buffer;
18. End for;
```

3.2 Shadow Results

Using the above algorithm, we have implemented shadows for two different types of light sources: parallel light sources and point light sources. More details on this algorithm can be found in [30]. Here, we will present some results before discussing our extensions to soft shadows.

The shadow of the rings composed of torus primitives is shown in Fig. 4. Notice how the per-pixel classification algorithm produces sharp shadows.

Fig. 5 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.

Our splatting algorithm has been extended to support hypertextures. Fig. 6 shows the shadow of a hypertextured object, which is constructed using Perlin's turbulence function [22].

Fig. 7 is the uncBrain with and without shadows. The insets are close-up renderings and precise curved shadows are generated. Again, notice that the shadows are calculated per-pixel rather than per-voxel.

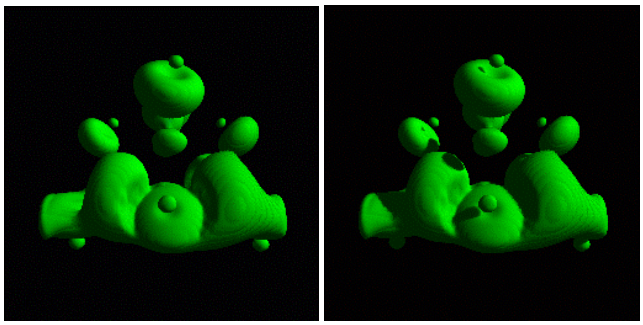
The above images are generated using a front-to-back rendering. If the light source is behind the objects, this algorithm proceeds as normal, but the compositing direction is changed from front-to-back to back-to-front. The room scene in Fig. 8 is an example of a back-to-front rendering: light comes into the room through the window from the back. A desk and a chair reside in the room filled with a light haze, and cast shadows.

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, providing a nice and efficient extension to our software based image-aligned sheet-based splatting software.

4 PROJECTIVE TEXTURED LIGHTS

Projective textures can be added for special effects. We use a

Fig. 4. Shadows of rings.



(a) (b)

Fig. 5. A scene of a HIPIP data set. (a) Without shadow. (b) With shadow.

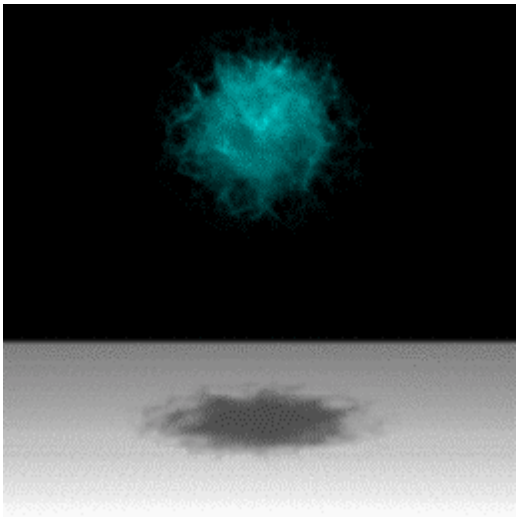
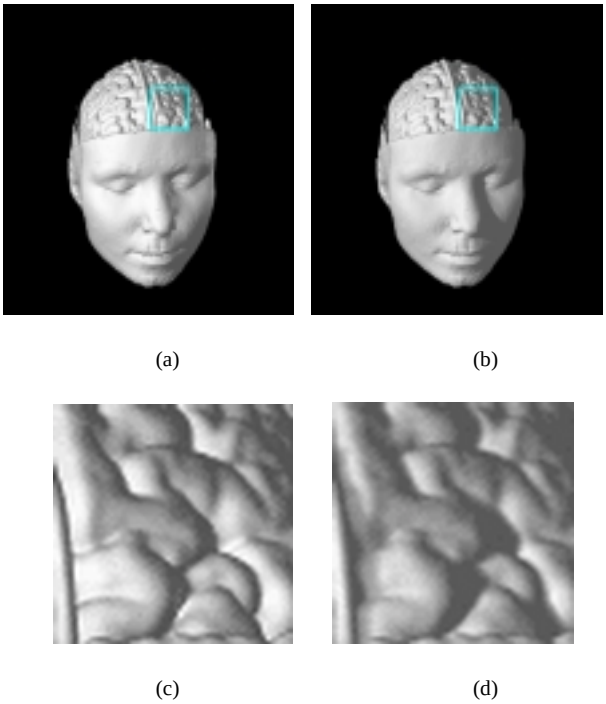


Fig. 6. A hypertextured object with the shadow.



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

Fig. 7. uncBrain with shadow. (a) Without shadow. (b) With shadow. (c) and (d) Close-up rendering of the specified patch.

Fig. 8. Room scene (an example of back-to-front rendering).

light screen to get the effect of a “light window” or slide projector cast into the scene. The range of the shadow buffer is determined by projecting the light screen to the shadow buffer plane. The light screen is then given an initial image.

The projective textured lights are modeled as in Fig. 9. Now, the light intensity at point $\boldsymbol{x}$ not only depends on the light attenuation, but also depends on the light color. Currently, we only support wavelength independent attenuation. This is a limitation of our current implementation, rather than the basic algorithm.

$$I(x) = I_{light} * light\_color(x) * (1.0 - \alpha(x)) \qquad (2)$$

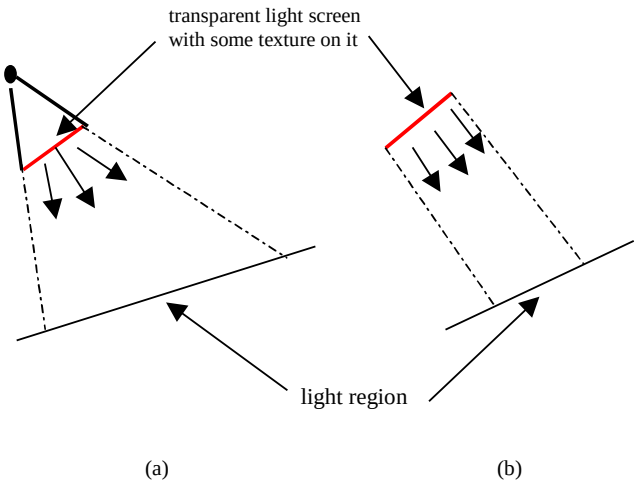


Fig. 9. A schematic of projective textured light models. (a) point light. (b) parallel light.

If the light screen, or projection direction, is not aligned to the shadow buffers, we need to warp the light pattern to the shadow buffer plane. This defines the initial distribution of the light intensity in the buffer. During the rendering, the corresponding values can be obtained from this buffer using a simple bilinear interpolation.

A room scene in Fig. 10 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 Fig. 11, a parallel area light source containing a grid texture casts the regular pattern onto the HIPIP dataset. By controlling the grid pattern, this gives us some dimensional information of the object.

Fig. 12 compares 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 image in Fig. 12(a) is without consideration of light attenuation, while the image in Fig. 12(b) is with light attenuation. The light intensity exiting the cube is the same as the original intensity entering the cube in the image in Fig. 12(a), while the resulting light intensity exiting the cube is diminished in the image in Fig. 12(b). Within the cube, the beam colors are partially blocked by the front participating media of the cube.

In Fig. 13, a light beam perpendicular to the eye vector passes through a translucent rectangular parallelepiped, which is rotated by 35°. The image in Fig. 13(a) is without

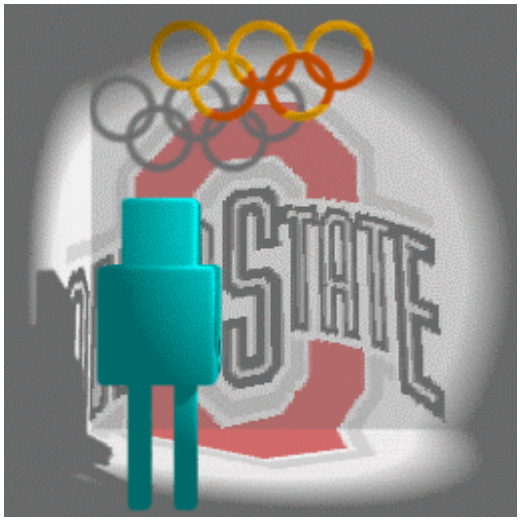


Fig. 10. A room scene for a light screen with an image of OSU logo.

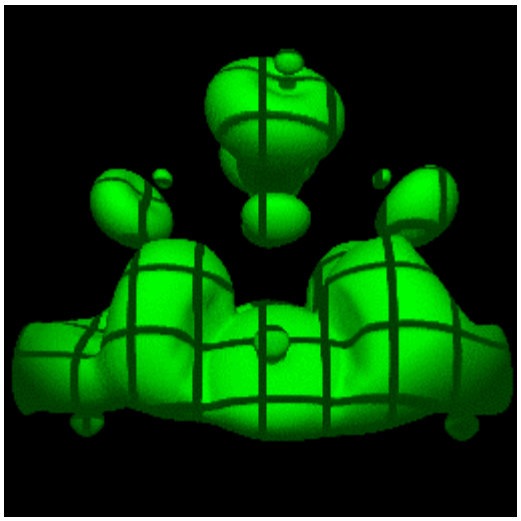
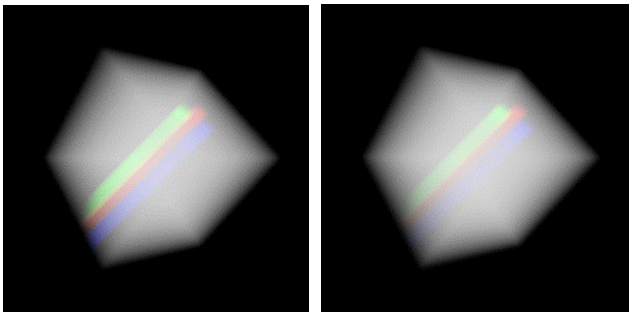
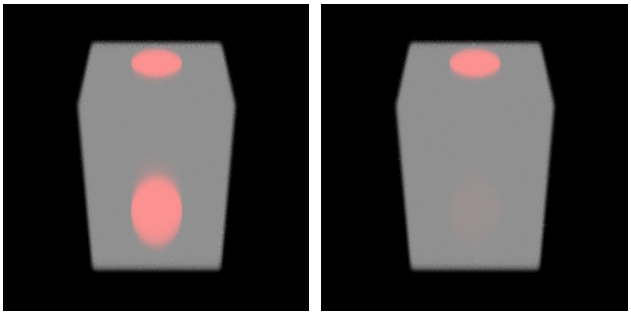


Fig. 11. HIPIP with grid pattern.



(a) (b)

Fig. 12. A scene with beams of light that pass through the cube. (a) Without attenuation. (b) With attenuation.



(a) (b)

Fig. 13. A scene with a beam of light that passes through the rectangular parallelepiped. (a) Without shadow. (b) With shadow.

attenuation, while the image in Fig. 13(b) considers the light attenuation. In the right image, most of the energy is attenuated, and only a little energy escapes from the rectangular parallelepiped.

5 SOFT SHADOWS FOR EXTENDED LIGHT SOURCES

5.1 Soft Shadow Algorithm

The generation of soft shadows is a difficult topic in computer graphics. Soft shadows include an umbra region, areas for which no part of the extended light source is visible, and a penumbra region, areas in which part of the extended light source is visible and part is hidden or occluded. The generation of soft shadows requires integrating the contributions of extended light sources on the illumination of objects.

In general, there are two main techniques to treat the extended light source: sampling techniques [3, 4] and analytical techniques [29]. The first technique is to sample the light source, and add the contributions of all the samples together to form a soft shadow. The sampling techniques are prone to image artifacts unless they are pushed to a stage where they become too expensive. In the second technique, the contribution of the extended light source is integrated using some form of numerical quadrature. These techniques typically require expensive data structures.

Soler and Sillion [24] use a convolution technique to calculate soft shadows that avoids both sampling artifacts and the building of expensive data structures to represent visibility. For the special case where the light source, the receiver and the occluder are all planar, and lie in parallel planes, they express the shadow as a convolution operation. For a general configuration, they construct a virtual light source, a virtual occluder and a virtual receiver, which are all planar and parallel to each other. They then compute the shadow for the virtual receiver using the constructed virtual geometry. Finally, they project the resulting shadow back to the actual receiver.

In this paper, we investigate an analytic method to generate soft shadows using the convolution technique. This soft shadow algorithm is based on the basic shadow algorithm discussed in section 3. Since we proceed in the volume rendering slice by slice, where all slices are parallel to each other, we can avoid some constraints and artifacts present in Soler's virtual occluders.

For an extended light source, we integrate over the light source to determine the contribution at a given point ***x***.

$$
\begin{aligned}
C(x) &= C_{obj} * k_a I_a \\
&+ \int_A (C_{obj} * k_d I_{light}(y) * (1.0 - \alpha(x,y)) * (N(x) \cdot L(x,y)))dy \\
&+ \int_A (k_s I_{light}(y) * (1.0 - \alpha(x,y)) * (E(x) \cdot R(x,y))^{k_n})dy
\end{aligned} \qquad (3)
$$

where, ***y*** is a point on the light source and A is the area of the extended light source.

At a given point ***x***, $I_{light}(y)$, $\alpha(x,y)$, $L(x,y)$ and $R(x,y)$ depend on the extended light source. We assume the light intensity is uniform across the extended light source. Also we denote the light vector from the center of the light to the point ***x*** as *L(x),* and approximate $N(x) \cdot L(x,y)$ by $N(x) \cdot L(x)$. Here, $N(x) \cdot L(x)$ can be considered to approximate the average of the $N(x) \cdot L(x,y)$ across the extended light source. This approximation is reasonable in cases where the light source is not very close to the objects. Similarly, we use $E(x) \cdot R(x)$ to approximate $E(x) \cdot R(x,y)$.

This leads to the following illumination model:

$$
\begin{aligned}
C(x) &= C_{obj} * k_a I_a \\
&+ C_{obj} * k_d I_{light} * (N(x) \cdot L(x)) \int_A (1.0 - \alpha(x,y))dy \\
&+ k_s I_{light} * (E(x) \cdot R(x))^{k_n} \int_A (1.0 - \alpha(x,y))dy
\end{aligned} \qquad (4)
$$

The term $\int_A (1.0 - \alpha(x,y))dy$ in the above equation is the integral of the light fraction arriving at the point ***x*** over the extended light source. We can also express the integral as $\int_A v(x,y)dy$, where *v(x,y)* is how much fraction of the light intensity at ***y*** on the light source arrives at point ***x*** on the receiver.

We calculate the term $\int_A (1.0 - \alpha(x,y))dy$ using convolutions. We use a box kernel, having a width determined by the penumbra region for the current slice. If *L* is the size of the extended light source, *Z* is the distance from the light to the occluder, and ΔZ is the distance from the occluder to the receiver (Fig. 14), then the width of the penumbra region is calculated by the formula:

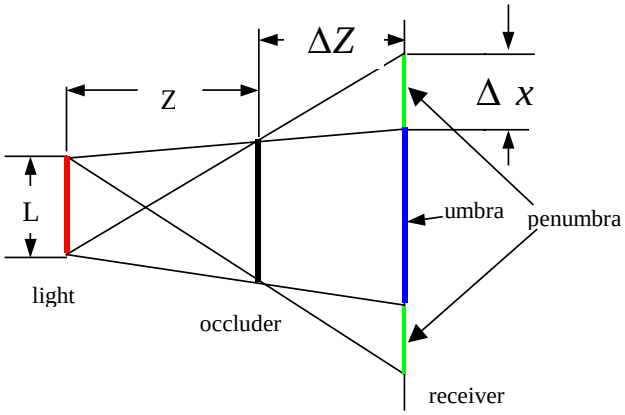


Fig. 14. A schematic of the light source, the occluder and the receiver.

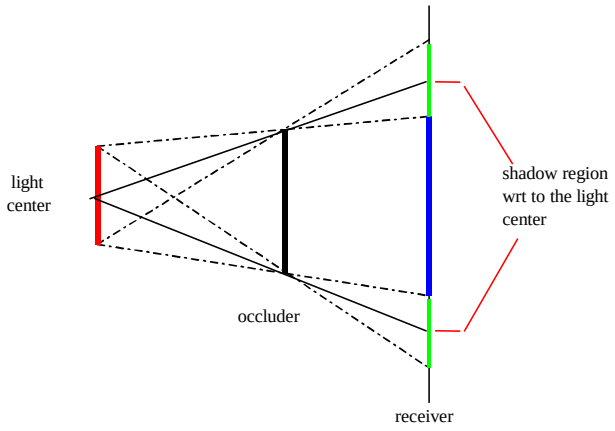


Fig. 15. A schematic of the shadow region with respect to the light source.

$$\Delta x = \frac{L * \Delta Z}{Z} \qquad (5)$$

We notice the Δx is constant across the receiver if the light source, the occluder, and the receiver are parallel, due to the geometrical properties of equivalent triangles. To achieve soft shadows, we can easily apply this mathematical formulation to analytically determine the penumbra region.

Using the shadow algorithm in section 3, we generate the shadow region with respect to the center of the extended light source (Fig. 15). The soft shadow, including both an umbra region and a penumbra region, is generated by convolving the above shadow region (as shown in Fig. 15) with a kernel size of Δx obtained from the above formulation. Referring to Fig. 15, the boundary of the shadow region with respect to the center of the virtual light is exactly in the middle of the penumbra region. We can derive it from the geometrical properties of the equivalent triangles. The shadow region is convolved using a box convolution kernel of size Δx. Thus, we get the exact penumbra region for the configuration in Fig. 15. The penumbra region depends on the size of the extended light, the distance from the light to the occluder, and the distance from the occluder to the receiver, as illustrated by equation (5). At a given point, the average shadow value of its neighborhood within the kernel is taken as its convolved shadow value.

In sheet-based splatting, we implement rendering slice by slice. At the current slice, all slices in front of it are occluders, and the current slice itself is the receiver. The contribution of the current slice should be composited into the accumulated shadow buffer to prepare for the next slice. Here, Z is the distance from the extended light source to the current sheet, and ΔZ is the distance between two adjacent sheets (Fig. 16). The penumbra region Δx is calculated for each slice using equation (5) and transformed to screen space. The contribution of the current sheet is obtained by projecting the occluder on the current sheet onto the shadow buffer with respect to the center of the extended light source. The accumulated shadow image, including the contribution of the current sheet, is taken

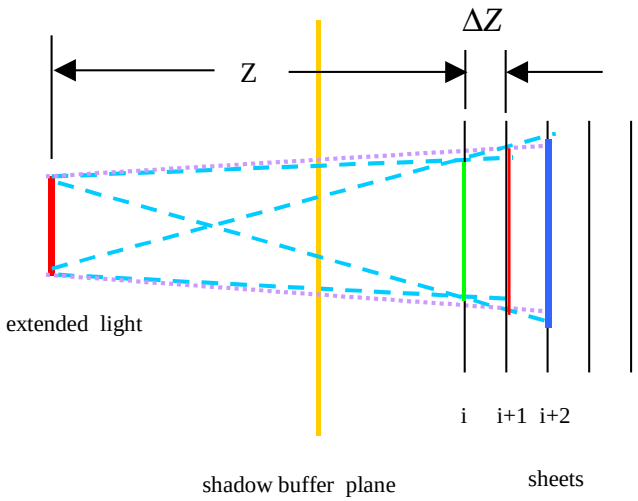


Fig. 16. Soft shadow algorithm in splatting.

to do the convolution and the convolved shadow values are stored in the accumulated shadow buffer to be used for the next slice.

At the sheet *i*, the shadow value obtained from the accumulated shadow buffer is the convolved shadow value, which has considered the contribution of the extended light source. The sheet shadow buffer contributed by the current sheet is composited to the accumulated shadow buffer, which is then convolved to prepare for the illumination at next sheet. We repeat the above convolution slice by slice (as shown in Fig. 16). At a pixel to be illuminated, we transfer it back to the eye space, then project it to the accumulated shadow buffer and obtain the light attenuation for it (Fig. 3). The obtained light intensity includes the contribution of the extended light source on the pixel.

Since we convolve the accumulated shadow buffer slice by slice, the contribution of a front sheet on the subsequent sheets

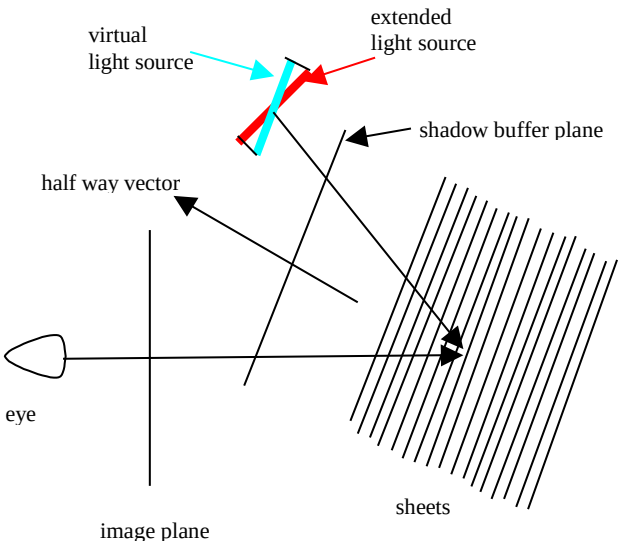


Fig. 17. Construction of a virtual light source.

is updated with the convolutions. For example, consider the contribution of the sheet *i* on the sheet *i+50*. The contribution of the sheet *i* is composited to the accumulated shadow buffer, and the shadow buffer has been convolved many times when the rendering proceeds to the sheet *i+50*. This satisfies equation (5), where the penumbra region caused by the sheet *i* on the sheet *i+50* depends on the distance ΔZ .

As discussed in section 3, we still slice the volume along the half-way vector. But, we keep the normal of the shadow buffer aligned with the half way vector, instead of the light vector, so that the shadow buffer is parallel to the slices (Fig. 17). This is required by equation (5).

To accomplish soft shadows, we add one extra step to the shadow algorithm given in Section 3.1. At each sheet, after compositing the sheet shadow buffer into the accumulated shadow buffer, we calculate the Δx . The accumulated shadow buffer is convolved using a kernel size of Δx to prepare for the next sheet.

5.2 Discussion of Soft Shadow Algorithm

In this section, we will discuss several factors which may affect the accuracy of the soft shadows.

(1) Constructing a virtual light

In our soft shadow algorithm, the occluder and the receiver are rendered slice by slice using sheet-based splatting. They have arbitrary geometry, but they are treated as parallel slices during the rendering. So, there is no need to approximate the occluder and the receiver in our soft shadow algorithm as in [24]. However, the above analytic soft shadow algorithm requires the extended light source to be parallel to the slices such that the penumbra and umbra regions can be calculated using equation (5). If the extended light source is not parallel to the slices, a virtual light source is created by using an orthogonal projection of the original light source (as shown in Fig. 17 and Fig. 18).

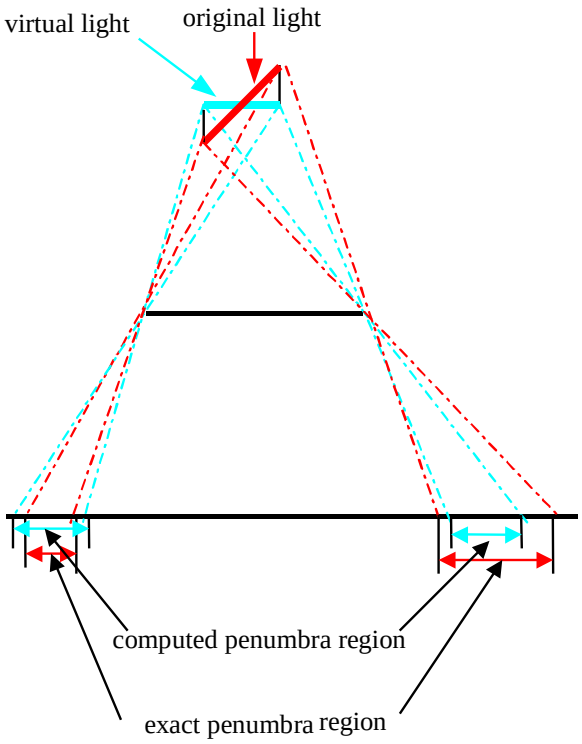


Fig. 18. Computed and exact penumbra regions.

If the angle between the normal of the extended light source and the volume slicing direction is small, the virtual light source generated by the above orthogonal projection will not introduce artifacts. If the angle is large (since the slicing direction is the half way vector between the eye vector and the light vector, the maximum degree is 45°), the virtual light will change the distribution of the penumbra region (as shown in Fig. 18). Here, we use a parallel planar occluder and receiver to analyze the approximation error. In Fig. 18, the penumbra region is smaller on the left side and bigger on the right side for the original extended light source, while the virtual light generates the same-size penumbra region on the both sides. In the cases where the light is small or not close to the occluder, and/or the receiver is not far from the occluder, the difference in the penumbra region will be small. A variable convolution kernel can be used to adjust the distribution of the penumbra region.

(2) Dealing with discretized shadow buffers

The above description is section 5.1 dealt with continuous convolution functions. The soft shadow algorithm is accurate mathematically. However, since we convolve the shadow buffer in screen space, we need to handle the discrete pixels. This implementation can introduce some artifacts.

From equation (5), we know the penumbra region Δx is calculated using $L * \Delta Z / Z$. In sheet-based splatting, ΔZ is the distance between two adjacent sheets. Since ΔZ is very small, Δx may also be very small. When Δx is transformed to the screen plane, it may be smaller than two pixels. In screen space, we accumulate Δx until it is greater than two pixels. Then, we do the convolution using the kernel size of the integer part of the Δx , and the rest part of the Δx is counted into the next accumulation. Therefore, we need to keep the last recent convolved sheet position, and use it to calculate the accumulated Δx . Here, Z is the distance from the light source to the last convolved sheet, and ΔZ is the distance from the last convolved sheet to next sheet following the current sheet.

The above convolution calculation can cause some inaccuracy problems. Since we keep only one accumulated shadow buffer, when the accumulated Δx is greater than two pixels, the contributions of all the sheets, between the last recent convolved sheet and the current sheet, are convolved. This problem is an implementation problem in dealing with discrete pixels, and it is not a problem mathematically. Using high-resolution shadow buffer can improve the accuracy. Also, obtaining shadow value using bilinear interpolation can

improve the accuracy.

(3) Calculating average light size

In our soft shadow algorithm, box kernels are used to do the convolution. Given a light source with an irregular shape, we first calculate its center and average size, then use these as the light center to which the occluder is projected and the light size L in equation (5) to calculate Δx.

The penumbra region depends on the average light size. The light shape is not considered. The effect of the light shape on the penumbra can be implemented by extending the convolution method by Soler and Sillion [24] to deal with the occluder with values in [0,1].

Compared with the convolution technique [24] by Soler and Sillion, our method has some advantages. Firstly, we don't need to approximate the occluder and the receiver. They are rendered slice by slice using splatting, so the occluders and the receivers are parallel slices during the rendering. We just need to create the virtual light. Secondly, we use splatting, a volume rendering method, so our soft shadow algorithm deals with the visibility in the range of [0,1], not just 0 or 1. Also we model the light attenuation slice by slice and we can generate self shadows.

Similar to the soft shadow algorithm in [24], the disadvantage of our soft shadow algorithm is that it is an approximate method. The orthogonal projection of light sources and the convolution of the shadow buffer using the accumulated Δx introduce some approximation.

5.3 Soft Shadow Results

The soft shadows, including umbra and penumbra, for extended light sources, are shown in Fig. 19–23 and Fig. 1(c), where the extended light source is a round area light. In the soft shadows of the rings (Fig. 19) and the robots (Fig. 20(b)), there is a penumbra region due to the extended light source. Compared to the hard shadows (Fig. 4 and Fig. 20(a)), the soft shadows have penumbra regions. The further the receiver is, the more blurred the shadow. For example, in Fig. 20(b), the shadows near the foot and the legs are hard, and the shadows of the body and the head become soft.

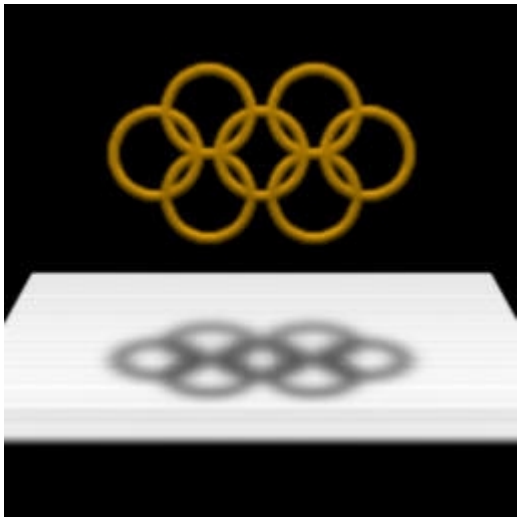


Fig. 19. Soft shadows of rings.

(a)

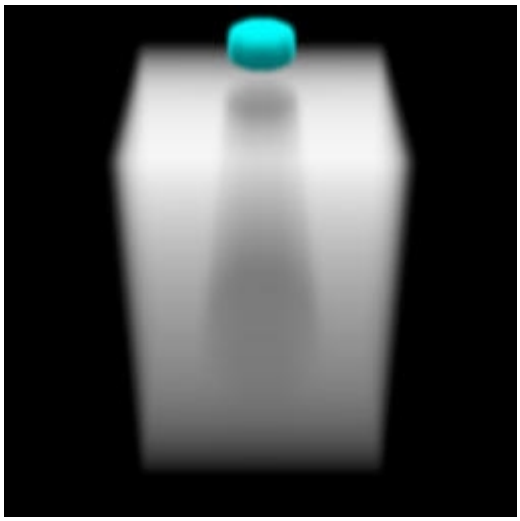
(b)

Fig. 20. (a) Soft shadow of robots. (b) Hard shadow of robots.

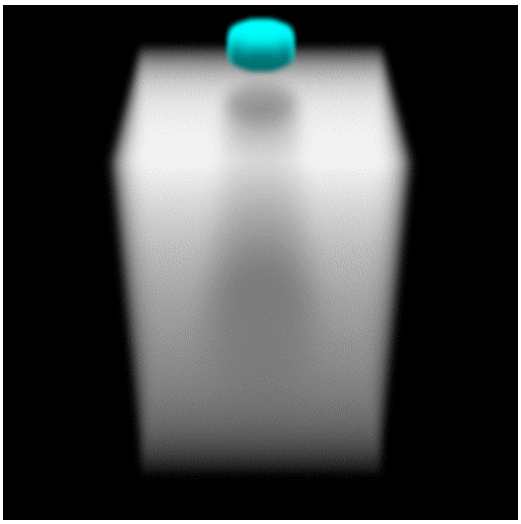
Fig. 21 shows that the shadow caused by the blue object passes through the translucent rectangular parallelepiped. The image in Fig. 21(a) is the hard shadow, while the image in Fig. 21(b) is the soft shadow. At the top entrance, the penumbra region is pretty small, so there is nearly no difference between the two images. As the shadow traverses the rectangular parallelepiped and comes out, the penumbra region becomes obvious for the soft shadow in Fig. 21(b), compared to the hard shadow in Fig. 21(a).

The soft shadow of the Bonsai tree is shown in Fig. 1(c). Compared with the hard shadow in Fig. 1(b), the Bonsai tree with soft shadows is more realistic. Also, the soft shadow of the hypertextured object is shown in Fig. 22.

In Fig. 23, a beam of light passes through a hole of an



(a)



(b)

Fig. 21. Soft shadow passing through the translucent rectangular parallelepiped. (a) Hard shadow. (b) Soft shadow.

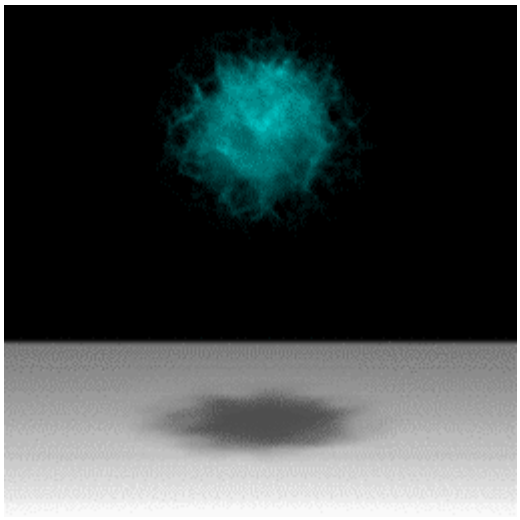


Fig. 22. Soft shadow of a hypertextured object.

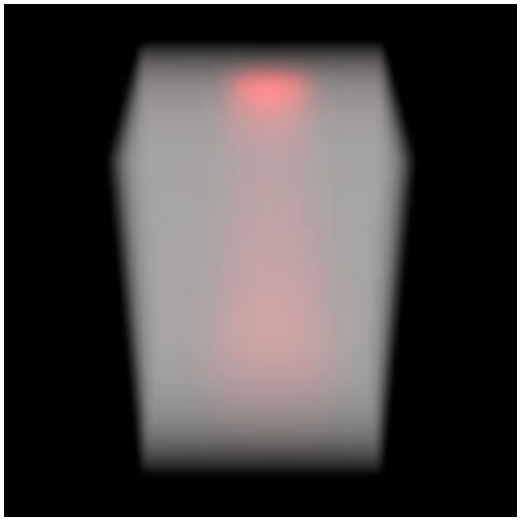


Fig. 23. A scene with a beam of light that passes through a rectangular parallelepiped, with soft shadows implemented.

opaque planar occluder (modeled in the light attenuation, but not displayed in the image), then traverses the translucent rectangular parallelepiped. In this image, soft shadows are implemented, so the light beam expands to the penumbra region. Also, due to the light attenuation as the light beam traverses the rectangular parallelepiped, the resulting light intensity is lower than the original light intensity.

Our method is a fast soft shadow calculation method. It keeps only one accumulated shadow buffer which stores the convolved shadow values. The contribution of the extended light source is integrated slice by slice using a convolution technique. For example, the running time of soft shadow of the robots takes 20% longer than the time for the same image with hard shadows.

6 CONCLUSIONS

In this paper, we have described an algorithm to model the light attenuation through a volume using the non-image-aligned sheet-based splatting. This algorithm models the light attenuation with respect to the light source and generates shadows. We need a 2D buffer to store the accumulated opacity. 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.

The basic shadow algorithm has been extended for projective textured light sources. Projective textured lights are used to create images with special effects or quantitative analysis. From some images lit by projective textured lights, we can see the light attenuation visually.

We propose an analytic soft shadow algorithm using splatting to deal with extended light sources and generate soft shadows with penumbra and umbra. Our soft shadow algorithm is a fast analytic method using a convolution technique. We discuss several factors which may affect the accuracy of the soft shadows. Also, we generate soft shadows

and compare them with the shadows generated using the basic shadow algorithm.

Our future work includes integrating polygonal rendering with our volume rendering with shadows, and extending the shadow algorithm for textured lights. We will study how the light texture changes using the convolution technique. In the future, we also plan to implement the shadow algorithm using hardware to improve the performance.

ACKNOWLEDGMENTS

Special thanks go to Rick Parent for his helpful reviews and proof-reading of the implementation. We also wish to thank the guest editor and the reviewers for their suggestions and careful reviews. We acknowledge the University of Erlangen-Nuremberg for the Bonsai tree dataset. This work was supported by the NSF Career Award (#9876022).

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] B. Brotman, N. Badler, "Generating Soft Shadow With a Depth Buffer Algorithm", IEEE CG&A, 4(10), pp.71-81, 1984.

[4] R. Cook, T. Porter, L. Carpenter, "Distributed Ray Tracing", Computer Graphics, 18(3), pp. 137-145, 1984.

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

[6] R. Crawfis, N. Max, "Texture Splats for 3D Scalar and Vector Field Visualization", Proc. Visualization'93 , pp. 261-266, 1993.

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

[8] D. S. Ebert, R. E. Parent, "Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-buffer Techniques", *Proc. SIGGRAPH'90*, pp. 357-366, 1990.

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

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

[11] J. T. Kajiya, B. P. Von Herzen, "Ray Tracing Volume Densities", *Proc. SIGGRAPH'84*, pp. 165-174, 1984.

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

[13] J. Kniss, S. Premoze, C. Hansen, D. Ebert, "Interactive Translucent Volume Rendering and Procedural Modeling", IEEE Visualization 2002.

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

[15] 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.

[16] 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.

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

[18] 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.

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

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

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

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

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

[24] C. Soler, F.X. Sillion, "Fast Calculation of Soft Shadow Textures Using Convolution", Proc. SIGGRAPH'98, pp. 321-332, 1998.

[25] 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.

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

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

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

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

[30] C. Zhang, R. Crawfis, "Volumetric Shadows Using Splatting", *Proc. Visualization 2002*, pp. 85-92, 2002.

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

Caixia Zhang received a MS degree in computer and information science from The Ohio State University in 2002. She is currently working on a PhD degree in computer and information science at The Ohio State University, where she also holds a graduate research appointment. Her research interests reside in volume graphics and scientific visualization. For more information see http://www.cis.ohio-state.edu/~zhangc.

Roger Crawfis received his BS in computer science and applied mathematics from Purdue University in 1984, and his MS and PhD degrees in computer science from the University of California, Davis, in 1989 and 1995, respectively. He is an associate professor at The Ohio State University. His research interests include scientific visualization, computer graphics, and volume rendering. Prior to joining OSU, Dr. Crawfis was the Graphics Technology Group Leader at the Lawrence Livermore National Laboratory, where he was in charge of coordinating several visualization projects for the past 12 years. He has published many research papers on scientific visualization and the volume rendering of scalar and vector fields.