# Deep Shadow Maps

Tom Lokovic Eric Veach

Pixar Animation Studios*

## Abstract

We introduce *deep shadow maps*, a technique that produces fast, high-quality shadows for primitives such as hair, fur, and smoke. Unlike traditional shadow maps, which store a single depth at each pixel, deep shadow maps store a representation of the fractional visibility through a pixel at all possible depths. Deep shadow maps have several advantages. First, they are prefiltered, which allows faster shadow lookups and much smaller memory footprints than regular shadow maps of similar quality. Second, they support shadows from partially transparent surfaces and volumetric objects such as fog. Third, they handle important cases of motion blur at no extra cost. The algorithm is simple to implement and can be added easily to existing renderers as an alternative to ordinary shadow maps.

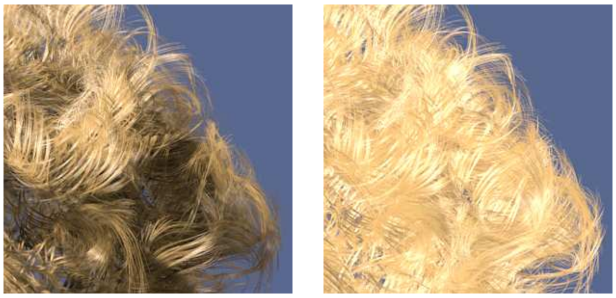


***Figure 1:*** *Hair rendered with and without self-shadowing.*

## 1 Introduction

Rendering hair, fur, and smoke is difficult because accurate self-shadowing is so important to their appearance [2]. To demonstrate this, Figure 1 shows a small patch of curly hair rendered both with and without shadows. Notice that the shadows cast by portions of the hair onto itself have a great influence on the overall illumination and apparent realism of the rendering.

Traditional shadow maps [11] need very high resolutions to capture this type of self-shadowing accurately. Many more depth samples must also be accessed during shadow lookups to compensate for the higher frequencies in the shadow map. This is especially obvious in animations, where inadequate sampling results in *sparkling* (a distracting artifact caused by rapidly changing noise patterns).

Furthermore, shadow maps cannot handle volumetric effects such as clouds or smoke. Traditional approaches such as integrating the atmospheric density along each shadow ray are very inefficient, simply because there can be a large number of shadow tests along each primary ray.

We propose a new type of shadow map to solve these problems, the *deep shadow map*. Rather than storing a single depth at each pixel, a deep shadow map stores a *fractional visibility function* (or simply *visibility function*) that records the approximate amount of light that passes through the pixel and penetrates to each depth. The visibility function takes into account not only the opacities of the surfaces and volume elements encountered, but also their coverage of the pixel's filter region. This allows deep shadow maps to accurately represent the partial attenuation that occurs as light passes through dense hair and fog.

Compared to ordinary shadow maps, deep shadows have the following advantages:

- They support semitransparent surfaces and volumetric primitives such as smoke.
- For high-quality shadows, they are smaller than equivalent shadow maps by an order of magnitude and are significantly faster to access.
- Unlike ordinary shadow maps, they support mip-mapping. This can dramatically reduce lookup costs when objects are viewed over a wide range of scales.

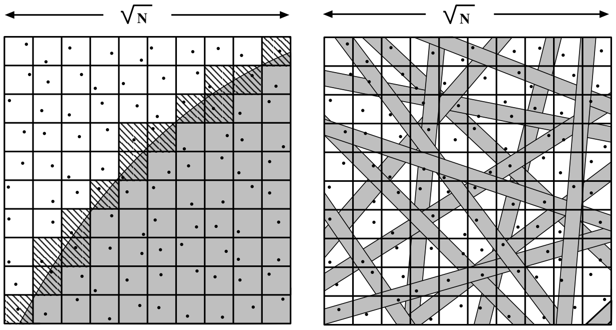
In addition, deep shadow maps can efficiently support high-quality motion blurred shadows. While this effect is also possible with ordinary shadow maps, the large filter widths required have made this technique quite expensive in the past.

In the following section, we discuss previous work and explain why traditional shadow maps are not appropriate for fine geometry such as hair and fur. Section 3 defines deep shadow maps and presents algorithms for creating and using them, while Section 4 describes their advantages and limitations. We then discuss some of the more practical implementation issues, and present the results from several experiments.

## 2 Background and Related Work

Traditional shadow maps [11] are generated by placing a camera (the *shadow camera*) at the light source origin such that the objects casting shadows are within the field of view. The result is a rectangular array of pixels where each pixel stores the depth of the closest visible surface. To determine whether a given point $P$ is in shadow, it is transformed into the coordinate system of the shadow camera and its depth is compared to the corresponding value from the shadow map. Higher-quality antialiased shadows are possible with *percentage closer filtering* [9], which examines depth samples within a given filter region and computes the fraction that are closer than a given depth $z$. This process relies heavily on stratified sampling [7], both in generating the original shadow map and in selecting a random subset of the depth samples for filtering.

*email: tdl@pixar.com, ericv@pixar.com




*(a)* *(b)*

***Figure 2:*** *Variance contributions to stratified sampling.* ***(a)*** *When a single silhouette edge passes through the filter region, $O(N^{1/2})$ samples contribute to the variance.* ***(b)*** *When the filter region is covered with fine geometry, all $N$ samples contribute to the variance, resulting in a much larger expected error.*

While shadow maps are very good for rendering shadows of large objects, they do not work as well for finely detailed geometry such as hair. This is because stratified sampling does not work very well when there are many discontinuities crossing the filter region [7]. As a result, many more samples are required in order to reduce noise artifacts such as sparkling to an acceptable level.

To understand why this is so, consider Figure 2a. Here we show an isolated silhouette edge crossing the filter region for a shadow lookup, evaluated using $N$ samples jittered over a $\sqrt{N} \times \sqrt{N}$ grid. The depth of each sample is obtained from the shadow map (whose resolution may be much finer than the sample grid). Using percentage closer filtering, each sample contributes either 0 or 1 to the average shadow value depending on the relative $z$ values of the shadow map and the test point $P$.

In this situation, observe that the only samples that contribute to the variance are the ones whose cells are crossed by the silhouette edge. There are only $O(N^{1/2})$ of these, and further analysis [7] shows that the expected error in this case is $O(N^{-3/4})$. Thus stratification yields much better results near large silhouette edges than unstratified Monte Carlo sampling, which has an expected error of $O(N^{-1/2})$.

In the case of hair or fur, however, the filter region is crossed by many silhouette edges (see Figure 2b). In this case, every one of the $N$ cells is crossed by an edge, and the corresponding expected error is $O(N^{-1/2})$. Thus in the case of very fine geometry, stratified sampling is no better than unstratified sampling.

These error bounds have a dramatic effect on the number of samples required to reduce noise below a given threshold. For example, to achieve an expected error of 1%, approximately $N = 140$ samples are needed near an isolated silhouette, while $N = 2500$ samples are required near a point that is 50% obscured by dense fur.[1] Furthermore, if the same amount of shadow detail is desired in both cases (*i.e.* the same filter size in world space), then the underlying shadow map resolution must be increased by the same factor. To gain any benefit from stratification, the shadow map would need to be fine enough to resolve the silhouettes of individual hairs, and the filter region small enough that only a few edges cross it. Since these conditions are rarely satisfied in practice, shadow maps for high-quality hair rendering are typically large and slow.

We briefly mention some other possible shadow techniques (the classic survey is [13]). Ray casting can generate accurate shadows, but on furry objects with millions of hairs it is too expensive in terms of time and memory. This is particularly true when hair receives shadows from fog or smoke, since ray marching is performed once for every hair sample. It is also difficult, other than by using an expensive area light source, to "soften" shadows for artistic purposes. In the case of standard shadow maps this can be achieved by simply increasing the filter width. Kajiya and Kay used ray-cast shadows on a volumetric hair representation [2], but their rendering times were relatively long.

Another possible approach to hair self-shadowing is to precompute the shadow density as a 3D texture. This technique has been used with some success for clouds (*e.g.* [3]) and for hybrid volume and surface rendering of medical datasets [5]. The main drawback of this approach is that 3D textures have a relatively coarse resolution, and in particular they have limited range and low accuracy in $z$ (which creates bias problems). A 3D texture with sufficient detail to capture accurate surface shadows of complex geometry would be prohibitively large.

*Multi-layer Z-buffers* [6] and *layered depth images* [10] store information at multiple depths per pixel, but are geared toward rendering opaque surfaces from new viewpoints rather than shadow evaluation. Keating and Max [4] apply multi-layer depth images to the problem of shadow penumbras, but their technique otherwise has the same limitations as ordinary shadow maps.

# 3 Deep Shadow Maps

## 3.1 Definition

A deep shadow map is a rectangular array of pixels in which every pixel stores a *visibility function*. Intuitively, a visibility function is defined by considering a beam of light that starts at the shadow camera origin and passes through the given pixel. The function value at a given depth is simply the fraction of the beam's initial power that penetrates to that depth. Note that the beam is not necessarily square; it can be shaped and weighted according to any desired pixel filter. Figure 3 gives several examples, showing how visibility functions can account for semitransparent surfaces, pixel coverage, and smoke. Each visibility function starts off with a value of 1, and decreases with depth as various types of blockers are encountered. If all light is blocked, the function drops off to a value of 0.

To make this definition more precise, consider a ray that starts at the shadow camera origin and passes through the point $(x, y)$ on the image plane. Some fraction of the light emitted along this ray will be attenuated by surfaces or by volumetric scattering and absorption. The fraction of light that penetrates to a given depth $z$ is known as the *transmittance* $\tau(x, y, z)$. We refer to $\tau$ as a *transmittance function* when we wish to consider the transmittance at a fixed image point $(x, y)$ as a function of $z$.

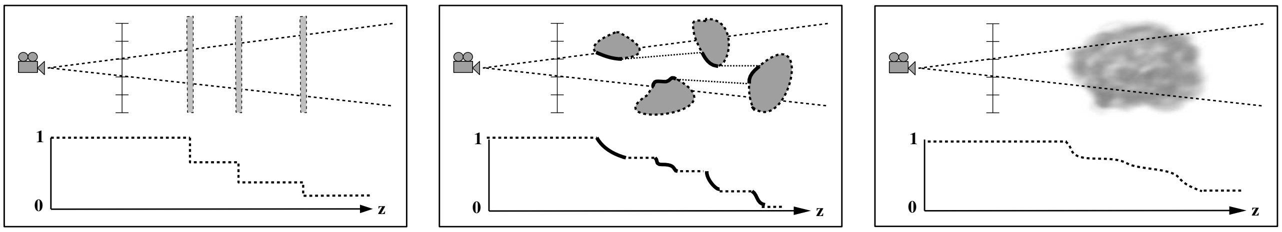
The visibility function for each pixel is now obtained by filtering the nearby transmittance functions and resampling at the pixel center. This is easier to understand if we restrict our attention to a particular depth $z$. The transmittance at every point in this $z$-plane is given by $\tau(x, y, z)$, and the visibility function $V_{i,j}$ for each pixel is obtained by filtering these values:

$$V_{i,j}(z) = \int_{-r}^{r} \int_{-r}^{r} f(s, t)\, \tau(i + \tfrac{1}{2} - s, j + \tfrac{1}{2} - t, z)\, ds\, dt\ ,$$

where $(i+\frac{1}{2}, j+\frac{1}{2})$ is the pixel center, $f$ is the desired bandlimiting pixel filter (centered around the origin), and $r$ is the filter radius. This definition is similar to ordinary image filtering, except that it applies to every $z$ value separately.

Notice that visibility functions are closely related to the *alpha channels* used for image compositing. The alpha channel of an im-

[1] These estimates assume that the silhouette edge is horizontal and has a random vertical position, in which case the constant factors for the expected error are $1/\sqrt{6}$ and $1/2$ respectively.




*(a) A stack of semitransparent objects* *(b) Partial coverage by opaque blockers* *(c) Volume attenuation due to smoke*

***Figure 3:*** *Visibility functions in flatland. Each diagram shows a beam of light that starts at the shadow camera origin (*i.e. *the light source) and passes through a single pixel of the deep shadow map, accompanied by that pixel's visibility function.* ***(a)*** *The beam's power is reduced as it passes through consecutive semitransparent surfaces.* ***(b)*** *The blockers are opaque, but each covers only part of the pixel's area; the emphasized segments of the function correspond to visible portions of the blockers.* ***(c)*** *Passage through smoke reduces the beam's power in a more continuous manner.*

age accounts for attenuation due to both semitransparent surfaces and partial coverage of pixels, but is equal to the fraction of light blocked rather than the fraction of light transmitted. The alpha channel also stores just a single value per pixel, corresponding to the light blocked at the plane $z = \infty$. Thus the relationship between visibility functions and alpha channels can be expressed as:

$$\alpha_{i,j} = 1 - V_{i,j}(\infty) .$$

A deep shadow map is equivalent to computing the approximate value of $1 - \alpha$ at all depths, and storing the result as a function of $z$. In this way each pixel contains the combined attenuation and coverage information for every depth.

## 3.2 Sampling

In this section we describe how deep shadow maps can be generated using the facilities of any standard renderer. Our strategy is similar to ordinary image sampling. To generate a deep shadow map, we select a set of sample points across the shadow camera's image plane (for example, 16 samples per pixel on a jittered grid). For each sample point we determine the corresponding transmittance function, which describes the light falloff along a particular primary ray. Finally, the visibility function for each pixel is computed by taking a weighted combination of the transmittance functions at nearby sample points.

We first describe how to compute a single transmittance function. Given an image point $(x, y)$, we compute the surfaces and volume elements intersected by the corresponding primary ray. The surface intersections can be found using either a ray tracer or an ordinary scan conversion renderer, and we assume that the properties of volumetric objects can be evaluated at any points desired. The transmittance function at the point $(x, y)$ can then be expressed as the product of a *surface transmittance function* $\tau^s$ and a *volume transmittance function* $\tau^v$, as described below.

Surface transmittance is estimated using all of the surface intersections along the primary ray at $(x, y)$. Each *surface hit* has a depth value $z_i^s$ and an opacity $O_i$. These are composited in the usual way, starting with a transparency of 1 and multiplying by $1 - O_i$ at each surface hit, to yield a piecewise constant function $\tau^s$ (see Figure 4a). Notice that each surface hit generates two vertices with the same $z$ value, in order to represent the discontinuous steps as a piecewise linear curve.[2] The "vertices" at $z = 0$ and $z = \infty$ are represented implicitly and are not part of the output.

[2]The wasted space in this representation is removed during the compression phase, which approximates the average of many discrete steps with a single linear segment. A true "step" in the compressed output function would occur only for a surface exactly parallel to the $xy$-plane.

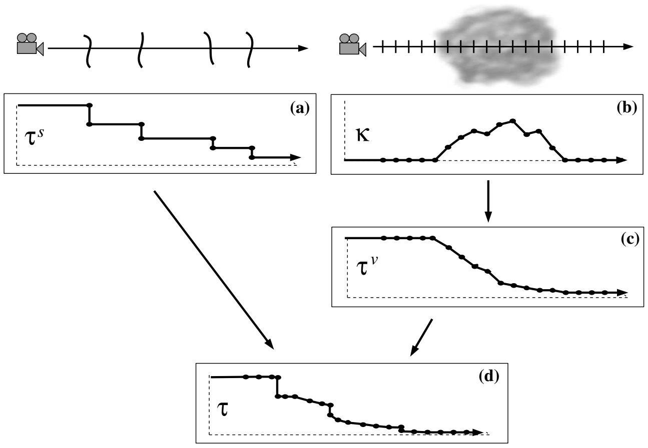



***Figure 4:*** *Constructing a transmittance function.* ***(a)*** *The object intersections along a given ray yield the surface transmittance function $\tau^s$, which has a discontinuity at the depth of each surface.* ***(b)*** *The extinction function $\kappa$ is obtained by sampling the atmospheric density at regular intervals along the ray.* ***(c)*** *The extinction function is integrated and exponentiated to yield the volume transmittance $\tau^v$.* ***(d)*** *The surface transmittance and volume transmittance are multiplied to obtain the final transmittance function $\tau$ for each ray.*

To estimate the volume transmittance, we sample the atmospheric density at regular intervals along the ray. Each *volume sample* has a depth value $z_i^v$ and an *extinction coefficient* $\kappa_i$ that measures the light falloff per unit distance along the ray. We linearly interpolate between these samples to yield the *extinction function* $\kappa$ (see Figure 4b). The fraction of light that penetrates to a given depth $z$ is then given by the formula[3]

$$\tau^v(z) = \exp(-\textstyle\int_0^z \kappa(z')\, dz') .$$

Since this function is not piecewise linear, we approximate it by evaluating the transmittance at each vertex of the extinction function and linearly interpolating. We do this by computing the transmittance of each linear segment as

$$T_i = \exp(-(z_{i+1}^v - z_i^v)(\kappa_{i+1} + \kappa_i)/2)$$

and compositing as we did for the surface transparencies, except that we interpolate between vertices rather than forcing discrete

[3]If the shadow camera is not orthographic, a correction factor is needed for each ray to account for the relationship between depth and distance.

steps. This yields the volume transmittance function $\tau^v$ (see Figure 4c). We then merge the surface and volume components by multiplying them:

$$\tau(z) = \tau^s(z)\,\tau^v(z)$$

(see Figure 4d). Since this function is again not piecewise linear, we evaluate it at the combined vertices of $\tau^s$ and $\tau^v$ and interpolate linearly between them.

Finally, we describe how the transmittance functions are combined to yield the visibility function $V_{i,j}$ for an entire pixel. At each depth $z$ the nearby transmittance functions are filtered just like ordinary image samples:

$$V_{i,j}(z) = \sum_{k=1}^{n} w_k\,\tau_k(z)\;, \qquad (1)$$

where $n$ is the number of transmittance functions within the filter radius around $(i+\frac{1}{2}, j+\frac{1}{2})$, and $w_k$ is the normalized filter weight for each corresponding sample point $(x_k, y_k)$. The result is a piecewise linear function that has approximately $n$ times as many vertices as the transmittance functions do. This function takes into account not only the light attenuation due to semitransparent surfaces and fog, but also the fractional coverage of these features.

## 3.3 Compression

Visibility functions sampled in this way may have a large number of vertices, depending on the filter radius and the number of samples per shadow pixel. Fortunately these functions are generally quite smooth, making them easily compressed. The compressed functions are stored as an array of floating-point pairs, each containing a $z$ value and a fractional visibility $V$.

It is very important that the compression method preserve the $z$ values of important features, since even small errors in $z$ can lead to undesirable self-shadowing artifacts. The method must also be appropriate for the unbounded domain $z \in [0, \infty)$. These facts preclude the use of compression methods based on a fixed or hierarchical set of basis functions (*e.g.* a wavelet basis). It also implies that the $L_1$ and $L_2$ error metrics are unsuitable, since visibility errors are important even if they occur over a very small range of $z$ values. Instead we use the $L_\infty$ error metric (maximum error), and compress functions using the simple greedy algorithm described below.

Given a visibility function $V$ and an error tolerance $\epsilon$ (see Figure 5a), our algorithm outputs a new visibility function $V'$ such that

$$\left|V'(z) - V(z)\right| \le \epsilon \qquad \text{for all } z$$

and where $V'$ typically has a much smaller number of control points (Figure 5d). The main feature of the algorithm is that it is incremental: It reads and writes control points one at a time in increasing $z$ order, and requires only a constant amount of state information.

The basic idea is that at each step, the algorithm draws the longest possible line segment that stays within the error bounds (similar to hitting a ball through as many wickets as possible in a game of croquet). The origin of the current segment is fixed, and we only need to choose the direction and length of the segment. To simplify the implementation, we restrict the output $z$ values to be a subset of the input $z$ values.

Let the origin of the current output segment be $(z'_i, V'_i)$. At every step we maintain the range of permissible slopes $[m_{lo}, m_{hi}]$ for the segment. Each new control point $(z_j, V_j)$ of the input function $V$ imposes a constraint on the current slope range by forcing the segment to pass through the *target window* defined by the wedge from the segment origin to the two points $(z_j, V_j \pm \epsilon)$ (see Figure 5b). The current slope range is initialized to $[-\infty, \infty]$, and is intersected

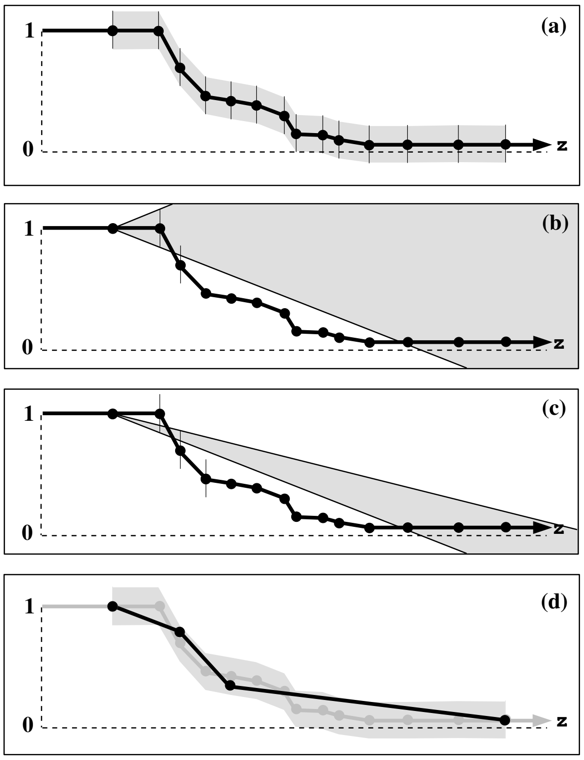



***Figure 5:*** *Our compression algorithm.* ***(a)*** *A piecewise linear curve and an illustration of its error bound.* ***(b)*** *Each input vertex defines a* target window *that constrains the slope of the next output segment.* ***(c)*** *The current slope range is intersected with each target window until it would become empty.* ***(d)*** *The output segment is extended to the current z value with a slope equal to the midpoint of the current slope range, and this process is repeated.*

with each target window in succession until further progress would make it empty (see Figure 5c). We then output the line segment with slope $(m_{lo} + m_{hi})/2$ terminating at the $z$ value of the last control point visited. The endpoint of this segment becomes the origin of the next segment, and the entire process is repeated. Note that the midpoint slope rule attempts to center each segment within the allowable error bounds.

This algorithm is fast, simple to implement, and requires constant storage. Slightly better approximations could be obtained by doing a least-squares fit once the $z$ values of the control points have been chosen. However, the basic algorithm satisfies the given error criteria and generates very good approximations in practice.

## 3.4 Lookups

Like textures, deep shadows are accessed by applying a reconstruction and resampling filter to a rectangular array of pixel values. In our case the pixel values are obtained by evaluating the visibility functions at a constant depth $z$. Given a point $(x, y, z)$ at which to perform the lookup and a two-dimensional filter kernel $f$, the filtered shadow value is given by

$$V(x, y, z) \;=\; \frac{\sum_{i,j} w_{i,j} V_{i,j}(z)}{\sum_{i,j} w_{i,j}}$$

where $w_{i,j} = f(i+\frac{1}{2}-x, j+\frac{1}{2}-y)$ is the filter weight for pixel $(i, j)$, and the sum is over all pixels within the filter radius.

Evaluating each visibility function requires a search through its data points to determine which segment contains the given $z$ value. This can be done using a linear or binary search, depending on the number of data points. In our implementation, we take advantage of the fact that many shadow lookups are often done at nearby $z$ values by storing a pointer with each pixel to the most recently accessed segment. On each lookup we search linearly either forward or backward from this position in order to reduce the average cost of visibility function evaluations.

# 4 Discussion

One of the main advantages of deep shadows over regular shadow maps is that they support prefiltering. Each deep shadow map pixel summarizes many individual depth samples in such a way that evaluating the visibility function at a given $z$ value is equivalent to percentage closer filtering all of the depth samples within the pixel's filter radius (to within the tolerance used for compression). Prefiltering is important because accurate shadows require large numbers of depth samples, as we saw in Section 2. This is equally true for both ordinary shadow maps and deep shadow maps.

Although deep shadows do not reduce the number of depth samples that must be taken from the scene, they greatly reduce the amount of data that must be accessed during filtering. For example, recall that in order to compute a shadow of dense hair with an expected error of $1\%$, approximately $N = 2500$ samples are needed. Using a deep shadow map with 250 samples per pixel, we would need to filter only $N = 10$ pixels to achieve the required accuracy. Furthermore deep shadows can be mip-mapped (see Section 5.3), and thus accurate shadows require only a constant number of pixel accesses even when filter widths are very large.

Prefiltering not only makes shadow lookups faster, but also allows deep shadows to be much smaller than the equivalent high-resolution depth map. This is an advantage when deep shadow maps are written, stored, and cached in memory. Note that the advantages of prefiltering are completely dependent on compression. If we did not compress the visibility functions at all, then each pixel would contain the data from all the underlying samples and would not be any smaller.

Fortunately, at any sampling rate there is an error tolerance that allows significant compression without compromising shadow quality. Specifically, recall from Section 2 that shadows of detailed geometry have an expected error of $O(N^{-1/2})$, where $N$ is the number of samples per deep shadow pixel. This error is a measure of the noise inherent in the sampled visibility function. Since there is no point in preserving noise, this suggests that $O(N^{-1/2})$ is a suitable tolerance for compression. The tolerance we actually use is $0.25/\sqrt{N}$, which is about half of the maximum expected noise magnitude of $0.5/\sqrt{N}$.[4]

Using this tolerance, we can show that deep shadows are asymptotically much smaller than regular shadow maps. Since each visibility function decreases monotonically from 1 to 0 (assuming that the filter function has no negative lobes), and the function decreases by at least the compression tolerance at each vertex, the compressed function can have at most $O(N^{1/2})$ vertices. The corresponding regular shadow map uses $O(N)$ storage for the depth values in each pixel, and so the deep shadow map must be smaller by at least a factor of $\Omega(N^{1/2})$.

[4] Recall that the expected error is only $O(N^{-3/4})$ when the shadow geometry is simple, which suggests that a compression tolerance of $O(N^{-1/2})$ is too large in this case. But since $N$ must be chosen large enough to control the noise artifacts in the worst-case pixels, there is little benefit to compressing more accurately than this.

The compression ratios are even better when the visibility functions are asymptotically piecewise smooth (which is the usual case). A simple Taylor series argument shows that in this case the compression error decreases quadratically in the number of output vertices, so that functions can be compressed with a tolerance of $O(N^{-1/2})$ using only $O(N^{1/4})$ vertices. Thus deep shadow maps are typically smaller than their regular counterparts by at least a factor of $\Omega(N^{3/4})$. This is a substantial advantage when many samples per pixel are used (say $N = 250$).

The main disadvantage of deep shadow maps is that they are significantly more expensive to compute than a regular shadow map of the same pixel resolution (because many more samples per pixel are taken). This contradicts the conventional assumption that shadow map generation is cheap [9]. On the other hand, they are typically no more expensive to compute than a shadow map with the same number of depth samples, and they are considerably cheaper to store and access. Note that generating more depth samples is relatively inexpensive in a scanline renderer, since it does not increase the shading cost.

Another potential issue with deep shadows is bias. Because filtering is performed at a constant $z$ depth, large objects may suffer from incorrect self-shadowing. Although this artifact also occurs with normal shadow maps, deep shadows exacerbate the problem because they encourage the use of large filter widths.

However, it is important to realize that the bias problems are no worse than they would be for an ordinary shadow map at the same filter width. The main limitation of deep shadow maps compared to high-resolution ordinary shadow maps is that the minimum filter width is larger (because each deep shadow pixel summarizes many depth samples). However, this can be considered an advantage: It provides deep shadows with an extra degree of freedom to control the tradeoff between shadow detail and noise. The deep shadow resolution should be chosen according to the minimum filter width desired (*i.e.* shadow detail), while the number of samples per pixel should be determined by the maximum acceptable noise. This strategy allows the depth samples that are required only for accuracy purposes to be represented very compactly, with bias problems that are no worse than they would be for a regular shadow map of the same pixel resolution.

# 5 Implementation Issues

## 5.1 Incremental Updates

Recall that each visibility function is defined as the weighted average of $n$ piecewise linear transmittance functions, according to equation (1). The naïve way to generate this function is to sort all of the input vertices and process them in $z$ order, evaluating the $n$ contributing functions at each vertex. Unfortunately this approach has $O(n^2)$ complexity: there are $O(n)$ input vertices, and $O(n)$ work is needed to compute the weighted average at each one. This is quite inefficient when large numbers of samples per pixel are used.

Instead, we describe an $O(n \log n)$ sweep algorithm that has a constant update cost per vertex. The algorithm is easier to understand if we first suppose that the transmittance functions are piecewise constant. In this case, we can efficiently compute the output function as follows. At $z = 0$, the weighted average is easily computed as $V(0) = 1$. We then process all of the input vertices in increasing $z$ order, which can be done in $O(\log n)$ time per vertex by storing the next vertex of each transmittance function in a heap. For every vertex, we update the current sum $V$ by simply subtracting out this transmittance function's old contribution and adding in its new contribution. That is,

$$V' = V + w_j(\tau_j' - \tau_j)$$

where $w_j$ is the filter weight for the chosen transmittance function, $\tau_j$ is the old value of this function and $\tau_j'$ is its new value.

This method can be extended to piecewise linear functions by using a similar technique to keep track of the output function's current value and slope. The update for each vertex consists of two steps: First, we extend the output function (using its current position and slope) to the $z$ value of the next input vertex, and then we update the current output slope using the method described above. (Vertical steps are handled as a special case, by updating the current position rather than the current slope.)

This technique is much more efficient than computing the weighted averages directly, and makes the algorithm practical even when very large numbers of samples per pixel are used. Note that all of the transmittance functions do not need to be stored simultaneously as the deep shadow map is rendered; it is sufficient to store only the previous several scanlines, as determined by the filter radius.

### 5.2 Colored Shadows

Colored shadows are supported by simply encoding a different visibility function for each color channel (one each for red, green, and blue). The compression algorithm processes all the channels simultaneously, and starts a new segment whenever any of the three functions would exceed its error threshold. The output is a sequence of tuples $(z, V_R, V_G, V_B)$. Notice that with this representation, three-channel maps are only twice as large as one-channel maps.

To reduce storage even further, our format allows each pixel to encode either one or three visibility functions depending on its needs. If all three channels happen to be the same, we store only one channel and set a flag indicating that this pixel is monochrome.

### 5.3 Mip-mapping

Since deep shadow maps are filtered like textures, it is straightforward to apply mip-mapping [12]. Starting with the highest-resolution deep shadow map, each new mip-map level is obtained by averaging and downsampling the previous level by a factor of two. Each pixel is defined by taking the average of four visibility functions, and recompressing the result.

To avoid the accumulation of too much error, the compression tolerance can be reduced on each successive level. For example, if the error threshold is cut if half each time, the total error will be at most twice that permitted for the highest-resolution map. This method corresponds to the analysis of Section 4, which suggests that the compression tolerance should be $O(N^{-1/2})$ in the number of contributing depth samples $N$. The storage per sample increases somewhat at coarser levels, but since the functions being compressed are asymptotically piecewise smooth, their compressed size is $O(N^{1/4})$ (see Section 2). This implies that the number of vertices per visibility function doubles once for every two mip-map levels, and that the full mip-map is approximately $1/(1 - \sqrt{2}/4) \approx 1.55$ times as large as the base level (rather than the $4/3$ ratio for an ordinary mip-map).

Mip-mapped deep shadows are filtered just like mip-mapped textures, including anisotropic filters, lerping between levels, etc. [1].

### 5.4 Tiling and Caching

We store deep shadow maps in a tiled format, similar to those for textures [8]. This lets us load and cache only the deep shadow map tiles that are actually accessed.

One important difference is that unlike textures, deep shadow map tiles and pixels require varying amounts of storage in memory. To deal with this, our file format includes a *tile directory* that specifies the starting offset and size of every tile. Similarly, each tile has a table indicating the starting position and size of every pixel. Tile caching is handled by providing a fixed number of tile slots; when a tile fault occurs, the slot chosen for replacement is resized if necessary to hold the incoming tile. In this way, each tile slot grows to accommodate the largest tile it has seen. (Optionally the tile slots could also be resized when the incoming tile is significantly smaller.)

### 5.5 Motion Blur

It is easy to support motion blur in deep shadow maps by simply associating a random time with every shadow image sample (and its corresponding transmittance function). When these samples are averaged together into visibility functions, they account for the average coverage over time as well as over the image plane.

While motion blur is also possible with ordinary shadow maps, it is very expensive because of the large filter widths needed for adequate anti-aliasing. Deep shadow maps do much of this filtering in advance, and thus reduce the number of pixels that need to be accessed for each lookup.

Motion-blurred shadows produced in this way are strictly correct only when the receiving object is stationary with respect to the shadow camera. In particular, moving objects cast incorrect shadows onto other moving objects (and onto themselves). This happens because the deep shadow map effectively blurs an object's shadow over the entire shutter interval, allowing one object to cast shadows onto other objects at a different times. However, even the ability to cast shadows onto stationary objects is useful, and the results are often acceptable even when the receiving object is moving.

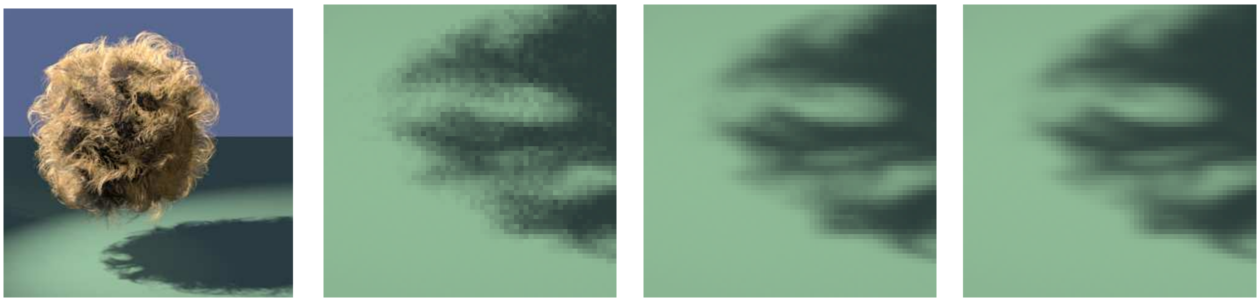
## 6 Results

We have implemented deep shadow maps in a highly optimized scanline renderer that also supports traditional shadow maps. We present experiments comparing these two techniques in terms of time and storage space. We also illustrate two additional capabilities of deep shadows: volumetric shadows and inexpensive motion blur.

Figure 6a shows a ball covered with 50,000 hairs. The individual hairs are significantly narrower than a pixel, and combine to form tufts and curls of various densities. The scene is illuminated by three spotlights, each of which casts shadows.

We have rendered this scene under various conditions to compare the performance of deep shadow maps with traditional shadow maps. Figure 6b shows a magnified view of the shadow cast by the hairball, rendered using a $512 \times 512$ normal shadow map. Shadow filtering was done using 16 samples per lookup on a jittered grid; using more samples does not increase the shadow quality due to the coarse resolution of the underlying shadow map. This image has obvious noise artifacts that would be unacceptable if animated. In order to eliminate these artifacts, Figure 6c was rendered using a $4k \times 4k$ shadow map with 400 samples per lookup for percentage closer filtering. The noise artifacts are much improved, but shadow filtering times were much longer (559 seconds *vs.* 19 seconds). Figure 6d was rendered using a $512 \times 512$ deep shadow map with 256 samples per pixel (the same number of depth samples as the previous case). Even though the deep shadows were much faster (37 seconds) and required only one-sixth the storage space, the shadow quality is actually slightly better than the $4k \times 4k$ shadow map image (because deep shadows consider every depth sample rather than randomly selecting a subset of them for filtering).

Figure 7 summarizes the results of similar tests at various maximum noise levels. Each row compares a deep shadow map to a normal shadow map with the same number of depth samples; in the deep shadow this was achieved by holding the resolution constant at $256 \times 256$ and adjusting the number of samples per pixel (as shown



***(a)*** *Ball with 50,000 hairs* ***(b)*** $512 \times 512$ *Normal shadow map* ***(c)*** $4k \times 4k$ *Normal shadow map* ***(d)*** $512 \times 512$ *Deep shadow map*

***Figure 6:*** *Hair ball and magnified shadows demonstrating noise from various methods.*

| | | *Time (sec)* | | *Space (MB)* | |
|---|---|---|---|---|---|
| *Samples* | *Error* | *Normal* | *Deep* | *Normal* | *Deep* |
| $2 \times 2$ | *25.00 %* | *34* | *45* | *0.34* | *0.49* |
| $4 \times 4$ | *12.50 %* | *39* | *46* | *1.34* | *0.79* |
| $8 \times 8$ | *6.25 %* | *60* | *45* | *5.34* | *1.18* |
| $16 \times 16$ | *3.12 %* | *145* | *46* | *21.34* | *1.53* |
| $32 \times 32$ | *1.56 %* | *554* | *46* | *85.34* | *1.84* |
| $64 \times 64$ | *0.78 %* | *2414* | *45* | *341.34* | *2.20* |

***Figure 7:*** *Comparison of deep shadows and normal shadows for the scene in Figure 6. The error column shows the expected error* $0.5/\sqrt{N}$ *associated with the given sampling density.*

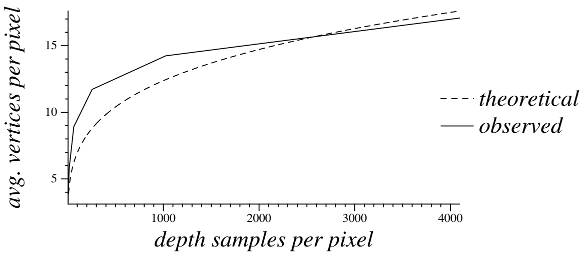



***Figure 8:*** *Theoretical and observed growth rates of deep shadow maps. (The theoretical growth rate of* $O(N^{1/4})$ *is taken from Section 4.)*

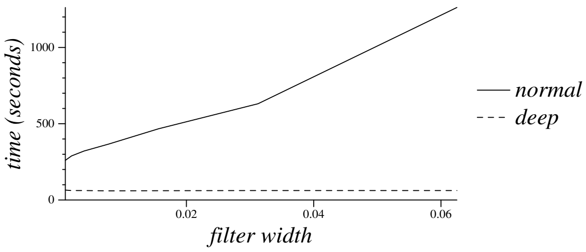



***Figure 9:*** *Filtering time as a function of filter width (expressed as a fraction of the shadow map size). While both methods access a constant number of pixels per lookup, larger filter widths result in worse cache coherence for normal shadow maps and slightly better cache coherence for deep shadow maps.*

in the first column). This column also represents the number of samples used for normal shadow map filtering, chosen to achieve the same accuracy as the deep shadow map at its minimum filter size. The second column shows the corresponding worst-case expected error level of $0.5/\sqrt{N}$; the deep shadows were compressed using an error tolerance of half this amount. All tests used the same absolute filter width, equal to the pixel size of a $256 \times 256$ shadow map. The shadow evaluation times were measured by rendering an image using each technique and subtracting the time for an image without shadows.

Observe that normal shadow maps become much more expensive as the error threshold decreases, while the deep shadow map filtering times are virtually constant. Normal shadow maps are only faster at $4 \times 4$ and fewer samples, sampling rates that are much too low for use in animations. In our implementation deep shadows have not been as extensively optimized as normal shadow maps, and it is likely that further speed improvements will be found. Shadow generation time was similar for both methods and is not shown; the compression overhead for deep shadows was negligible.

Notice that deep shadow maps grow very slowly in size as the error threshold decreases. The deep shadow sizes do not include mip-map overhead, which would make them approximately 1.5 times larger (see Section 5.3). Figure 8 plots the growth of the deep shadow map size with respect to the number of depth samples per pixel. About 40% of the pixels in this deep shadow map contain hair, while the rest contain simple geometry. Notice that the average number of vertices per compressed visibility function closely matches the $O(N^{1/4})$ behavior predicted in Section 4.

Figure 9 shows how both methods perform when a fixed accuracy is desired, but progressively larger filter widths are applied. In this case the number of filter samples for normal shadow maps can be held fixed. A single shadow map of each type was rendered to support the smallest desired filter size, and the same shadow maps were used to render progressively larger blurs. The filtering time for the normal shadow map grows rapidly with the amount of blur, even though the number of filter samples was held constant at $8 \times 8$. This can be attributed to worse cache performance as the filter region spans larger and larger portions of the shadow map. With the deep shadow map, on the other hand, mip-mapping allows the filtering times to be virtually constant. (In theory the cache performance of deep shadows is actually better at very large filter widths, since the lower-resolution mip-map levels contain fewer pixels.)

Figure 10 illustrates the importance of self-shadowing to the appearance of volumetric objects, while Figure 11 demonstrates that a single deep shadow map can be used for both volumetric and surface shadows. Finally, Figure 12 demonstrates the artifacts that occur when shadows are not motion blurred; this effect appears as strobing when animated. Unlike normal shadow maps, deep shadows allow motion blur to be added without incurring an extra filtering cost.

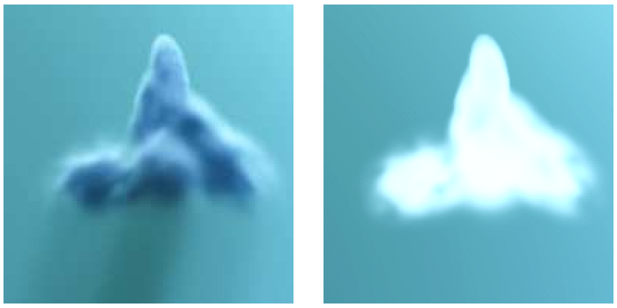


***Figure 10:*** *Cloud with and without self-shadowing.*

***Figure 11:*** *A cloud with pipes. Notice the shadows cast from surfaces onto volumetric objects and vice versa. A single deep shadow map contains the shadow information for the cloud as well as the pipes.*

## 7 Conclusion and Future Work

A nice feature of deep shadow maps is their generality: they support ordinary surfaces, volumetric objects, dense fur, and even motion blur, effects that would normally be handled using different techniques. With deep shadows they can all be combined in a single compact data structure, and rendered efficiently under a wide range of viewing and filtering conditions.

Carrying this idea further, deep shadow maps are an attractive representation for arbitrary volumetric data: fog densities, approximate illumination information, and so on. In this context, a deep shadow map can be viewed as a two-dimensional *function image* of piecewise linear one-dimensional functions. This representation is sparse in $z$, which allows it to take advantage of any smoothness in the raw data, while it is discrete in $x$ and $y$ (and allows binary search in $z$) in order to facilitate fast lookups. The domain can easily be mapped to a frustum or an orthographic space, and has the advantage of having an arbitrary extent and resolution in one of its dimensions. Three-dimensional filtering would be an easy extension, since the adjacent data values in $z$ can be found without searching and the representation is sparse in this direction.

Although the function image representation may not achieve quite as much compression as a three-dimensional octree or wavelet expansion, and it is not quite as fast to index as a three-dimensional grid, it is an excellent compromise that retains most of the benefits of both of these extremes. We expect that deep shadow maps and their underlying representation will find other interesting applications in the future.

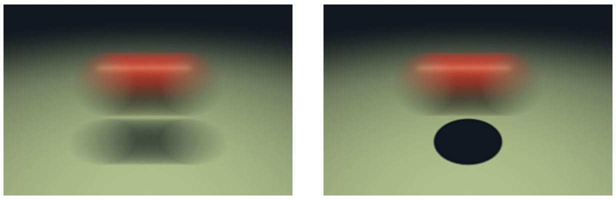


***Figure 12:*** *Rapidly moving sphere with and without motion blur.*

## Acknowledgements

Thanks to Adam Woodbury for modeling the cloud in Figure 10, Christian Hill and Brad Winemiller for production help, and Rob Cook and Matt Pharr for valuable comments on early drafts of the paper.

## References

[1] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, November 1986.

[2] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 271–280, July 1989.

[3] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 165–174, July 1984.

[4] Brett Keating and Nelson Max. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Eurographics Rendering Workshop 1999*, pages 205–220, New York, June 1999. Springer-Verlag.

[5] Marc Levoy. *Display of Surfaces from Volume Data*. Ph.D. thesis, University of North Carolina at Chapel Hill, 1989.

[6] Nelson Max. Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In *Eurographics Rendering Workshop 1996*, pages 165–174, New York, June 1996.

[7] Don P. Mitchell. Consequences of stratified sampling in graphics. In *SIGGRAPH 96 Proceedings*, pages 277–280. Addison Wesley, August 1996.

[8] Darwyn R. Peachey. Texture on demand. Unpublished manuscript, 1990.

[9] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 283–291, July 1987.

[10] Jonathan W. Shade, Steven J. Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *SIGGRAPH 98 Proceedings*, pages 231–242. Addison Wesley, July 1998.

[11] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings)*, 12(3):270–274, August 1978.

[12] Lance Williams. Pyramidal parametrics. *Computer Graphics (SIGGRAPH '83 Proceedings)*, 17(3):1–11, July 1983.

[13] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, November 1990.