Hair Self Shadowing and Transparency Depth Ordering Using Occupancy maps

Erik Sintorn[*]
Chalmers University of technology

Ulf Assarsson[†]
Chalmers University of Technology

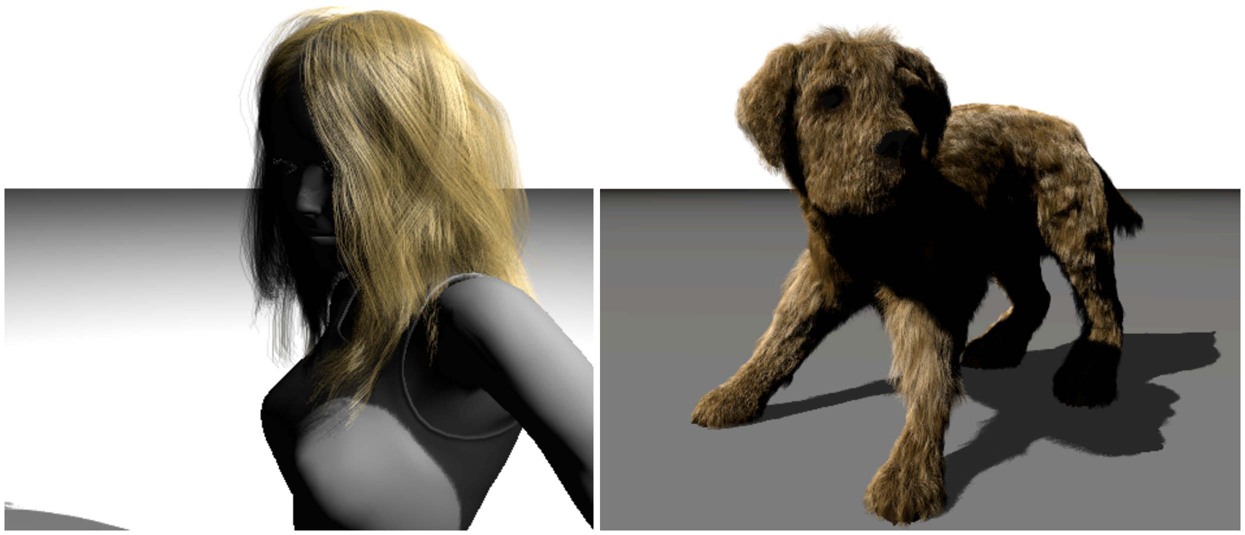


Figure 1: *The woman renders in 37.3 fps using 20k hair strands (300k line segments). The dog renders in 17.2 fps using 400k hair strands (2M line segments).*

Abstract

This paper presents a method for quickly constructing a high-quality approximate visibility function for high frequency semi-transparent geometry such as hair. We can then reconstruct the visibility for any fragment without the expensive compression needed by Deep Shadow Maps and with a quality that is much better than what is attainable at similar framerates using Opacity Maps or Deep Opacity Maps. The memory footprint of our method is also considerably lower than that of previous methods. We then use a similar method to achieve back-to-front sorted alpha blending of the fragments with results that are virtually indistinguishable from depth-peeling and an order of magnitude faster.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shadowing I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Visible line/surface algorithms

Keywords: hair rendering, opacity maps, deep shadow maps

1 Introduction

Rendering convincing images of hair and fur remains a very challenging problem in real-time graphics. The complex light-scattering properties of hair-fibers (thoroughly examined in [Marschner et al. 2003]) combined with the sheer number of very thin hair-strands on a human head make conventional rendering techniques either extremely time consuming, even by offline rendering standards, or fail completely due to aliasing problems.

When rendering individual hair strands, either as thin cylinders or as line primitives, the two main challenges are the self-shadowing effects within the volume of hair and appropriately capturing the sub-pixel width of each hair strand. The self shadowing problem is usually dealt with by using a shadow-map representation that, instead of storing a single depth, stores a visibility function for any depth in each shadow-map pixel [Lokovic and Veach 2000]. *Opacity Maps* [Kim and Neumann 2001] discretize the visibility function as a 3D-grid, centered around the hair, with a visibility value per grid-cell. Our solution is related to Opacity Maps but uses a representation of the visibility information that can be rendered in

[*]e-mail:erik.sintorn 'at' chalmers.se
[†]e-mail:uffe 'at' chalmers.se

three passes, as opposed to one pass per grid-slice, and requires only three 2D textures per light instead of a large 3D texture per light. The latter characteristics improve both speed and memory footprint, which enhance the usability for games and allows for using significantly higher resolutions of the opacity map.

Since each hair strand will normally be much thinner than the pixel width, no current hardware super-sampling scheme is sufficient to capture the high-frequency geometry. Supersampling by rendering to an offscreen buffer and then downsampling this to screen would of course work if the resolution was high enough, but since this causes an explosion in the number of fragments generated, and the fragment shader for hair is typically quite expensive this is not a viable option for realistically thin hair. Instead, alpha blending is normally applied to approximate antialiasing. Also, blond or light-colored hair fibers are semi-transparent which in itself requires alpha blending. Properly using alpha blending to achieve transparency effects requires that the fragments are drawn in a back-to-front order, which is not trivially done in hardware accelerated rendering. The only fail-safe solutions available so far are explicit sorting, which is prohibitively expensive for real-time rendering of hundreds of thousands of line-segments, or depth-peeling which requires as many rendering passes as there are fragments occupying one pixel, i.e. the depth-complexity of the image, which in hair rendering can be several hundreds. In this paper we suggest a novel technique, which builds on the technique we introduce for self-shadowing, to approximate the depth-order of each fragment and then render the hair in a single pass with approximate alpha blending which is very close to the result obtained with depth-peeling.

Our contributions are:

- A very fast reconstruction of the visibility function with a resolution much higher than that of previous realtime methods,
- Significantly less memory consumption than for opacity maps
- Very fast approximate back to front ordering of semi-transparent hair fragments for alpha blending

Our occupancy shadow map can also be used orthogonally with existing techniques that considers light scattering effects [Marschner et al. 2003; Zinke et al. 2004; Zinke et al. 2008].

2 Previous and Related Work

An extensive amount has been written on hair-rendering and semi transparent self shadowing, and even more on shadow-generation in general. In this section we will attempt to cover the previous work most relevant to this paper and refer the reader to [Ward et al. 2007] for a more complete survey.

2.1 Self-Shadowing

Deep Shadow Maps were introduced in 2000 by Lokovic and Veach [Lokovic and Veach 2000], as a substitute for shadow maps when rendering high frequency geometry such as hair or smoke. For each pixel in the shadow map the visibility is sampled at regularly or irregularly spaced intervals. This function is then compressed into a piecewise linear function of the depth from the light-space near-plane. Using sufficiently short sampling intervals, this method allows for storing a visibility function that maintains the overall shape of the true visibility function even with fairly strong compression. The method has been used successfully in many offline rendering projects, but the sampling and compression is not trivial in realtime on current graphics hardware. In [Mertens et al. 2004] a solution for generating the visibility function in realtime is suggested. They cluster the fragments in K bins, based on the fragment depth value, whose position is generated in a first rendering of the hair. In a second pass, the variance of each bin is calculated, and finally histogram binning of the fragments is performed in a third pass. While this algorithm did not perform in real time at the time the paper was written, it should work well on current graphics cards, and should significantly improve on the results obtained from using opacity maps with few slices. More recently, [Hadwiger et al. 2006b] suggest a different way to generate the deep shadow maps on graphics hardware, but their method does not run in real-time and does not support semi-transparent primitives as is necessary for hair rendering.

Opacity Shadow Maps [Kim and Neumann 2001] is a similar technique, but instead of storing a complex visibility function per pixel, the volume of hair is divided into a number of slices, and the hair geometry is then rendered once for each slice, each time moving the far-plane one slice width further back. By rendering the hair with a constant color and using additive blending, this gives us the opacity at each slice, which can then be interpolated at any depth. The Opacity Maps technique was implemented for the *Nalu* demo by nVidia in 2005 to produce realtime self shadowing of a model with 4096 hair strands [Nguyen and Donelly 2005]. Their implementation used multiple render targets to render the opacity of 16 slices in a single renderpass. As shown in [Kim and Neumann 2001], 16 slices is nowhere near enough for realistic self shadowing, and the technique does not trivially extend to render more slices than would fit in the maximum allowed number of rendertargets, without reverting to several render passes.

[Sintorn and Assarsson 2008] improves on the performance of generating the opacity maps . The individual hair-segments are approximately sorted into the slices on the GPU in $O(nlog(n))$ where n is the number of slices. The hair can then be rendered into the opacity maps in a single pass through the hair geometry. This technique allows for rendering self-shadowed hair with 256 slices of opacity maps in real-time, but it does put the requirements on the hair geometry that it is rendered as line primitives, with finely tesselated hair segments for good performance.

In a paper called Deep Opacity Maps [Yuksel and Keyser 2008], the authors suggest modifying the algorithm so that the hair geometry is rendered as opaque primitives in a first pass to establish the depth of the fragment nearest to the lights near-plane for each pixel. They then use this as a per-pixel start of the opacity map depth range to significantly reduce the depth between two slices. While this almost completely removes the *banding* artifacts that are a big problem with opacity maps, the visibility for each pixel is still regularly sampled at different depths and linearly interpolated in between, so the technique fails to correctly capture the sudden steep changes that are very common in the visibility function unless a very high number of slices are used (see Figure 5). We borrow this same technique in our solution, and we also find the *furthest* fragment depth for each pixel.

In a related paper [Zinke et al. 2008], a complete light scattering model is introduced, based on [Marschner et al. 2003] that divides the multiple scattering function into *global* multiple scattering and *local* multiple scattering. In the global multiple scattering calculations, the selfshadowing properties of hair are taken into account, considering not only the visibility of direct illumination (as in our and previous shadowing models) but also an approximation of the forward-scattered light through each hair-strand along the shadow ray. While they achieve impressive results at interactive framerates, sometimes almost indistinguishable from a pathtracing reference, they use no more than four slices to sample the light-transmittance and direct illumination visibility, which, as noted above, would not be sufficient to capture high-frequency changes in the visibil-

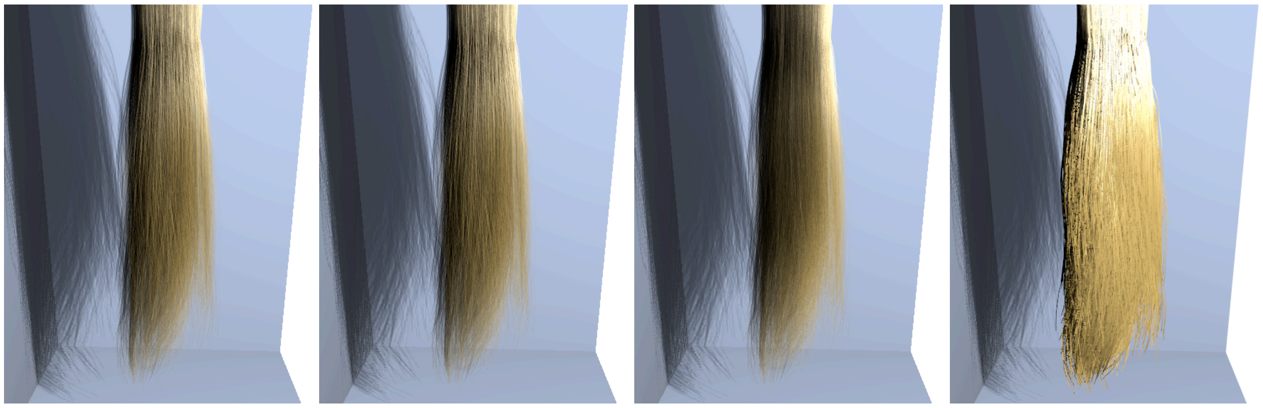


Figure 2: *Transparency sorting using depth-peeling (ground truth, 4.01 fps, max depth complexity of 112 layers), our method (51 fps), weighted-average blending (72 fps), and using no transparency (120 fps). All images use our occupancy maps for self-shadowing.*

ity function. In addition, they do not address transparency sorting. Our presented algorithm could orthogonally be used with their light scattering solution to enhance the quality of the self-shadowing and add transparency sorting.

The generation of our *occupancy-maps* resembles the voxelisation performed in [Eisemann and Décoret 2006], and they mention emulation of deep shadow maps as an application of their method.

The term *occupancy map* was used in a previous paper [Staneker et al. 2003] to describe a map used to improve occlusion queries.

2.2 Alpha blending

Alpha blending is commonly used in hair-rendering for two purposes. First, blond or light-colored hair is semi-transparent. It is standard practice in realtime rendering to render semi-transparent, non-refractive materials using alpha blending. If the semi-transparent fragments are blended in back to front order, a very convincing transparency effect is achieved. Secondly, hair strands are, in normal viewing scenarios and screen-resolutions, much thinner than the pixel width. So much thinner in fact, that no current super-sampling scheme is sufficient to capture the hair without severe aliasing. Human hair strands have a width of 17-100μm. An approximate, but visually convincing solution to this is to use alpha blending to render the strands with a very low alpha value. Again, to properly blend alpha weighted fragments with the standard transparency blending equation, they must be drawn in a back to front order, which is not trivial to do fast on current graphics hardware.

In [Sintorn and Assarsson 2008] the individual hair segments are Quicksorted on their depth into a number of bins (typically 256) on the GPU using the Geometry Shader and Transform Feedback. While this does not guarantee that fragments are drawn in exact back to front order, it was shown that this approximate sorting was sufficient to produce very plausible images.

To assure that all fragments are drawn in a back to front order, a technique called *depth-peeling* is commonly used. With this method, the same geometry is drawn several times with the depth test set to pass the furthest fragment, and each pass uses the depth buffer of the previous pass as a depth qualifier. In this way, the fragments are peeled off one layer at a time until an occlusion query reports that no more fragments are drawn. The problem with using this technique for hair rendering is that it requires as many passes as there are fragments occupying the same screenspace pixel, often resulting in several hundred render passes.

An attempt at improving the speed of the depth-peeling algorithm was made by [Liu et al. 2006]. They peel several layers at the same time by rendering to all available MRTs and sorting the fragments in the fragment shader. The theoretical peak performance of this algorithm would be an 8 times improvement over standard depth peeling, but due to the possibility of read-modify-write hazards, this is not achieved in practice. In a recent technical report by nVidia [Bavoil and Meyers 2008], a technique is suggested where the frontmost and furthest layer can be peeled in each pass, effectively doubling the performance of the depth peeling. While both of these algorithms increase the speed of depth-peeling, the running time is still dependent on the depth-complexity, and neither would work for rendering individual hair strands in real time.

In [Bavoil and Meyers 2008] a single pass approximation of depth peeling is suggested, where the alpha weighted color of all fragments are summed, together with the alpha sum. The average alpha is then found as the sum of alphas divided by the depth complexity for each pixel and the final color is the sum of colors divided by the averaged alpha. When rendering hair, where all fragments will have similar alpha values and very varying colors, the resulting image becomes very dull however, compared to the depth-peeled image (see Figure 2).

3 Algorithm

Previous solutions to rendering opacity maps and to solving the problem of alpha blending rely either on sorting the input primitives [Sintorn and Assarsson 2008] or rendering the geometry in several passes [Kim and Neumann 2001], [Everitt 2001]. Both approaches are very time consuming in hair rendering as the geometry when rendering individual hair-strands is invariably very heavy.

We try to solve both problems by using a new method for approximating the depth-order of each fragment. For opacity mapping, this is the order the fragment would have if all fragments that are projected into the same light-space pixel are sorted on their depth from the near plane, and for alpha blending it is the depth-sorted order in screen space. As the visibility function and the "alpha contribution" function (explained below) are very similar, we will begin by talking only about generating and sampling our visibility function first, and then proceed to explain how this applies to alpha blending.

The *visibility* of a fragment at some depth d from the light source is a function of the *opacity* at that depth. We will write that the visibility $V = e^{-\Omega}$ where the opacity Ω is the integral of the density of hair along that ray. The Ω function can easily be sampled at

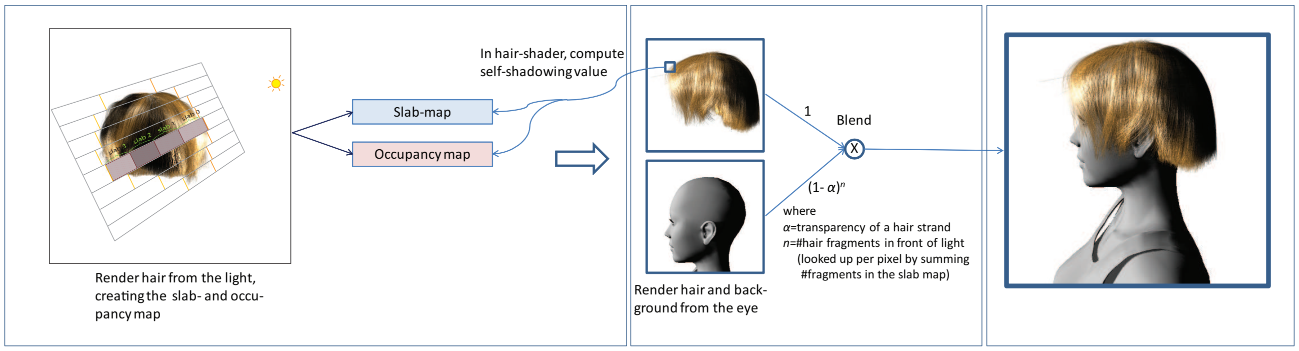
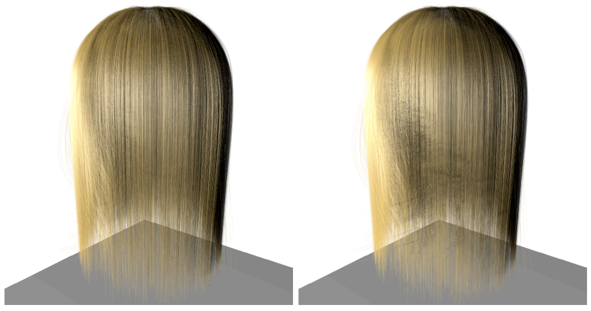

slab 3
slab 2
slab 1
slab 0
Render hair from the light, creating the slab- and occupancy map
Slab-map
Occupancy map
In hair-shader, compute self-shadowing value
Render hair and background from the eye
1
Blend
X
(1- α)^n
where
α=transparency of a hair strand
n=#hair fragments in front of light (looked up per pixel by summing #fragments in the slab map)

Figure 3: *Outline of the full rendering process. Initially, the hair is rendered from the light source, resembling a 3D-rasterization process. The output image consists of an occupancy map storing hair-strand occupancy information in 128 depth slices per pixel, with one bit per slice. In addition, a slab-map stores for each slab (chunk of 32 bits) the exact number of rasterized fragments. These two maps can be combined to reconstruct the shadow value with high precision at any position inside the hair. Secondly, the hair and background is rendered from the eye into separate textures. For each rasterized hair-fragment, the shader uses the occupancy- and slab-map from step 1 to compute the shadow value. Finally, the hair- and background image are correctly blended together.*

regularly spaced depths using the Opacity Maps, or Deep Opacity Maps algorithms.

The basic observation we make is that the visibility function will always be a strictly decreasing function and that only a few fragments are enough to cause very sharp dips in visibility at any depth. As can be seen on the Figure 5(b), a regularly spaced sampling of this function will require many sampling points to accurately reproduce the shape of the function. Using too few samples, as in Figure 4(b), can lead to overestimating the visibility for the fragments close to the light, making the hair look opaque. In [Yuksel and Keyser 2008] it is suggested that one should use slices of linearly increasing width to increase the precision at low depth values, but as can be seen in Figure 5(a) this is not a good strategy when the main dip of the function (the first clump of hair fragments) is not caused by the first fragments recorded for this shadow-map texel. This is a big problem for Deep Opacity Maps with few slices, since it means that in an animated sequence, a stray hair strand may suddenly cause unnaturally strong shadows to appear (see Figure 4).

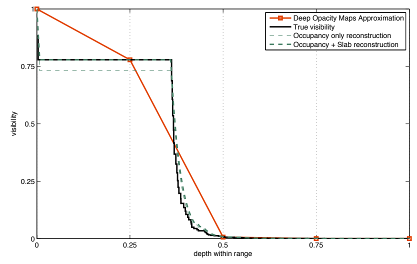
Essentially, we propose replacing the opacity maps (or deep opacity maps), which have a real-value opacity stored per texel and depth-slice, with an *occupancy-map* plus a *slab-map*.



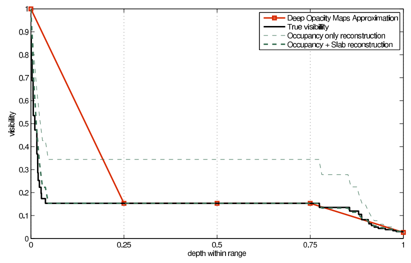
(a) Our algorithm (b) Deep Opacity Maps (4 slices)

Figure 4: *Deep shadows in hair. On the left is the results of our algorithm and on the right Deep Opacity Maps with four slices. Notice the incorrect shadow stripes in the right image.*

The occupancy map has only a bit per texel and slice, storing whether the slice is occupied by at least one fragment. This map can be generated in a single pass for up to $N * 128$ slices where N



(a) The ray from the light is first intercepted by a stray hair strand.



(b) A ray from the light that enters at the specular highlight

Figure 5: *Two examples of the visibility function in Figure 4. The "ground truth" was obtained through depth peeling and represents the result of opacity mapping with an infinite number of slices.*

Figure 6: *Stills from an animation with 20000 hair strands, running at 30 fps.*

is the number of simultaneous render targets allowed by the hardware, as explained below. By letting each set bit in this map signify an opacity increase of $\frac{F}{S}$ where F is the number of fragments recorded to fall within this texel and S is the total number of set bits in the occupancy map, we could reconstruct the visibility function with the occupancy map alone. However, as the hair fragments tend to be clustered around certain depth values, the reconstruction need not necessarily be very good. Consider the diagram in Figure 5(a) where a stray strand gives rise to a first dip in visibility (at depth 0.0) and then a large number of fragments are clustered around a larger depth value (around depth 0.4). Using only the occupancy map would under-estimate the visibility of the fragments directly following the first. Note however that while the visibility function we can reconstruct from only an occupancy map with 128 slices may not give the correct visibility within the depth range, it does follow the sudden changes of the real visibility function and as such may still be a better choice than a Deep Opacity Map with four slices (see Figure 5(b)) since the two techniques would use the same number of rendering passes and have the same memory footprint.

To further improve the quality of our reconstructed visibility function, we introduce what we have called a *slab-map*. This map stores for each texel the number of fragments recorded within one *slab*, which is what we call a group of slices. To reconstruct the visibility function we can now instead let a set bit in the occupancy map represent an opacity increase of $\frac{F_i}{S_i}$ where F_i is the number of fragments recorded in slab i and S_i is the number of set bits in the occupancy map corresponding to slices within slab i (see Figure 8). Note that this will force our visibility function to have correct values at the start and end of each slab, which limits the deviation from the ground truth.

In all examples in this paper we have used an occupancy map with 128 slices, which can be stored in a four channel unsigned int texture and is generated as explained in the next section. These slices are divided into four slabs of 32 slices each, meaning that the slab-map is a four channel float texture. With these two maps we can reconstruct a visibility function that stays very close to the "true" visibility funciton, i.e. the visiblity function one would obtain from using an infinite number of opacity maps or, as we have done for comparison, by depth peeling the hair from the light's viewpoint.

3.1 Generating the maps

As noted above, we will use a per texel depth range to maximize the precision in our occupancy- and slab-maps. We begin by generating a *depth-range map*, containing for each texel the nearest and furthest fragment-depth from the light's point of view (see Figure 7). This can be obtained in a single pass by rendering the hair from the light and writing the depth to the r and a channels, using a *min* blending function for the color channels and a *max* blending function for the alpha channel. However, we have found that it was faster to render the hair twice with depth testing enabled and changing the depth-test function, due to the large amount of fragments that can be discarded in early z-culling.

Given the depth-range map, we can construct the *occupancy map*. The occupancy map is a four-channel unsigned int texture that will be read as a continuous string of 128 bits, where a set bit signifies that the corresponding slice contains at least one fragment. To create this map we again render the hair from the lights viewpoint. In the fragment shader, we fetch the $near$ and far value of the depth range at this texel and can then obtain the fragments relative depth $d \,\epsilon\, \{0, 1\}$ within the depth-range as

$d = \frac{fragcoord.z - near}{far - near}$

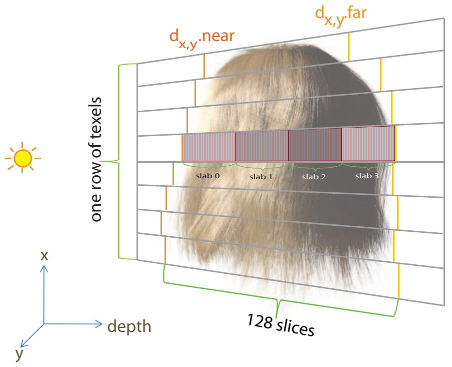


Figure 7: *Construction of the depth-range map. The hair is rendered from the light. For each texel of the occupancy map, we store the nearest and furthest fragment-depth, to achieve a per-texel depth range to maximize the utilization of the 128 slices.*

We now know the fragment falls into slice $\lfloor d * N_{slices} \rfloor$, where N_{slices} is the number of slices in the occupancy-map and so we set the corresponding bit in the output color. Using the *bitwise-or* as the blending operation on the framebuffer, this will give us the occupancy map in a single pass.

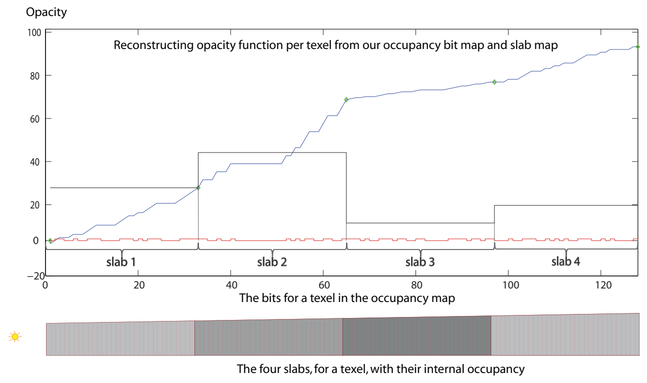


Figure 8: *Reconstructed opacity (visibility is $e^{opacity}$) function (blue) per texel from our occupancy bit map (red) and slab map (black). Each slab holds the number of hair strand fragments rasterized to this slab and each set occupancy-bit means that at least one fragment was rasterized into that slice.*

As noted above, we can create a better reconstruction of the visibility function if we, besides occupancy information for each slice know something about the actual number of fragments that fall within each slab of the depth-range. We can construct our *slab map* in an additional pass over the hair geometry where the fragment shader simply writes a value of 1.0 to the color channel corresponding to the slab $\lfloor d * N_{slabs} \rfloor$ the fragment falls within. N_{slabs} is the number of slabs, i.e. 4 in our implementation. We use additive blending during this pass to sum up the number of fragments that fall within each slab.

Note that the construction of the occupancy- and the slab-map are independent of each other and that both could be generated in a single pass on hardware that supports different blendfunctions for different rendertargets (i.e. recent ATI cards with DirectX 10.1). We have not yet had the opportunity to test this.

3.2 Finding the depth order of the fragment

We will use the three maps (the occupancy-, slab- and depth-range map) when rendering the hair to screen to approximate the depth order of the fragment, either in its depth from the light-source or from the camera. To do this, we make two approximating assumptions about the distribution of fragments. First, we assume that there will be an equal number of fragments in each of the occupied slices within one slab. Second, we assume that the fragments within one slice will be uniformly distributed over that slice.

We can then, in the fragment shader when rendering the hair from the cameras viewpoint, find the slab and slice the fragment occupies as described in Section 3.1 and estimate the depth-order of the fragment as the sum of the slab-sizes of the slabs preceding the current plus the number of fragments preceding the fragment within its slab. The latter is where we have to approximate. What we know for certain is the number of fragments F_i occupying the slab and which individual slices are occupied. We will count the total number of bits B_{total} set within the slab and the number of bits B_{before} set that correspond to slices in front of the current. $\frac{F_i}{B_{total}}$ is then the average number of fragments contributing to a set bit in the slab and we can estimate the number of fragments preceding this fragment within the slab as $\frac{F_i}{B_{total}} * B_{before}$. Counting the number of set bits to obtain B_{total} is easily done in $O(log\ n)$ [E.], where n is the number of bits. To obtain B_{before} we simply mask out the bits corresponding to slices after the current fragments before counting.

Since the fragment will lie at some relative depth into the slice, we will linearly interpolate between the results of it's occupied slice and the previous slice.

3.3 Self shadowing

As in the original opacity map implementation [Kim and Neumann 2001], we will define the *opacity* of a fragment as the sum of the number of fragments in preceeding slices times a constant shadow weight w. Given the estimated order, o, of the fragment this is computed as $opacity = o*w$. Note that while the order of the fragment will be approximated due to the assumptions stated in section 3.2, it will converge towards the correct value on each of the slab boundaries (see Figure 8). It will also converge as the resolution of the occupancy map increases.

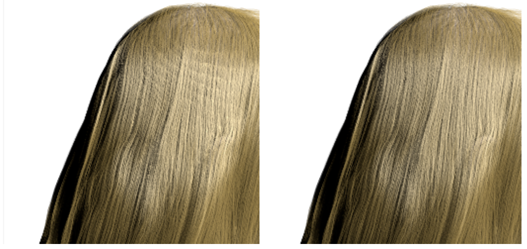
As in any shadow-map based algorithm, ours suffers from aliasing problems along the silhouettes. This could be handled by sampling the various maps several times but this would become very expensive. However, we have found that since we are alpha blending many fragments per pixel, simply jittering the texture coordinate by half a texel-width before doing any map-lookups takes care of most aliasing problems (see Figure 9).

3.4 Alpha blending

Correctly alpha blending the hair would require drawing all the fragments in a back to front order. The only non-approximate method for doing this is, to our knowledge, to use depth-peeling. For hair rendering this is impractical however as it requires as many render-passes as is the depth complexity of the image, which will be in the hundreds in most cases.

Instead, we suggest an approximation that works since we usually render all hair-fragments with the same alpha-value, depicting some transparency and thinness of the hair strand.

The blending function typically used for rendering transparent objects is: $f = \alpha c + (1-\alpha)b$ where f is the resulting framebuffer



(a) Without jittering (b) With jittering

Figure 9: *Since many fragments are blended per pixel, aliasing problems can be easily fixed by jittering texture coordinates slightly.*

color and c is the color to be blended on the background color b. For three fragments blended in back to front order, this would expand to:

$$f = \alpha_2 c_2 + (1-\alpha_2)(\alpha_1 c_1 + (1-\alpha_1)(\alpha_0 c_0 + (1-\alpha_0)b)),$$

which can be rewritten as:

$$\begin{aligned} f = &\alpha_2 c_2 + (1-\alpha_2)\alpha_1 c_1 + \\ &(1-\alpha_2)(1-\alpha_1)\alpha_0 c_0 + \\ &(1-\alpha_2)(1-\alpha_1)(1-\alpha_0)b \end{aligned}$$

The above equation shows us that we could chose to blend the fragments front to back, using separate blending equations for color and alpha: $f_{color} = b_\alpha(\alpha c) + b_{color}, f_\alpha = (1-\alpha)b_\alpha$. This is called *front-to-back compositing* and is commonly used in Volume Rendering [Hadwiger et al. 2006a]. In our case, alpha values are approximated as being the same for all hair fragments which allows us to rewrite the equation as:

$$f = (1-\alpha)^0 \alpha c_2 + (1-\alpha)^1 \alpha c_1 + (1-\alpha)^2 \alpha c_0 + (1-\alpha)^3 b$$

In fact, it turns out that for n fragments the result is:

$$f = \sum_{i=0}^{n-1} (1-\alpha)^i \alpha c_{n-1-i} + (1-\alpha)^n b$$

Using our approximation of the depth-order of the fragment, we can additively blend unsorted fragments by writing their fragment color as:

$color_{out} = (1-\alpha)^o * \alpha * color_{in}$,

where o is the estimated order of the fragment. Note that this will amount to a sum of hundreds of very small values and that a 32-bit float rendertarget is necessary to get correct results. We finally add this buffer to the background image, multiplying the background color with $(1-\alpha)^n$, where n is the total number of fragments written to that pixel, found by summing all four slabs.

This algorithm gives very good results most of the time when compared to depth-peeling, but artifacts can at some times be seen where very many fragments fall into the same pixel (typically along silhouettes), due to the limited precision. This can be remedied almost for free however by using an alpha component of 1.0 in the $color_{in}$ for each hair fragment. We know that the correct sum of the alpha channel should be:

$\Lambda = \sum_0^{n-1} (1-\alpha)^i * \alpha * 1.0 = \alpha \frac{1-(1-\alpha)^n}{1-(1-\alpha)} = 1 - (1-\alpha)^n$,

where n is the total number of hair-fragments written to this pixel. When the final image is composited from the hair and background images, we can then adjust the final color of the hair as $hair.rgb = hair.rgb * \frac{hair.a}{\Lambda}$. With this adjustment, the intensity of the hair color will be correct while the actual color may differ slightly from the depth-peeled ground truth. In practice the images obtained by depth peeling and by using our algorithm are almost indistinguishable (see Figure 2).

3.5 Opaque objects

The shadow map for all opaque objects is rendered in a first pass. Then, when rendering the hair, the fragment shader first looks into the shadow map as usual to see whether an opaque object shadows the fragment and if so, no hair self-shadowing need be considered. When rendering the opaque objects, the slab-map is sufficient to find the number of hair-fragments between the light and the current fragment, and the visibility can be calculated from that. When using our algorithm for alpha blending, the background image is first rendered with a depth buffer. This depth buffer is then used when generating the depthrange- occupancy- and slab-map to discard hair fragments that lie behind opaque objects.

4 Results

All measurements were performed an a GeForce GTX 280 using an image resolution of 800x600 unless otherwise stated. The occupancy- and slab-map resolution in all examples is 512x512 pixels. We have used the simple hair shader suggested by Kajiya and Kay [Kajiya and Kay 1989] throughout our work for simplicity. Compared to our own implementation of Deep Opacity Maps with four slices, our algorithm requires one more render pass (i.e., rendering the occupancy map since in terms of computation time, rendering the slab-map is exactly the same as rendering the Deep Opacity Map), and a slightly more complex pass for the final hair rendering. In our experiments, rendering the same model with Deep Opacity Maps and with our algorithm took 33.5 ms and 41 ms respectively (our alpha blending algorithm was used in both cases). The quality is very much improved however, especially for animations (see Figure 4). When run on the same model and at the same resolution as the title image of [Sintorn and Assarsson 2008], using the same graphics card (GeForce 8800GTX), our algorithm renders the image at 16.1 fps which is about twice as fast as the timings reported in that paper (see Figure 10).

Using a resolution of 512x512 for our depthrange-, occupancy- and slab-map, the memory requirement is 10MB. This can be compared to using Deep Opacity Maps with 16 or 32 slices (18MB or 34MB respectively) or standard Opacity Maps with 128 or 256 slices (130MB or 258MB respectively). Note that in both of these algorithms one could represent the opacity per slice with a byte instead of a float, resulting in lower memory consumption, but a coarser approximation of the opacity.

One of the main contributions of this paper is the algorithm suggested for out-of-order alpha blending for transparency. Using this algorithm we can produce images that are very close to a depth-peeling reference at a fraction of the time. In Figure 2 our algorithm renders about 15 times faster than the depth-peeling reference, and that is for a fairly low depth complexity (in our tests, a depth complexity of 300-400 is not uncommon for normal viewing conditions of some models).

The timings for each part of our algorithm (including the times for interpolating the hair and rendering the background for completeness) are given in the table below. These are taken for a frame of the animation shown in figure 6.

Algorithm step	time (ms)
Generate hair	
Interpolate control hair keyframes	0.01
Interpolate hairs from control hairs	2.9
Self-shadowing	
Find depth range	2.54
Render occupancy map	2.56
Render slab-size map	2.89
Alpha approximation	
Find depth range	2.63
Render occupancy map	1.8
Render slab-size map	1.88
Final image	
Render BG shadow map	0.5
Render background	1.0
Render hair	12.8
Compose BG and Hair	0.1

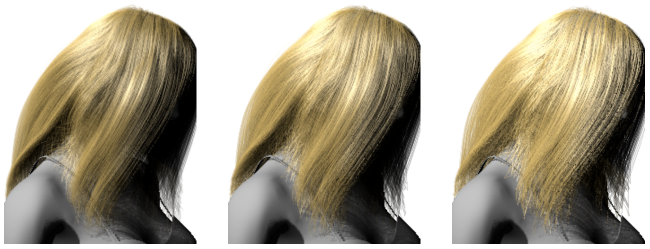


Figure 10: *Left to right: 20000 hair strands at 27.6 fps, 10000 strands at 37.0 fps, 5000 strands at 60.6 fps. Each hair strand consists of 30 line segments.*

5 Conclusion

This paper has presented a method for self-shadowing in hair that can replace opacity maps or deep shadow maps. Our method has the advantage of using more than an order of magnitude lower memory footprint, with a maintained high quality compared to Opacity Maps and significantly higher quality compared to Deep Opacity Maps with few slices. In addition, we show how the same method can be used to solve the problem of transparency sorting, when rendering the hair-strands from the eye. Used in combination, the two techniques allow for high quality rendering of dynamic hair, and the speed makes the technique suitable not only for offline rendering but also real-time applications, e.g. games.

References

BAVOIL, L., AND MEYERS, K. 2008. Order independent transparency with dual depth peeling. Tech. rep., nVidia, February.

E., A. S. Bit twiddling hacks.

EISEMANN, E., AND DÉCORET, X. 2006. Fast scene voxelization and applications. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, ACM, New York, NY, USA, 8.

EVERITT, C., 2001. Interactive order-independent transparency. NVIDIA white paper, citeseer.ist.psu.edu/everitt01interactive.html.

HADWIGER, M., KNISS, J. M., REZK-SALAMA, C., WEISKOPF, D., AND ENGEL, K. 2006. *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA.

HADWIGER, M., KRATZ, A., SIGG, C., AND BÜHLER, K. 2006. Gpu-accelerated deep shadow maps for direct volume rendering. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/Eurographics symposium on Graphics hardware*, ACM, New York, NY, USA, 49–52.

KAJIYA, J. T., AND KAY, T. L. 1989. Rendering fur with three-dimensional textures. *Proceedings of SIGGRAPH*, 271–280.

KIM, T., AND NEUMANN, U. 2001. Opacity shadow maps. In *In Rendering Techniques 2001*, Springer, 177–182.

LIU, B., WEI, L.-Y., AND XU, Y.-Q. 2006. Multi-layer depth peeling via fragment sort. Tech. rep., Microsoft Reasearch.

LOKOVIC, T., AND VEACH, E. 2000. Deep shadow maps. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 385–392.

MARSCHNER, S. R., JENSEN, H. W., CAMMARANO, M., WORLEY, S., AND HANRAHAN, P. 2003. Light scattering from human hair fibers. *ACM Trans. Graph. 22*, 3, 780–791.

MERTENS, T., KAUTZ, J., BEKAERT, P., AND REETH, F. V. 2004. A self-shadow algorithm for dynamic hair using density clustering. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, ACM, New York, NY, USA, 44.

NGUYEN, H., AND DONELLY, W. 2005. Hair animation and rendering in the nalu demo. *GPU Gems 2*, 361–380.

SINTORN, E., AND ASSARSSON, U. 2008. Real-time approximate sorting for self shadowing and transparency in hair rendering. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 157–162.

STANEKER, D., BARTZ, D., AND MEISSNER, M. 2003. Improving occlusion query efficiency with occupancy maps. In *PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, IEEE Computer Society, Washington, DC, USA, 15.

WARD, K., BERTAILS, F., KIM, T.-Y., MARSCHNER, S. R., CANI, M.-P., AND LIN, M. 2007. A survey on hair modeling: Styling, simulation, and rendering. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 13*, 2 (Mar-Apr), 213–34. To appear.

YUKSEL, C., AND KEYSER, J. 2008. Deep opacity maps. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2008) 27*, 2, 675–680.

ZINKE, A., SOBOTTKA, G., AND WEBER, A. 2004. Photorealistic rendering of blond hair. In *Vision, Modeling, and Visualization (VMV04)*, 191–198.

ZINKE, A., YUKSEL, C., WEBER, A., AND KEYSER, J. 2008. Dual scattering approximation for fast multiple scattering in hair. *ACM Trans. Graph. 27*, 3, 1–10.