Real-Time Hair Filtering with Convolutional Neural Networks

Real-Time Hair Filtering with Convolutional Neural Networks

ROC R. CURRIUS, ULF ASSARSSON, and ERIK SINTORN, Chalmers Institute of Technology, Sweden

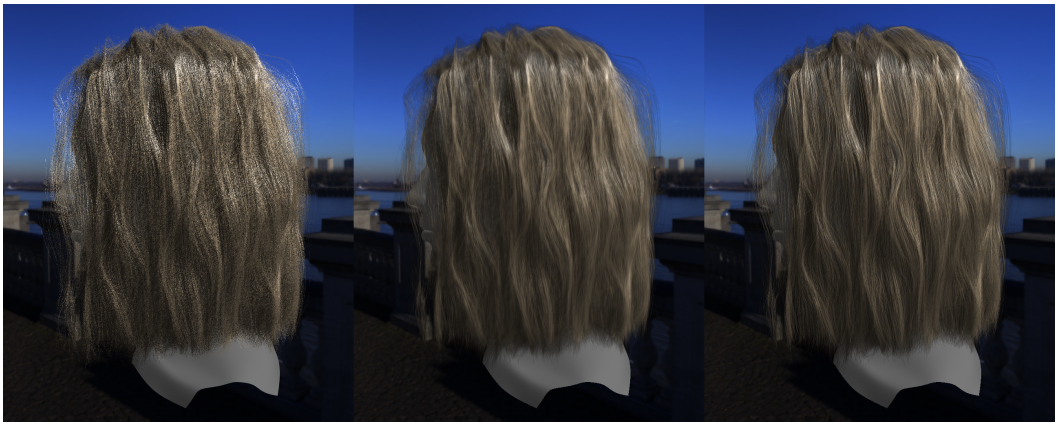

Fig. 1. From left to right: hair rendered with stochastic transparency, filtered using our method, and the ground truth.

Rendering of realistic-looking hair is in general still too costly to do in real-time applications, from simulating the physics to rendering the fine details required for it to look natural, including self-shadowing.

We show how an autoencoder network, that can be evaluated in real time, can be trained to filter an image of few stochastic samples, including self-shadowing, to produce a much more detailed image that takes into account real hair thickness and transparency.

CCS Concepts: • **Computing methodologies** → **Rendering**; *Image processing*; **Neural networks**; **Antialiasing**.

Additional Key Words and Phrases: hair, real-time, transparency, filtering, neural networks

ACM Reference Format:
Roc R. Currius, Ulf Assarsson, and Erik Sintorn. 2022. Real-Time Hair Filtering with Convolutional Neural Networks. *Proc. ACM Comput. Graph. Interact. Tech.* 5, 1, Article 15 (May 2022), 15 pages. https://doi.org/10.1145/3522606

1 INTRODUCTION

Rendering realistic hair and fur in real time is still an unsolved problem. An average human head has on the order of 100,000 hair strands, and rasterizing that much geometry has only recently become feasible in real time [Tafuri 2019]. Hair fibers, being semi-transparent, scatter light in a complex manner [Marschner et al. 2003], and while approximations exist [Zinke et al. 2008], rendering hair with correct indirect lighting is still only possible in off-line renderers.

Authors' address: Roc R. Currius, roc.ramon@chalmers.se; Ulf Assarsson, uffe@chalmers.se; Erik Sintorn, erik.sintorn@chalmers.se, Chalmers Institute of Technology, Gothenburg, Sweden.

Even direct lighting from a single light source is complex due to hair strands being extremely thin (15-200 μm). Common practice has until recently been to render a simplified textured geometry that represents several strands. Unfortunately, this method is usually quite noticeable; requires a lot of work from artists, and the simplified mesh is difficult to animate realistically. Thus, recently, the industry has turned to rendering strand-based hair [de Rousiers et al. 2020; Tafuri 2019], but aliasing remains a serious problem.

To avoid aliasing through supersampling, hundreds of samples per pixel would be required. This, and the fact that hair fibers are somewhat transparent, has led to approximating hair strands as thicker semi-transparent lines and resolving the image with alpha-compositing. Similarly, light-visibility can then be evaluated with Shadow Map [Williams 1978] techniques. However, alpha compositing traditionally requires fragments to be processed in back-to-front order, and most *Order Independent Transparency* (OIT) techniques are either very expensive or give insufficient quality for the high depth complexity of hair [Kern et al. 2021; Münstermann et al. 2018; Salvi et al. 2011].

Many stochastic transparency methods have hair strands randomly sampled by discarding fragments based on their transparency [Enderton et al. 2011], which leads to unbiased but noisy alpha compositing, unless a large amount of samples are taken.

In this paper, we suggest a method for denoising the results of Stochastic Transparency, which allows for fast rendering of very complex hair geometry, without noise, while maintaining high frequency details. Similarly to recent work on denoising, e.g. path-traced indirect illumination [Chaitanya et al. 2017], we train a U-Net [Ronneberger et al. 2015] with skip connections to reconstruct a high-quality result from stochastically rendered input data. Our method achieves high-quality close-up results, including shadows, at over 60 fps at a resolution of 1024x1024 pixels.

2 PREVIOUS WORK

2.1 Hair Rendering

Kajiya and Kay showed a first attempt at approximating a transfer function for hair [Kajiya and Kay 1989]. More physically accurate models have since been suggested [Marschner et al. 2003; Sadeghi et al. 2010; Zinke et al. 2008]. These models produce high-quality results for off-line rendering, but may be too computationally expensive for real-time applications. Several real-time approximations have been suggested [Karis 2016; Scheuermann 2004a]. As our proposed method for transparency is mostly orthogonal to the shading model used, we use the simple phenomenological model proposed by Scheuermann.

Fully evaluating the indirect illumination in hair is still much too computationally expensive for real-time applications, but treating hair as being completely opaque leads to very unrealistic results [Sintorn and Assarsson 2009]. A common compromise is to render hair as semi-transparent, using alpha-blending, both for primary rays and shadows. Unfortunately, most *Order Independent Transparency* (OIT) methods are very inefficient in cases where depth complexity can be very high. For this purpose, Sintorn and Assarsson suggest a method for sorting line segments on the GPU [Sintorn and Assarsson 2008] which, however, can be inefficient for complex hair geometry. The same authors later suggest approximating a per-pixel visibility function [Sintorn and Assarsson 2009], however this method only works when all fragments have the same opacity, and quality deteriorates when the fragments are unevenly distributed. Adaptive Transparency [Salvi et al. 2011] is a similar method, except that the visibility function is more accurate due to adaptively minimizing the error while rendering. Unfortunately, it has unbounded memory requirements on current hardware and is quite costly due to the large amount of data that needs to be saved for each pixel. The method by Münstermann et al. also estimates a visibility function but with power, or trigonometric, moments [Münstermann et al. 2018]. This results in a low frequency visibility

function and is found by Kern et al. to frequently over or under estimate visibility [Kern et al. 2021]. Maule et al. present a comprehensive survey of OIT methods [Maule et al. 2011].

Rendering shadows cast by transparent objects is similarly difficult. An early method, intended for off-line rendering, is Deep Shadow Maps [Lokovic and Veach 2000], in which an A-buffer is compressed to a piecewise linear visibility function per pixel. A real-time alternative, Opacity Shadow Maps [Kim and Neumann 2001], stores discrete functions in a 3D texture, and in Deep Opacity Maps [Yuksel and Keyser 2008] the depth resolution is improved by maintaining a depth range per pixel. At high resolutions, these methods use a large amount of memory and require several rendering passes over the geometry.

Recent approaches to rendering strand-based hair include the method by Tafuri, where alpha blending is avoided and MSAA is used to reduce aliasing [Tafuri 2019]. However, at reasonable sample counts, this method does not allow for realistically thin hair. For shadows, a few layers of Deep Opacity Maps are used. In a different approach, suggested by Jansson et al., the hair is voxelized and ray-marched each frame for distant characters, while for close-up views, the authors fall back on rasterizing alpha blended lines using a k-buffer [Bavoil et al. 2007], and the voxelized volume can also be used for self shadowing [Jansson et al. 2019].

Enderton et al. propose a method for rendering transparent objects in any order by randomly discarding each fragment based on its transparency [Enderton et al. 2011], extending the idea of *Screen-door Transparency* [Mulder et al. 1998]. Similarly, a stochastic shadow map can be created. The authors show that this method produces correct results on average and that a large number of samples per pixel can be achieved by combining this method with *Multi Sample Antialiasing* (MSAA). The technique has since been improved to allow for colored shadows [McGuire and Enderton 2011]. Laine and Karras show that variance can be reduced by applying stratification techniques [Laine and Karras 2011], but this is only suitable for geometry with few overlapping surfaces. Unless a large amount of samples are taken, these techniques still produce noisy images. This noise is even more visible in animations. To reduce temporal noise, Wyman and McGuire use a hashing algorithm based on the discretized model-space position of each fragment to determine whether it should be discarded [Wyman and McGuire 2017]. In this paper, we show that images rendered with Stochastic Transparency with just a few samples per pixel for primary-ray visibility and a single sample per pixel for shadows can be reconstructed to closely resemble the alpha-blended ground truth.

2.2 Neural Networks Hair Generation

In our suggested method, a *Convolutional Neural Network* (CNN) is used to reconstruct a plausible image from a noisy input. There exists some previous work on using machine learning to generate images of hair. For instance, Chai et al. target generating realistic-looking hair on real-life images from an input example and the desired shape, and add temporal conditioning to reduce the temporal variance [Chai et al. 2020]. In the work by Wei et al., latent-space information from processing real-life hair images is gathered and applied to the input, which consists of a processed rendering of hair strands [Wei et al. 2018]. The result is a plausible image at interactive rates, but the method is not directly applicable to scenarios where control over local lighting or compositing with a 3D scene is required. Qiu et al. instead use for input a style reference image and a processed hand drawing representing the desired shape of the hair [Qiu et al. 2019]. Similarly, Qiao and Kanai apply a GAN to transfer the hairstyle from a reference image to an arbitrarily rendered image [Qiao and Kanai 2021]. They also target reproducing realistic lighting on the hair based on the scene.

NeRF-Tex [Baatz et al. 2021] is a method for rendering fur by randomly distributing volumetric patches over a mesh. A *Neural Radiance Field* (NeRF) [Mildenhall et al. 2020] is trained for the patch, i.e. an MLP with positional encoding that can approximate the emitted radiance for a given position,

viewing direction, and light direction. The patch intersected by a primary ray is ray-marched to estimate the radiance in the viewers direction. This method yields promising results but is still much too costly for realtime applications.

2.3 Denoising

Rendering a sparsely sampled image and applying *denoising* has been studied for many decades. The most common application is to reduce variance in Monte Carlo rendered images. The objective here is to smooth the image, while retaining sharp image features, such as edges in geometry or materials. This is achieved by filtering the image in the spatial and/or temporal domain, or even filtering in path space [Keller et al. 2014]. The survey by Zwicker et al. provides a detailed list of such methods [Zwicker et al. 2015], and we will cover only those most related to our work.

For denoising at interactive or real-time framerates, a common approach is to employ edge-avoiding filters [Bauszat et al. 2015; Dammertz et al. 2010; Hanika et al. 2011; Munkberg et al. 2016]. While such methods may be applicable to reconstructing a few layered transparent surfaces with stochastic transparency, the geometry in our use-case is composed exclusively of edges, which means that very few samples will be found to lie on the same surface.

To address the problem of filtering these sparsely sampled signals, many authors make use of auxiliary features per pixel from the rendering process, e.g., positions, normals and material properties [Gastal and Oliveira 2012]. With higher dimensional input, successfully choosing filter parameters can be difficult, and therefore Kalantari et al. turn to machine learning algorithms for learning the parameters of a non-local means filter [Kalantari et al. 2015]. More recently, Bako et al. use a *Convolutional Neural Network* (CNN) to decide the best filter kernels for each point of their input features, treating diffuse and specular information in separate networks [Bako et al. 2017]. This method is intended for offline rendering and requires higher quality input and more complex networks than is currently feasible in real time. Chaitanya et al., however, describe a recurrent CNN with which they achieve high-quality denoising of inputs with very few samples per pixel, suitable for real-time rendering [Chaitanya et al. 2017]. Although the problem solved by that paper, which reconstructs contiguous geometry with sparsely sampled illumination, is quite different from ours, i.e. reconstructing sparsely sampled geometry, their ability to learn high-quality filters that can plausibly reconstruct a very sparsely sampled input image has served as a direct inspiration to us.

A related technique is *Temporal Anti Aliasing* (TAA), which has been used both for antialiasing and as a denoising method for Monte Carlo rendering. We refer the reader to a recent survey of such techniques [Yang et al. 2020]. In these methods, the current pixel sample is reprojected onto the previous frame and a *history buffer* is queried. If the color in the history buffer can be validated as representing the same surface point, it is weighted in with the new color. The output image then becomes the history buffer for the next frame. This technique has been shown to work acceptably to denoise a few layers of transparent surfaces rendered with Stochastic Transparency [Salvi 2016]. When rendering hair, however, a pixel will contain one stochastically chosen fragment of the very many that project to that pixel, and there is no one-to-one mapping to the history buffer.

3 METHOD

3.1 Overview

Rendering hair with stochastic transparency results in very noisy images if few samples are taken, and taking a sufficient number of samples is expensive, both in terms of computation and memory. Our approach is to render a few samples with stochastic transparency and train a U-net to reconstruct the original image. As input to the network, we provide not only color but also additional features such as tangents and depth, and we show that the trained network can denoise

novel views and even different hair styles with very good quality. Figure 2 shows a high level diagram describing our method.

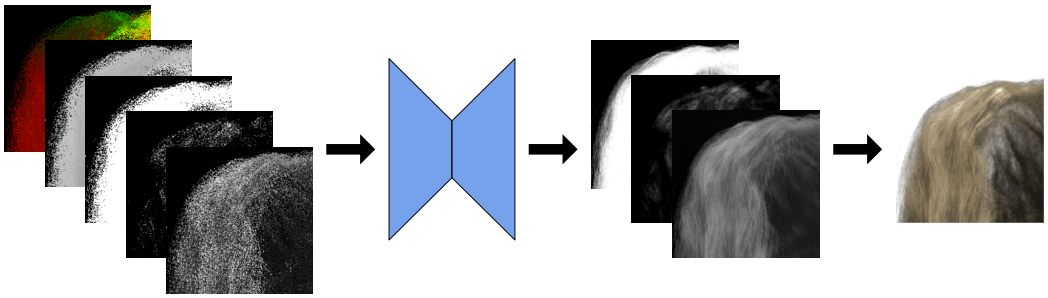

Fig. 2. Overview of our method. Stochastically sampled color factor, highlight, alpha, depth and tangents are filtered with a CNN to obtain the filtered color factor, highlight, and alpha, which are composited to produce the final image.

3.2 Input Rendering

The true light transport in hair is very complex and indirect illumination effects are important [Marschner et al. 2003]. In this paper, we consider only direct illumination and use a simplified approach [Scheuermann 2004a,b]. The formulas are shown in Figure 3. Real-time estimates to achieve indirect illumination exist [Zinke et al. 2008] but are orthogonal to our work.

$$
\begin{aligned}
\alpha_{H\text{-}L} &= \cos^{-1}\left(\frac{T}{\|T\|}\cdot\frac{P_L - P}{\|P_L - P\|}\right)\\
F_d &= \sin(\alpha_{H\text{-}L})\\
F_R &= \sin(\alpha_{H\text{-}L} + \alpha_R)^{250}\\
F_{TRT} &= \sin(\alpha_{H\text{-}L} + \alpha_{TRT})^{80}\\
L_o &= L_S\left(C_H\left(F_d + F_{TRT}\right) + F_R\right)
\end{aligned}
$$

Fig. 3. Formula for shading hair. F_d is the diffuse factor; F_R and F_{TRT} are the *Reflected* and the *Reflected-Transmitted-Reflected* specular factors [Scheuermann 2004a]; L_o is the outgoing radiance; T is the tangent; P_L and P are the positions of the light and the fragment, respectively; C_H is the color of the hair; L_S is the intensity of the light source; α_R and α_{TRT} are the specular angle shift values, for which we use values from the ranges suggested by the earlier work that inspired it [Marschner et al. 2003].

To make the network capable of handling arbitrary hair colors, we separate the outgoing radiance of the hair into two components: one factor that will be multiplied to the color of the hair, (F_d+F_{TRT}) in Figure 3, and one factor that will only depend on the intensity of the light, F_R. We refer to these as the *color factor* and *specular highlight* respectively. This separation is discussed further in Section 5. We also add a small ambient term to the colored component of the light.

The input to the network is composed of stochastically sampled values for several features (e.g. color, depth, transparency). To obtain more than one sample per pixel in a single pass, we use a coverage mask for a multisample buffer by setting or unsetting each of the bits with a probability of *alpha* [Enderton et al. 2011]. While the average of these samples is still an unbiased estimate,

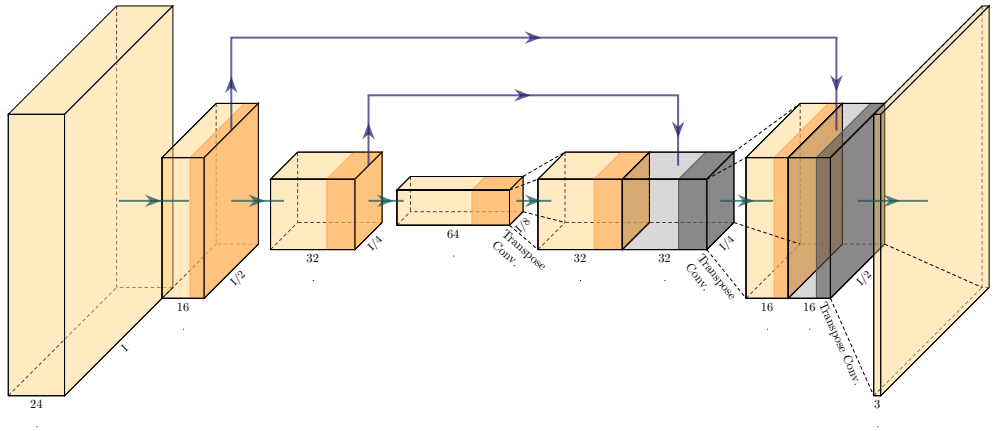

Fig. 4. Diagram of the network used to filter the hair. Convolutions are followed by a ReLU. Encoding steps are downsampling convolutions, while decoding steps are transposed convolutions set up to obtain outputs double the size of the input. All convolutions are followed by ReLU.

the result is still very noisy even at high number of samples, as can be seen in Figure 6(c). Since rendering the hair at its actual size would produce aliasing, even at high sample rates, we render the hair strands at 1px width and approximate the real size by baking the area factor into the alpha value.

The semi-transparency and thin geometry also precludes using standard shadow-mapping. However, Enderton et al. show that a stochastic shadowmap can be created in a similar manner and provides an unbiased estimator to the light visibility. In our case we, use a single sample per pixel for self-shadowing.

To reduce temporal noise, we make use of Hashed Alpha Testing [Wyman and McGuire 2017], i.e., we create random samples by applying a hash function to the model-space position of the fragment in such a way that adjacent points in screen-space will have stable random values.

3.3 Network

3.3.1 Input and Output. The input features to our network are: the color factor and specular highlight (Section 3.2), the alpha (transparency) value, the screen-space depth of the sample, and the view-space tangent (only x and y components, since they are the ones that will give the directionality information most relevant to our purpose). Reducing the number of input features improves the evaluation time of the network. However, we have found that removing either of our chosen features leads to poorer results and is not compensated with enough gain in performance. In Figure 10, we show how the presence of these features impact the MSE of the training.

The network is trained to reconstruct the color factor, the specular highlight, and the alpha. To compose the final image, we multiply the color factor by the color of the hair, add the predicted highlight, and blend with the background using the reconstructed alpha.

3.3.2 Architecture. The network is composed of several downsampling convolutions, each reducing the resolution to half of the input by using a stride of 2. These are followed by the same number of transpose convolutions, each doubling the resolution of their input, to upsample the encoded data back to the original size. We use the downsampling and upsampling properties of the convolutions to avoid using pooling and unpooling layers and thus reduce the evaluation time. As in previous

work, to improve quality of the output of the upsampling steps, skip connections are added between each pair of down and upsampling layers except for the first one. The input to the network is not skipped, as that would require an extra one-to-one convolution which is quite expensive, and we have not found that this provides much benefit in quality. All convolutions and transpose convolutions are followed by a ReLU, except for the output layer. Figure 4 shows a diagram of the layers and connections of the network.

The output of the last convolution is treated specially for each of the channels, because each represents different kinds of information that has to fit different ranges. Specifically, the output corresponding to the alpha is clipped to be between 0 and 1, using $y = min(1, max(0, x))$; the color factor output is modified by $y = x^3/32 + x/5 + 1$, so as to provide more precision around 1; the specular highlight is only clipped to be positive by $y = max(0, x)$. In these formulas, x represents the corresponding output of the network, and y corresponds to the value used in the formula for the compositing of the final image.

We use the MSE of outputs as our loss function. As we do not care about the value of the color or highlight functions where they are invisible, we premultiply the color outputs with alpha prior to calculating the MSE. As the structure of the hair is important, we additionally add the MSE of image gradients as a secondary loss.

3.3.3 Training and Validation. To train the network, we create several hundreds of images for two different styles of hair (straight and wavy, except for Figure 9), taken from randomised distances and orientations of the camera and light. Figure 12 shows how the network generalises well for different light directions thanks to this. We set aside 10% of these images to be used as the validation set to verify the correct training of the network.

The input training data is obtained by rendering the hair with stochastic transparency, producing multisampled images with the different features that the network will receive. The target training data is composed of images rendered at very high resolution and downsampled to the same size as the input. The hair translucency is approximated by averaging images rendered with stochastic transparency in the supersampled resolution, until converged.

The training itself is performed using the PyTorch library. Our implementation takes between 6 and 12 hours to converge, depending on network size and number of input features. The trained parameters are then exported to be used in the real-time application.

3.3.4 Inference. For the real-time implementation, we use a combination of OpenGL and CUDA with cuDNN. First, the input features are rendered into OpenGL multisampled color buffers. Then, the result is moved to cuDNN tensors and the convolutional network is applied. The resulting tensors are copied back to an OpenGL texture to be composed into the final image.

We make use of the convolution backwards algorithm in cuDNN to implement the transposed convolution layers, as CuDNN does not provide a specific API for transposed convolutions.

4 RESULTS

Our experiments are run on an Nvidia RTX 2080, for images of 1024x1024 pixels. The inference time of the network is proportional to the resolution, and so the numbers presented here are chosen to represent close-ups at HD resolution.

We release the source code used to evaluate the results presented at https://gitlab.com/ror3d/hair-filtering-cnn.

In order for the network to optimally use the GPU's tensor cores for acceleration, we keep both convolution parameters and data tensors as 16-bit floating point values, stored as NHWC, and the number of channels of the intermediate tensors as multiples of 8.

Table 1 details the variants of the network we use for the presented results.

Name	Samples per Pixel	Down-sampling layers	Channels per layer
Base	4 spp	3	32 > 64 > 128
Base@1spp	**1 spp**	3	32 > 64 > 128
Base@2spp	**2 spp**	3	32 > 64 > 128
Large	4 spp	3	**64 > 128 > 256**
Small	4 spp	3	**16 > 32 > 64**
Shallow	4 spp	**2**	32 > 64
Deep	4 spp	**4**	32 > 64 > 128 > 256

Table 1. The different variations on the input and architecture used in the various results tables. All networks have the same number of upsampling layers as downsampling layers, with a skip layer between each pair, with the exception of the first layer. The number of inputs depends on the number of samples per pixel. The output is always 3 channels.

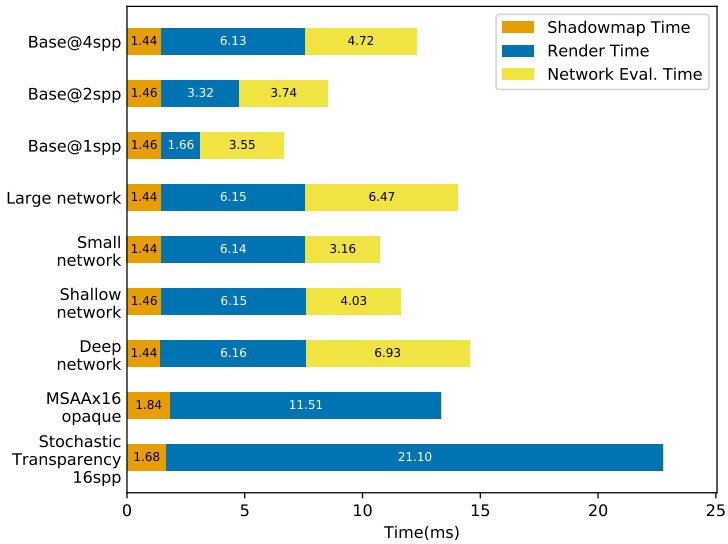

Fig. 5. Performance timings for the execution of the network during real-time evaluation (*Network Eval. Time*), and the time to render the input for different network sizes (*Render Time*). Also included is the time taken for 16 samples of pure stochastic transparency, and for rendering with only MSAA, disregarding transparency. *Shadowmap Time* is the time taken to render the stochastic shadowmap at a resolution of 1024x1024, 1SPP.

Figure 5 shows the computation times of our method. The time required to evaluate the network is mostly dependent on the size of the network, while the time taken to render the input depends mostly on the number of fragments generated (constant in our experiments) and the number of stochastic samples per pixel. We find that using 4 samples per pixel and our **Base** network configuration gives very compelling results and is about twice as fast as Stochastic Transparency with 16 samples per pixel, which still produces a noisy output.

In Figure 6, we compare our method to Stochastic Transparency and, for completeness, to rendering without transparency at high MSAA rates. While stochastic transparency alone converges to the ground truth with increasing number of samples, even 16 samples per pixel produces a noisy output. Disregarding transparency and relying on MSAA alone is much faster than stochastic

transparency (due to hi-Z culling not being available when discarding samples, and the cost of computing the hashes), but the hair looks opaque and does not converge to the ground truth. Our network can reconstruct acceptable images with only 1 sample. However, we find that 4 samples is a good performance/quality trade-off.

(a) Stochastic 1spp (3.5ms) (b) Stochastic 4spp (6.5ms) (c) Stochastic 16spp (23ms) (d) MSAAx16 (opaque) (13.5ms)

(e) Base@1spp (6.5ms) (f) Base@2spp (8.5ms) (g) Base@4spp (12.5ms) (h) Ground Truth (200ms)

Fig. 6. Result of rendering the hair using different methods: (a), (b), and (c) use stochastic transparency at different SPP; (d) is the hair rendered with MSAAx16 without transparency; (e), (f), and (g) are the results of our network for different SPP; (h) is the super-sampled reference image. In parentheses is the total time to render a frame.

Figure 9 shows how the network trained for a single hairstyle can generalise to other hairstyles, with better or worse results depending on how similar the hairstyles are to the one used for training, and the result of using all hairstyles in training.

4.1 Network configurations

As can be seen in Figure 7, which compares results for networks of different sizes, the number of convolution layers determines the effective filter size. We do not find much visible improvement with more than 3 convolution and corresponding deconvolutions. Conversely, in Figure 8, it can be seen that increasing the number of channels per layer tends to improve the visual results, at the cost of inference time. We have found the skip connections in the inner layers to be necessary to get good quality in the results.

4.2 Input parameters

In Figure 10, we show the convergence of the network for different sets of input features. Providing tangent information is vital for good results, while depth and alpha give relatively low improvements in MSE. We find that all three features provide large visual improvements to the image, however.

(a) Shallow (12ms) (b) Base (12.5ms) (c) Deep (14.5ms)

Fig. 7. Results for different sizes of the network. Details for each configuration are in Table 1.

(a) Small (11ms) (b) Base (12.5ms) (c) Large (14.5ms)

Fig. 8. Results for different numbers of layers in the network. Details for each configuration are in Table 1.

5 DISCUSSION AND LIMITATIONS

The simplification we use for separating the components of the radiance as only 3 values per pixel would not be applicable if the lighting setup was significantly more complex; using multiple differently colored lights, for instance, is not possible, as there is no way to differentiate them. Such cases would require to either evaluate the network several times, once for each color, or to train the network to receive and produce RGB values for color and highlight.

Similarly, the presented algorithm is only intended to work with hair of uniform color, but note that, thanks to the way the color is accounted for, the same network can be used to render any hair color without retraining, as demonstrated in Figure 11. Training for 3-color channels for the input and output of the network, which would allow for multi-colored hair, increases complexity, and we have not found it to work straightforwardly, but it might be interesting to further explore in future work.

A remaining problem with the method is temporal stability. We originally attempted a recurrent architecture as suggested by Chaitanya et al. [Chaitanya et al. 2017], but that provided very little improvement at a high cost. Temporal reprojection is not easily applied either, since the high frequency geometry means that a large amount of the samples are invalid due to occlusion when reprojected. The hashed alpha method significantly improves temporal stability. However, as can be seen in the accompanying video, some flickering remains.

Fig. 9. Results of training the network for a single hairstyle and evaluating it for different hairstyles. Each row has the network parameters trained for the corresponding hairstyle.

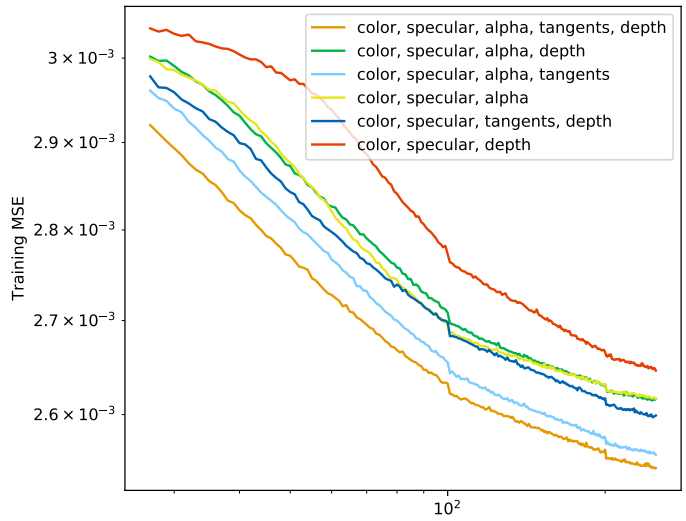

Fig. 10. Plot of MSE during training up to 250 epochs for different sets of the input features to the network.

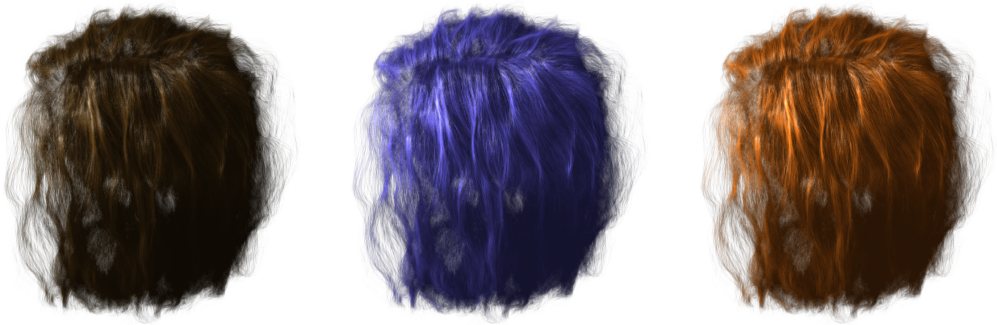

Fig. 11. As the values produced by the network are independent of color, and used as a multiplicative factor to the color of the hair, the same network parameters can be used for any uniform hair color.

6 FUTURE WORK

As mentioned in the limitations section, the presented method requires uniformly colored hair across the same mesh. An improvement would be to allow for any color variation for the hair and the light, including color gradients in the hair, by using RGB channels or some intermediate color space. There is the possibility that the network would need to be too large for good performance with current hardware to handle such input, albeit the first issue to address would be obtaining the larger amount of training data with enough variation that would be needed to train this version of the network.

Rendering the stochastic input constitutes a large part of the total frame time (Figure 5). In order to achieve higher frame rates, as well as less memory usage, the mesh could potentially be simplified while having the network still produce good-looking results. Other ways of obtaining the stochastic samples that do not require processing each hair strand individually could perhaps also

Fig. 12. Hair illuminated from different light directions.

work, such as a parametric probability distribution that determines the depth in the hair volume at which the view is blocked.

ACKNOWLEDGMENTS

We would like to thank Yuksel for the hair meshes used throughout this paper [Yuksel [n. d.]].

This work was supported by the Swedish Research Council under Grant 2014-4559, and 2017-05060.

REFERENCES

Hendrik Baatz, Jonathan Granskog, Marios Papas, Fabrice Rousselle, and Jan Novák. 2021. NeRF-Tex: Neural Reflectance Field Textures. In *Eurographics Symposium on Rendering - DL-Only Track*. The Eurographics Association, 13. https://doi.org/10.2312/sr.20211285

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan NováK, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Trans. Graph.* 36, 4 (July 2017), 1–14. https://doi.org/10.1145/3072959.3073708

P. Bauszat, M. Eisemann, S. John, and M. Magnor. 2015. Sample-Based Manifold Filtering for Interactive Global Illumination and Depth of Field. *Comput. Graph. Forum* 34, 1 (feb 2015), 265–276. https://doi.org/10.1111/cgf.12511

Louis Bavoil, Steven P. Callahan, Aaron Lefohn, João L. D. Comba, and Cláudio T. Silva. 2007. Multi-Fragment Effects on the GPU Using the k-Buffer. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (I3D '07)*. Association for Computing Machinery, New York, NY, USA, 97–104. https://doi.org/10.1145/1230100.1230117

Menglei Chai, Jian Ren, and Sergey Tulyakov. 2020. Neural Hair Rendering. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Vol. 12363. Springer International Publishing, Cham, 371–388. https://doi.org/10.1007/978-3-030-58523-5_22

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Transactions on Graphics* 36, 4 (July 2017), 1–12. https://doi.org/10.1145/3072959.3073601

Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik P. A. Lensch. 2010. Edge-Avoiding À-Trous Wavelet Transform for Fast Global Illumination Filtering. In *Proceedings of the Conference on High Performance Graphics* (Saarbrucken, Germany) *(HPG '10)*. Eurographics Association, Goslar, DEU, 67–75.

Charles de Rousiers, Gaelle Morand, and Michael Forot. 2020. An Early Look at Next-Generation Real-Time Hair and Fur. https://www.unrealengine.com/en-US/tech-blog/an-early-look-at-next-generation-real-time-hair-and-fur.

Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. 2011. Stochastic Transparency. *IEEE Transactions on Visualization and Computer Graphics* 17 (2011), 1036–1047. https://doi.org/10.1109/TVCG.2010.123

Eduardo S. L. Gastal and Manuel M. Oliveira. 2012. Adaptive Manifolds for Real-Time High-Dimensional Filtering. *ACM Trans. Graph.* 31, 4, Article 33 (jul 2012), 13 pages. https://doi.org/10.1145/2185520.2185529

Johannes Hanika, Holger Dammertz, and Hendrik Lensch. 2011. Edge-optimized À-trous wavelets for local contrast enhancement with robust denoising. *Comput. Graph. Forum* 30, 7 (2011), 1879–1886. https://doi.org/10.1111/j.1467-8659.2011.02054.x

Erik Sven Vasconcelos Jansson, Matthäus G. Chajdas, Jason Lacroix, and Ingemar Ragnemalm. 2019. Real-Time Hybrid Hair Rendering. *Eurographics Symposium on Rendering - DL-only and Industry Track* (2019). https://doi.org/10.2312/SR.20191215

J. T. Kajiya and T. L. Kay. 1989. Rendering Fur with Three Dimensional Textures *(SIGGRAPH '89)*. Association for Computing Machinery, New York, NY, USA, 271–280. https://doi.org/10.1145/74333.74361

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. *ACM Trans. Graph.* 34, 4, Article 122 (jul 2015), 12 pages. https://doi.org/10.1145/2766977

Brian Karis. 2016. Physically Based Hair Shading in Unreal.

Alexander Keller, Ken Dahm, and Nikolaus Binder. 2014. Path Space Filtering. In *ACM SIGGRAPH 2014 Talks* (Vancouver, Canada) *(SIGGRAPH '14)*. Association for Computing Machinery, New York, NY, USA, Article 68, 1 pages. https://doi.org/10.1145/2614106.2614149

Michael Kern, Christoph Neuhauser, Torben Maack, Mengjiao Han, Will Usher, and Rüdiger Westermann. 2021. A Comparison of Rendering Techniques for 3D Line Sets With Transparency. *IEEE Transactions on Visualization and Computer Graphics* 27, 8 (Aug. 2021), 3361–3376. https://doi.org/10.1109/TVCG.2020.2975795

Tae-Yong Kim and Ulrich Neumann. 2001. Opacity Shadow Maps. In *Rendering Techniques 2001*. Springer Vienna, 177–182. https://doi.org/10.1007/978-3-7091-6242-2_16

Samuli Laine and Tero Karras. 2011. Stratified Sampling for Stochastic Transparency. *Computer Graphics Forum* 30, 4 (June 2011), 1197–1204. https://doi.org/10.1111/j.1467-8659.2011.01978.x

Tom Lokovic and Eric Veach. 2000. Deep Shadow Maps. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '00*. ACM Press, Not Known, 385–392. https://doi.org/10.1145/344779.344958

Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan. 2003. Light Scattering from Human Hair Fibers. *ACM Trans. Graph.* 22, 3 (July 2003), 780–791. https://doi.org/10.1145/882262.882345

Marilena Maule, João L. D. Comba, Rafael P. Torchelsen, and Rui Bastos. 2011. A Survey of Raster-Based Transparency Techniques. *Computers & Graphics* 35, 6 (Dec. 2011), 1023–1034. https://doi.org/10.1016/j.cag.2011.07.006

Morgan McGuire and Eric Enderton. 2011. Colored Stochastic Shadow Maps. In *Symposium on Interactive 3D Graphics and Games (I3D '11)*. Association for Computing Machinery, New York, NY, USA, 89–96. https://doi.org/10.1145/1944745.1944760

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *arXiv:2003.08934 [cs]* (Aug. 2020). arXiv:2003.08934 [cs]

J.D. Mulder, F.C.A. Groen, and J.J. van Wijk. 1998. Pixel masks for screen-door transparency. In *Proceedings Visualization '98 (Cat. No.98CB36276)*. IEEE, 351–358,. https://doi.org/10.1109/VISUAL.1998.745323

Jacob Munkberg, Jon Hasselgren, Petrik Clarberg, Magnus Andersson, and Tomas Akenine-Möller. 2016. Texture space caching and reconstruction for ray tracing. *ACM Trans. Graph.* 35, 6 (2016), 1–13. https://doi.org/10.1145/2980179.2982407

Cedrick Münstermann, Stefan Krumpen, Reinhard Klein, and Christoph Peters. 2018. Moment-Based Order-Independent Transparency. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1 (July 2018), 7:1–7:20. https://doi.org/10.1145/3203206

Zhi Qiao and Takashi Kanai. 2021. A GAN-Based Temporally Stable Shading Model for Fast Animation of Photorealistic Hair. *Comp. Visual Media* 7, 1 (March 2021), 127–138. https://doi.org/10.1007/s41095-020-0201-9

H. Qiu, C. Wang, H. Zhu, X. Zhu, J. Gu, and X. Han. 2019. Two-Phase Hair Image Synthesis by Self-Enhancing Generative Model. *Computer Graphics Forum* 38, 7 (2019), 403–412. https://doi.org/10.1111/cgf.13847

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv:1505.04597 [cs.CV]

Iman Sadeghi, Heather Pritchett, Henrik Wann Jensen, and Rasmus Tamstorf. 2010. An Artist Friendly Hair Shading System. *ACM Trans. Graph.* 29, 4 (July 2010), 56:1–56:10. https://doi.org/10.1145/1778765.1778793

Marco Salvi. 2016. An Excursion In Temporal Supersampling. In Game Developers Conference.

Marco Salvi, Jefferson Montgomery, and Aaron Lefohn. 2011. Adaptive Transparency. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics (HPG '11)*. Association for Computing Machinery, New York, NY, USA, 119–126. https://doi.org/10.1145/2018323.2018342

Thorsten Scheuermann. 2004a. Hair Rendering and Shading. https://developer.amd.com/wordpress/media/2012/10/Scheuermann_HairRendering.pdf

Thorsten Scheuermann. 2004b. Practical Real-Time Hair Rendering and Shading. In *ACM SIGGRAPH 2004 Sketches* (Los Angeles, California) *(SIGGRAPH '04)*. Association for Computing Machinery, New York, NY, USA, 147. https://doi.org/10.1145/1186223.1186408

Erik Sintorn and Ulf Assarsson. 2008. Real-Time Approximate Sorting for Self Shadowing and Transparency in Hair Rendering. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games - SI3D '08*. ACM Press, Redwood City, California, 157. https://doi.org/10.1145/1342250.1342275

Erik Sintorn and Ulf Assarsson. 2009. Hair self shadowing and transparency depth ordering using occupancy maps. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (Boston, Massachusetts) *(I3D '09)*. ACM, New

York, NY, USA, 67–74. https://doi.org/10.1145/1507149.1507160

Sebastian Tafuri. 2019. Strand-Based Hair Rendering in Frostbite. https://doi.org/10.1145/3305366.3335035

Lingyu Wei, Liwen Hu, Vladimir Kim, Ersin Yumer, and Hao Li. 2018. Real-Time Hair Rendering Using Sequential Adversarial Networks. In *Computer Vision – ECCV 2018*. Vol. 11208. Springer International Publishing, Cham, 105–122. https://doi.org/10.1007/978-3-030-01225-0_7

Lance Williams. 1978. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques (SIGGRAPH '78)*. Association for Computing Machinery. https://doi.org/10.1145/800248.807402

Chris Wyman and Morgan McGuire. 2017. Hashed Alpha Testing. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, San Francisco California, 1–9. https://doi.org/10.1145/3023368.3023370

Lei Yang, Shiqiu Liu, and Marco Salvi. 2020. A Survey of Temporal Antialiasing Techniques. *Comput. Graph. Forum* 39, 2 (2020), 607–621. https://doi.org/10.1111/cgf.14018

Cem Yuksel. [n. d.]. HAIR Model Files - Cem Yuksel. http://www.cemyuksel.com/research/hairmodels/

Cem Yuksel and John Keyser. 2008. Deep Opacity Maps. *Computer Graphics Forum* 27, 2 (2008), 675–680. https://doi.org/10.1111/j.1467-8659.2008.01165.x

Arno Zinke, Cem Yuksel, Andreas Weber, and John Keyser. 2008. Dual Scattering Approximation for Fast Multiple Scattering in Hair. In *ACM SIGGRAPH 2008 Papers*. ACM Press, Los Angeles, California, 1. https://doi.org/10.1145/1399504.1360631

Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and Sung-Eui Yoon. 2015. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)* 34, 2 (May 2015), 667–681. https://doi.org/10/f7k6kj