Thesis for The Degree of Licentiate of Engineering

Realistic Real-Time Rendering of Global Illumination and Hair through Machine Learning Precomputations

Roc Ramon Currius

Division of Computer and Network Systems
Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden, 2022

Realistic Real-Time Rendering of Global Illumination and Hair through Machine Learning Precomputations

Roc Ramon Currius

ISSN 1652-876X
Department of Computer Science and Engineering
Division of Computer and Network Systems
Computer Graphics Research Group
Chalmers University of Technology | University of Gothenburg
SE-412 96 Göteborg, Sweden
Phone: +46(0)32 772 1000

Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2022.

"Any sufficiently advanced technology
is indistinguishable from magic."
– Arthur C. Clarke

Abstract

Realistic Real-Time Rendering of Global Illumination and Hair through Machine Learning Precomputations

Roc Ramon Currius
Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg

Over the last decade, machine learning has gained a lot of traction in many areas, and with the advent of new GPU models that include acceleration hardware for neural network inference, real-time applications have also started to take advantage of these algorithms.

In general, machine learning and neural network methods are not designed to run at the speeds that are required for rendering in high-performance real-time environments, except for very specific and usually limited uses. For example, several methods have been developed recently for denoising of low quality pathtraced images, or to upsample images rendered at lower resolution, that can run in real-time.

This thesis collects two methods that attempt to improve realistic scene rendering in such high-performance environments by using machine learning.

Paper I presents a neural network application for compressing surface lightfields into a set of unconstrained spherical gaussians to render surfaces with global illumination in a real-time environment.

Paper II describes a filter based on a small convolutional neural network that can be used to denoise hair rendered with stochastic transparency in real time.

Keywords

Real-time rendering, Global Illumination, Lightfields, Hair Rendering, Realistic Rendering, Neural Networks, Machine Learning

Acknowledgments

First I would like to thank my supervisor Erik Sintorn for all the help, support and guidance he has provided throughout these years. I would also like to thank my co-supervisor Ulf Assarsson for providing invaluable input on the projects and papers.

I want to also thank my colleagues from the research group — Alexandra, Dan, Mads, Sverker, and from the department and student council — Irene, Linda, Nadja, Carlo, for their help and moral support.

Finally, I need to thank family and friends from back in Spain, whose moral support has been invaluable throughout the years.

This thesis and the works included in it were supported by the Swedish Research Council under Grant 2014-4559 and 2017-05060.

List of Publications

Appended publications

This thesis is based on the following publications:

[**Paper I**] **Roc R. Currius**, Dan Dolonius, Ulf Assarsson, Erik Sintorn, *Spherical Gaussian Light-Field Textures for Fast Precomputed Global Illumination*
Computer Graphics Forum 39, 2 (May 2020), 133-146.

[**Paper II**] **Roc R. Currius**, Erik Sintorn, *Real-Time Hair Filtering with Convolutional Neural Networks*
Submitted, under review.

Contents

Part I

Summary

Chapter 1

Introduction

When a computer has to display something on the screen, it has to decide and set the colors of each pixel. The process of "painting" the pixels, on the screen or on an arbitrary image, is called *rendering*. This can be in order to represent any kind of information, from text to 3D scenes.

An object in a 3D scene is most usually represented as a set of triangles that follow its surface (often called a *mesh*). This is due to several reasons: on one hand it is a very simple and straightforward way to represent an arbitrary surface; on the other, and also in part because of the first one, until very recently graphics rendering hardware has been almost exclusively built and optimised to handle this specific case — *rasterisation* of triangles.

Rasterisation follows a very fast algorithm which computes the pixels that each triangle will occupy on the screen and then calculates the color for each of them individually, as depicted in Fig. 1.1. However, when rendering triangles this way, there is no readily-available information from the rest of the scene, only information included in the three vertices of the triangle, so seemingly simple things, like representing reflections, become a complex matter.

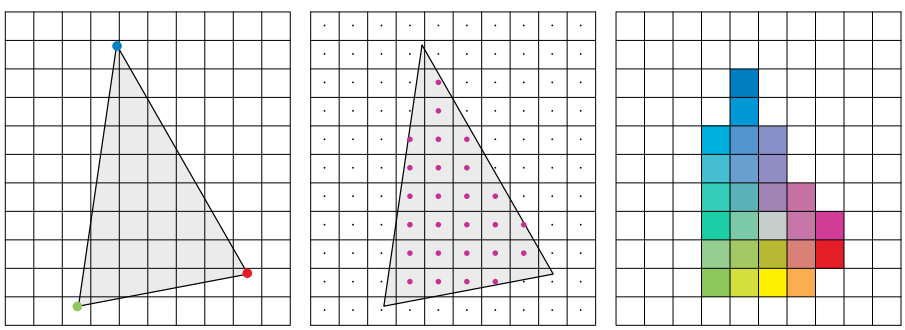


Figure 1.1: Rasterisation works by determining which pixels' centers are inside of the triangle, and then interpolating the values of the surface properties at the vertices.

Rendering of realistic scenes in real-time applications has improved immensely over the last two decades, in part thanks to many advancements in

hardware, but also thanks to new techniques being developed that can be used to more closely represent reality while achieving high performance.

In contrast, *Raytracing* techniques such as *Pathtracing*, instead of attempting to rapidly calculate the pixels that each triangle would occupy and paint them based on the triangle's local properties, take a physically-based approach by calculating how all the incoming light at every point of the scene would reflect towards the eye. To accomplish that, the *rendering equation* [14] needs to be evaluated:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\mathcal{S}^2} f(p, \omega_o, \omega_i)\, L_i(p, \omega_i)\, |\omega_i \cdot \mathbf{n}|\, d\omega_i \qquad (1.1)$$

Where L_o is the radiance outgoing from point p towards the outgoing direction ω_o, L_e is the radiance emitted at p in the outgoing direction, f is the *bidirectional distribution function*, which determines in what way (absorption, tint, scattering) the radiance coming from direction ω_i is reflected at the point towards ω_o, and $\mathbf{n}$ is the normal of the surface at the point. The integral is taken over $\mathcal{S}^2$, the surface of the 3D unit sphere, effectively integrating all directions around the point. Since light will come from reflections on other points on the scene, this becomes a recursive equation.

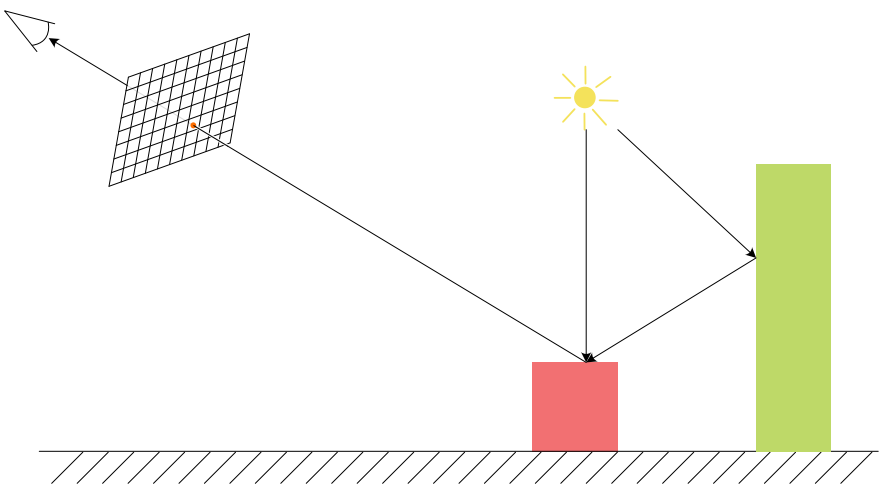


Figure 1.2: Raytracing follows the trajectory that a ray would travel from a point in the scene to the viewer's eye through the screen, and paints the pixel it crosses with the color from the point in the scene.

To calculate this, pathtracing follows the path that rays of light would have to traverse from their source to reach the viewer's eye, after having randomly bounced across the scene, as represented in Fig. 1.2. This approach, from its very definition, will already use information of the structure of the scene to calculate the final color of each pixel. The problem is that the process of finding the points where the light rays collide with the objects in the scene can be slow, making this method orders of magnitude slower than rasterising the same scene; while also producing very noisy images due to its stochastic

nature, causing it to require many samples (i.e. rays) per pixel, and so it has been usually used exclusively in offline rendering, for example in movies.

In recent years, high-end graphics hardware has started to include dedicated units for ray-tracing acceleration, but even with the use of this specialised hardware, only a few samples per pixel can be taken every frame, so aggressive simplifications of the light interactions, as well as filtering and post-processing of the generated images, are required to hide the noise.

The methods developed throughout the last decades for more realistic rendering through rasterisation are of different kinds — usage of formulas based on real-world physical observations to calculate how the light interacts with the objects in the scene to bring more realistic illumination from lights; pre-processing of the scene to *bake* some properties to be used during rasterisation allows for taking into account some structural information of the scene that would not be available otherwise, such as light occlusions to produce shadows; and post-processing steps can add further effects that also take the structure of the visible parts of the scene into account, for instance to add reflections, as well as simulate properties of the medium in which the light travels before reaching the eye, accomplishing effects such as volumetric light.

In this thesis we will explore two methods, using offline machine learning-based precomputations, that can be implemented in rasterised scenes in real-time applications to represent effects that are otherwise difficult to replicate because of time constraints, due to the amount of extra computations they would usually require.

1.1 Global Illumination

When rendering a 3D scene, we make a distinction between the illumination that comes directly from light sources, which is referred to as *Direct Illumination*, and the illumination produced from light reflected on other surfaces, which is called *Indirect Illumination*. Together, direct and indirect illumination add up to what is called *Global Illumination*. Indirect illumination is one of the effects that is recreated easily with pathtracing, from its very definition, but is very complex to reproduce using rasterisation, as it requires considering the light coming onto each surface from everywhere around it, which would either require rasterising the scene as seen from every point, or using some form of raytracing from each point to get the contributions from objects around it (basically replicating pathtracing), both approaches being too expensive.

As shown in Equation 1.1, we can define a function that determines how light rays are going to reflect on a surface, called the *Bidirectional Reflectance Distribution Function* (BRDF), corresponding to the f term in Equation 1.1 if we ignore the light transmitted through objects. This function defines how and to where a ray of light will reflect on a surface, but due to its bidirectionality, it can also be understood as what direction the light rays would need to hit the surface to be reflected in a specific direction, and how their color will be affected in every case. From this we will obtain directions that are very close to the perfect reflection direction for smooth surfaces, and a much wider range of directions for rough surfaces, as depicted in Fig. 1.3.

A common approximation for indirect illumination in rasterised scenes uses

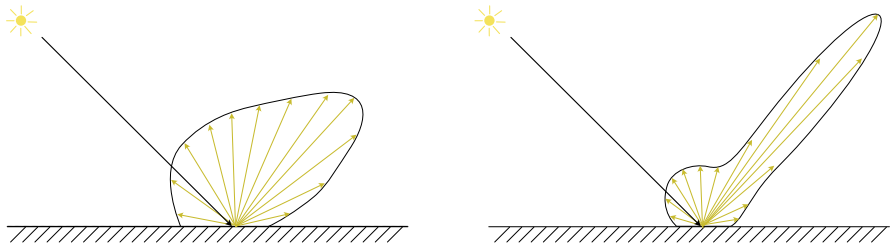


Figure 1.3: Side view of the BRDF directions for an incoming ray on rough (left) and smooth (right) materials.

reflection probes - instead of rasterising the entire surroundings of every point in the scene, a few select points are distributed on the scene, tipically chosen by the artists creating the scene, and the image of the surroundings is taken only for these points, similar to what is shown in Fig. 1.4. Then, interpolation is used for surfaces and points in-between the reflection probes. This, obviously, can give good results for surfaces that are very close to a reflection probe, but not so good for surfaces away from them. Moreover, reflections on rough surfaces need be taken into account explicitly, otherwise they would reflect images too sharply.

Figure 1.4: *Left*: A scene with a few reflection probes, displayed as spheres (only visible when building the scene, not during normal rendering of the scene). *Right*: What the map rendered for one of the reflection probes would look like.

A different approach is to distribute a much larger set of points around the scene and compress the light arriving to them from all directions in a way that can be interpolated between the points. The usual method chosen for this is to use a set of linearly independent components that can be added up, recreating the desired function. For example, a set of functions used to this end is *Spherical Harmonics*, as shown in Fig. 1.5, which work in a similar way to a Fourier Transform or Discrete Cosine Transform.

Another set of functions used to this effect is *Spherical Gaussians* (SGs) — gaussian functions constrained on the surface of the unit sphere, each pointing towards a different direction, and with independent amplitudes and sharpnesses, as depicted in Fig. 1.5(d). Equation 1.2 shows the formula for

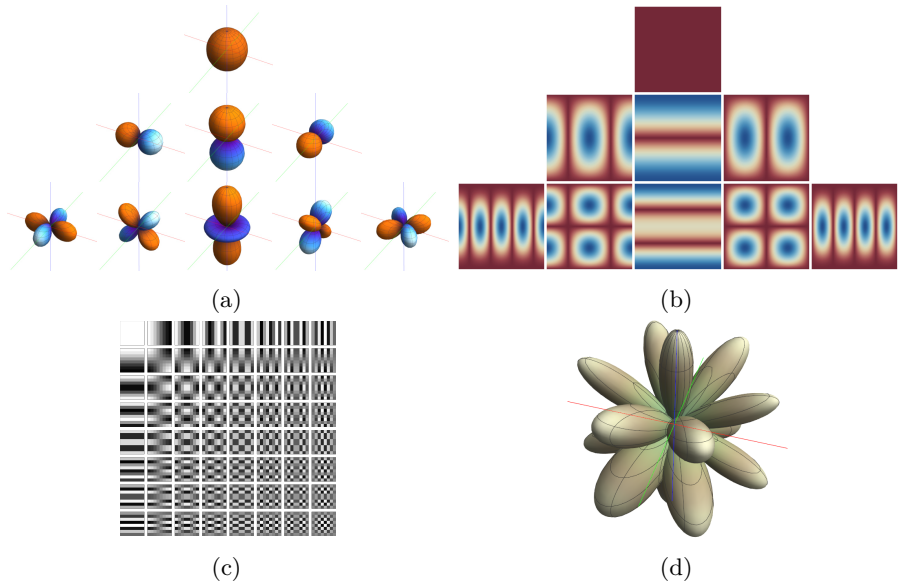


Figure 1.5: (a) and (b) show a few components of spherical harmonics, as seen in 3D, and their values on spherical coordinates, respectively. Added together they can approximate any functions in a similar way to a Discrete Cosine Transform (DCT). (c) shows the DCT components for an image of size 8x8, as used in the JPEG algorithm (*image from Wikipedia [6], public domain*). (d) shows a set of spherical gaussians with uniformly distributed directions around the origin, with randomised sharpness.

spherical gaussians, with A being the amplitude, ω being the vector direction of the gaussian, λ the sharpness of the lobe, and $\mathbf{d}$ the vector of the direction for which we want to calculate the value of the function.

$$SG(A, \omega, \lambda, \mathbf{d}) = A\, e^{\lambda\,(\omega \cdot \mathbf{d} - 1)} \tag{1.2}$$

1.2 Transparency

Realistic-looking refractions are another complex effect to reproduce, since they require keeping track of where a ray of light would enter a translucent object and where it would exit, along with the incident angles with the interacting surfaces. That is because the light is deflected from its path by an amount proportional to the incident angle with respect to the interacting surface and the refraction index between the two materials.

This is usually approximated in real-time applications by a surface that only tints the light according to the material's color, and otherwise lets it through unmodified, not changing its path at all. This works well enough for translucent objects that are flat and thin, like windows, but it quickly becomes obviously unrealistic when used for any curved surfaces or thicker objects.

When using this approximation, it is usually encoded how much a surface will let light through with a single value in the range $[0, 1]$ that represents

how opaque the material is, commonly referred to as *alpha* — a material with an alpha of 0 is completely transparent, and a material with an alpha of 1 is totally opaque.

For a given list of overlapping colors and transparencies, C_i and α_i respectively, sorted from closest to furthest, the final color C of the pixel can be calculated using the Porter-Duff **over** operator [21], as given in Equation 1.3. Fig. 1.6 exemplifies this graphically.

$$C = \alpha_1 c_1 + (1 - \alpha_1)(\alpha_2 c2 + (1 - \alpha_2)(\alpha_3 c_3 + ...)) \tag{1.3}$$

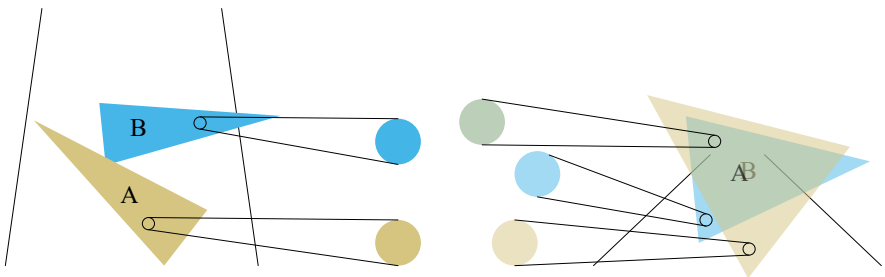


Figure 1.6: Semi-transparent object A is in front of object B, both with 50% transparency. Equation 1.3 can be used to calculate the color seen in the overlapping area.

When rasterising transparent objects, the fragments[1] for each pixel need to be rendered in order from furthest to closest to the viewer to be able to calculate the correct color.

The classical approach to do this is to sort all the triangles before rendering, but when the amount of triangles grows too large the sorting process can become too costly, and this approach only really works for non-intersecting geometry, so extra steps need to be taken to fix such geometry, further slowing the process if the geometry is dynamic.

Methods have been developed that allow for rendering without pre-sorting the triangles, generally referred to as Order Independent Transparency methods.

Most of these methods rely on some variation of the *K-buffer* [1] algorithm (which is an adaptation of the *A-buffer* [2] for bounded memory), which works by keeping a list, for each pixel on the image, of the separate values of each fragment, together with their depths, then sorting each list when all the geometry has been processed, and finally adding them together in order to produce the final color.

Another way to solve the issue is to take a probabilistic approach. The screen-door [8] method replaces transparent surfaces by a pattern of pixels that are either fully opaque or fully transparent, more or less of them depending on the transparency. As shown in Fig. 1.7, this can create undesired artifacts and visible patterning. If instead we randomly discard every fragment with a

[1] A fragment is a collection of values produced by the rasteriser for a geometry primitive, such as a triangle, corresponding to a pixel on the screen after it is processed by the shader. Multiple fragments can correspond to the same pixel, for example when there is geometry that overlaps.

probability equal to its transparency, the average of taking several samples will tend towards true value of the pixel while removing artifacts, as long as the samples are uniformly random. Fig. 1.7 also shows an example of this.

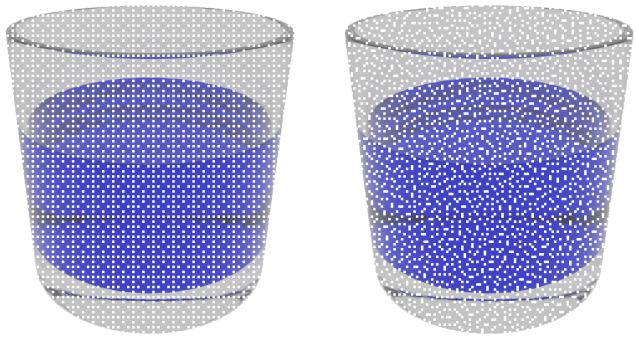


Figure 1.7: The basic screen-door transparency (*left*) uses a predefined pattern to decide which pixels are on and which off depending on the surface's transparency. Stochastic transparency (*right*) instead decides randomly for every pixel, eliminating patterning artifacts.

All these methods solve the problem for specific situations, but there is no general method that works equally well for all cases, and very complex semitransparent geometry is still hard to handle.

1.3 Rendering Hair

When trying to render realistic scenes that include people, several problems appear. Some of these problems are: skin is soft, and that is usually difficult to animate; skin also scatters light in a very specific and complex manner, which is usually quite difficult to simulate without the skin ending up looking more like rubber; eyes have a quite complex geometry if looked from close-up, and it is very easy for people to discern when eyes do not look lifelike; hair is made up of very thin and numerous translucent geometry, both things being problematic for real-time rendering. All these are open topics of research, but we will more closely concentrate on the issues with the rendering of hair.

The light transport model for hair is significantly different from that of more common surfaces. If we examine a hair strand closely, we will see that it has a non-symmetrical structure across its length. As shown in Fig. 1.8, the cross-section of a hair fiber is not circular, more similar to an ellipse (if we ignore irregularities), and the surface is composed of scales, which lends to light being reflected in a non-uniform way [15]. First the light is partly reflected on the surface of the hair, which is called the Reflected (R) term. Since hair is translucent, a lot of light is also transmitted through the surface, exiting the hair in the opposite side of the fiber, what we call the Transmitted-Transmitted term (TT). Part of the transmitted light will be reflected back on the opposite side of the fiber and exit the hair on the same side it entered, called the Transmitted-Reflected-Transmitted term (TRT), producing a secondary reflection highlight, this time tinted by the light traveling through the hair. A

common simplification of the measured BRDF from real-life hair fibers encodes these R and TRT terms as two specular lobes with angular offsets, also shown in Fig. 1.8.

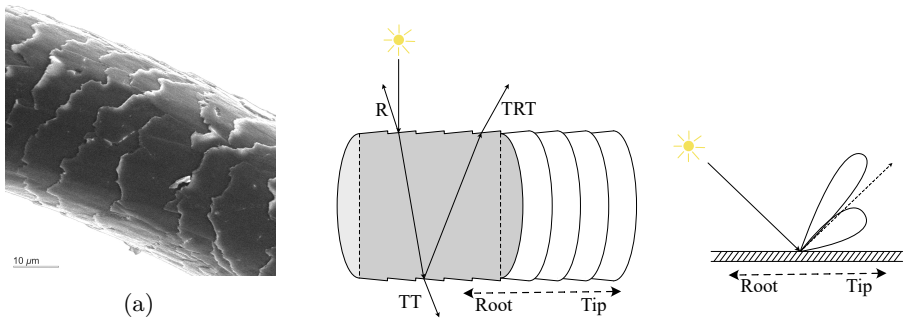


Figure 1.8: Surface of a hair fiber used for the lighting model. (a) is an electron micrograph of the surface of a hair fiber (*image by Nanjundaswamy [19] under Creative Commons BY-NC-SA 3.0 US*). (b) shows the simplification of the surface and the BRDF lobes by Marschner et al. [15].

As initially mentioned, rendering of hair through rasterisation has several other complications which make it a non-straightforward task.

On one hand, hairs are very thin, having a diameter ranging between 15 to 200 µm, which means that the real width of each hair is going to be significantly thinner than that of a single pixel. Trying to render it without taking that into account would lead to *aliasing*, as shown in Fig. 1.9. That is, when rasterising, a pixel will display a triangle only if the center of the pixel is inside the triangle, which will not always be the case if the triangles are thinner than a pixel.

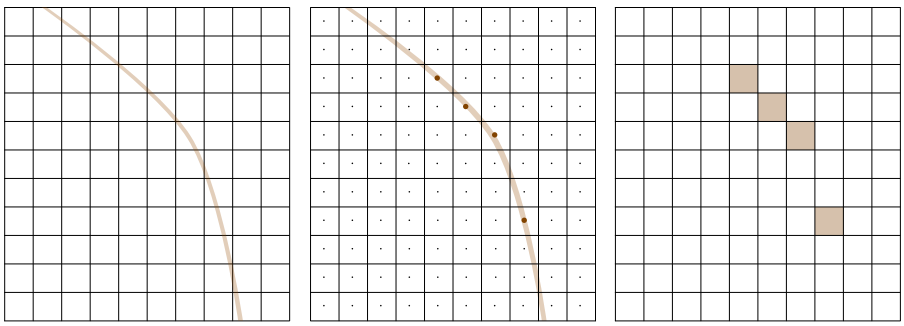


Figure 1.9: Aliasing when rendering hairs that are significantly thinner than a pixel — most pixels along the hair's path will not be shown because they do not cross the sampling point at the center of the pixel.

There is also the issue of the quantity of hair. A typical person's head has an average of 100K to 200K hairs. If we represent the hair with a few segments for each hair strand, 10 for example, that means we will have over one million segments to render every frame, which is orders of magnitude more geometry than most objects will have in a typical scene for a real-time application.

Finally there is the problem with transparency. Hair is not opaque, but translucent (more for light hair than dark hair), and that needs to be taken into account when attempting to draw hair that looks realistic.

Until recently, the most relevant of these problems has been the amount of geometry to render. The usual approach to mitigate that issue has been to use *hair cards*: simplifying the hair into a set of planes that cover the head and follow the shape of the hair, and applying textures of drawn hair on the planes, an example of which is shown in Fig. 1.10. This approach usually requires a lot of work from the artists building the model, since it is quite difficult to achieve realistic-looking results this way, and it is usually easy to detect. A benefit of this approach, which is another part of why it has been popular in the past, is that it makes the transparency issue significantly more manageable by reducing the amount of geometry to be rendered.

Figure 1.10: Hair model simplified into hair cards. It requires a lot of work from artists, and usually the use of this simplification becomes very evident (*3D model by Haynes [10] under royalty-free license*).

Thanks to the continued increase in GPU processing power, it is now possible to render large amounts of geometry in significantly less time, allowing for the use of meshes of individual hair strands instead of relying on hair cards. This makes it possible to achieve more realistic results, but brings back relevance to the other problems mentioned previously.

1.4 Neural Networks and Machine Learning

Machine learning refers to algorithms that can use existing data to improve their results, usually based on various statistical methods. A machine learning algorithm is also commonly called a *model*, which is *trained* using existing information, either labeled (matched pairs of input and expected result of the algorithm), or unlabeled (for which the algorithm is expected to find similarities in the data).

This is usually done by defining the model to be a parametrised function that is expected to approximate the expected data. The parameters of this

function are then modified in an attempt to improve how well the function fits its target shape.

These kind of techniques have been showing up in one way or another in most areas of computer science and engineering, as they allow for more generic algorithms to solve problems, and the field of computer graphics is no exception. For example, there are pathtracing algorithms that use machine learning to take better samples [4, 17, 26, 18], denoising algorithms learning shapes to better clean noise from images [3], or data-based animation [12].

1.4.1 Gradient Descent and Expectation-Maximisation

A common approach to update the parameters of a function so it better matches its expected value is by using *gradient descent*. This method relies on comparing the output of the function with the expected values, determining how different they are using an *error* or *loss function*, and determining in which way the parameters should change to make the error smaller. This can be accomplished by calculating the partial derivatives of the error function with respect to the parameters, obtaining a gradient vector, which will point towards the direction where the slope of the error function is highest. Changing the values following this gradient by a small-enoug step should then lead to the error function becoming smaller. Repeating the process enough times can bring the error function to a local minima. Fig. 1.11 exemplifies this.

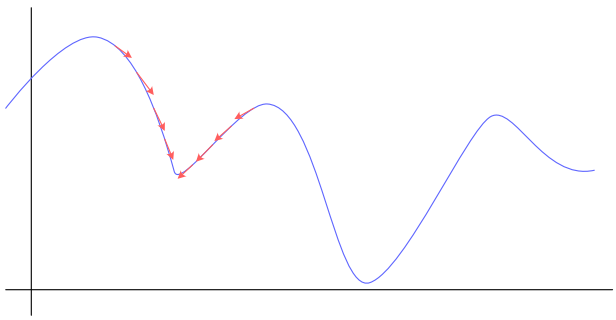


Figure 1.11: Gradient Descent algorithm: Following the direction of the gradient vector at a point converges to a local minima of the function.

If the functions to be optimised represent a probability distribution, then Expectation-Maximisation [5] can be used, which is a different approach that can usually converge to a valid set of parameters faster than gradient descent. Given a set of observed sample points, the algorithm first makes a guess about the parameters of each distribution function; then an *expectation* step is done where the probabilities that each sample point belongs to each distribution are calculated; followed by a *maximisation* step, which will find a new guess of the parameters that maximises the fit of each distribution to the samples that belong to it.

The expectation-maximisation algorithm is usually applied to fit Gaussian Mixtures, which are probability distributions made up of the sum of several Normal distributions. On the surface of a sphere, the von Mises-Fisher dis-

tribution can be used instead of normal distributions, which is a probability distribution based on the spherical gaussians mentioned in Section 1.1.

1.4.2 Neural Networks

Artificial Neural networks (NN) are a subset of machine learning algorithms that follow a rough simplification of how real neurons work. They are a composition of connected *artificial neurons* — nodes that hold information, which they obtain from their input connections from other neurons, and send it to other neurons through output connections, as shown in Fig. 1.12.

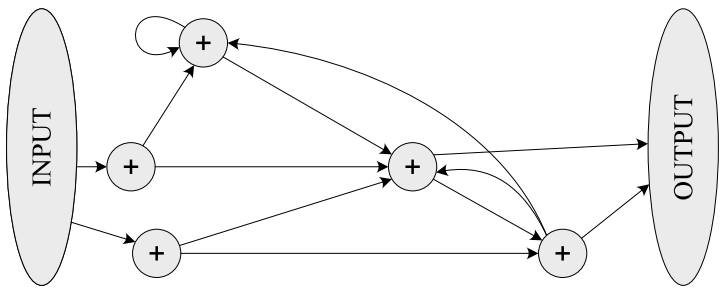


Figure 1.12: Several networks with several connections between them, representing the way the data from each will flow to each other. When updating them, the incoming values for each neuron will be added together to create its new value.

More specifically, each neuron is represented by a single value, and each connection has a factor that is multiplied by the value at its origin. A neuron's value, then, is obtained by adding-up all the products of input values and factors.

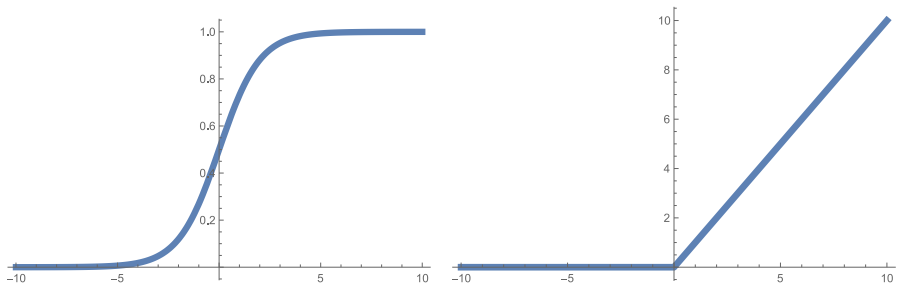


Figure 1.13: Plot of a sigmoid (left) and a ReLU (right) activation functions, typically used in artificial neural networks.

The multiplication and addition in each neuron represents a linear operation. Because of this, only connecting neurons to others would not provide any benefit to having a single neuron. To actually benefit from having multiple neurons at once, a non-linear operation is applied after the addition, thus removing the possibility of simplifying them. Common non-linearity functions are *sigmoid* and *ReLU* (Rectified Linear Unit), shown in Fig. 1.13.

To simplify creation and management of these networks, the neurons are most frequently set up in *layers*, each holding a number of neurons, and two connected layers have all the possible connections between their individual neurons (since missing connections can be represented with a factor of 0), as shown in Fig. 1.14. This allows for using matrix multiplication algorithms to calculate the output values of an entire network layer at the same time. We can represent the values of a layer and the factors of the connections in a *tensor* — an extension to matrices from linear algebra to higher number of dimensions.

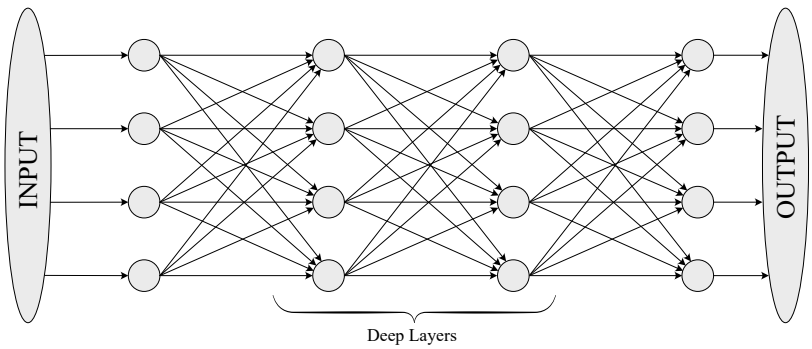


Figure 1.14: Multiple layers of neurons with full connectivity between every two adjacent layers.

The training process for neural networks is usually called *backpropagation*. That is because the algorithm it follows starts by evaluating the network with some input, comparing the output of the network with the expected value using an error function, and then *backpropagating* the derivative of the error (the value obtained from the error function) through the network, effectively using stochastic gradient descent optimisation of the parameters. This way, each neuron connection receives a value that represents how its factor should change to improve the result obtained.

Deep Neural Networks are networks that have more than two layers; the intermediate layers are called *deep layers*.

1.4.2.1 Convolutional Neural Networks

When dealing with spatial data such as images, for which we would like to get similar results even if the information is moved by some amount in the spatial dimensions, Convolutional Neural Networks (CNNs) are helpful.

CNNs, instead of sending each value of the input through a different connection, apply a smaller set of connections to contiguous subsets of the input, as represented in Fig. 1.15. This small set of connections is called a *filter*. Usually we will have several independent filters applied at the same time, each producing a value for their application. The set of output values for each of these filters for the same inputs are usually considered as the channels of the output tensor, also represented in Fig. 1.15.

A typical convolution can produce tensors that are equal or smaller in size than the input (for dimensions other than the channels), depending on the size of the filters, the amount of values skipped between every application of

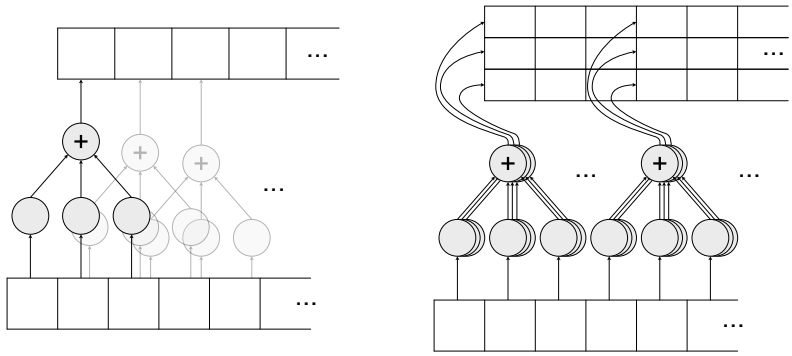


Figure 1.15: representation of a 1-dimensional convolutional neural network, where a network with fewer inputs than the total amount of data points is *convolved* with the data, i.e. the input is advanced and the network is applied again, storing all the results in a new list. Each different filter produces a different channel in the final output, as represented on the diagram on the right.

the filter (stride), and other parameters. *Transpose Convolutions*, also called *Backward Convolutions* or *Deconvolutions*, can produce tensors bigger than the input. The backpropagation step of a convolution is usually implemented this way, hence the name.

A typical architecture where CNNs are used consists of an initial set of convolutions, which take the size of the input down by several orders of magnitude (while increasing the number of channels), followed by a set of convolutions and upsampling[2] layers, which take the tensors back to the original size.

CNNs are used, for example, for semantic segmentation [22] — determining if a pixel belongs to a specific kind of object, by gathering structural information around each pixel and using it to determine properties about the context of each pixel. It is also used for more straightforward image filtering [9], colorisation of black-and-white images [13], and many other applications.

[2]Operations that increase the size of the input. For example, duplicating the size by copying each pixel next to it, or linearly interpolating for the points between pixels.

Chapter 2

Summary of Included Papers

2.1 Spherical Gaussian Light-Field Textures for Fast Precomputed Global Illumination

In this paper we present a method to precompute information that can be used to reproduce global illumination in real-time.

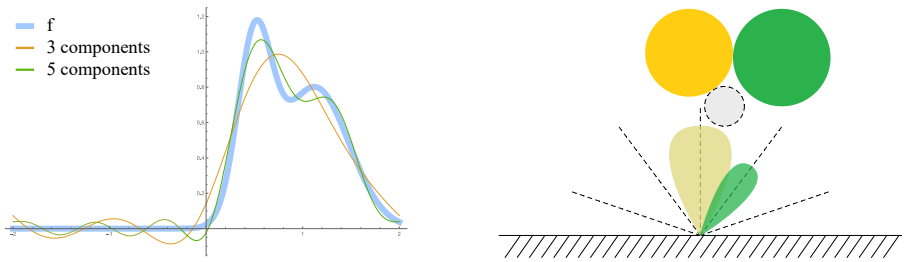
Problem

As mentioned in Section 1.1, functions that have been used to encode the BRDF and the incoming light to points on a scene include Spherical Harmonics and Spherical Gaussians. Both these methods have cheap pre-computation steps as well as being fast to evaluate.

The usual way Spherical Harmonics are used to represent incoming radiance at a point is by first deciding a number of harmonics to use, and then their values can be calculated for each of the three RGB color channels. Unfortunately, spherical harmonics can produce a lot of visible banding, as depicted in Fig. 2.1.

On the other hand, when Spherical Gaussians are used to represent incoming radiance at points on the scene, they are usually implemented by first deciding a number of gaussians to use and pre-determining a direction for each of them. Then their values can be obtained by applying an optimisation algorithm, such as gradient descent or expectation maximisation, on the sharpness and amplitude of the gaussian, with the amplitude taken as a set of 3 values for RGB. This method also has issues with high-frequency details: since there is only a small chance that the predetermined direction of the functions used matches properly with those details, most places with sharp changes in color will end up blended together, as depicted in Fig. 2.1.

Naively attempting to optimise the direction of each gaussian would result in sets of gaussians that cannot be interpolated, as the direction for each would need to be considered together with that of its neighbors. We show this in Fig. 2.2.



(a) (b)

Figure 2.1: (a) shows the problem of spherical harmonics, where high frequency details will create high frequency noise in areas that should be flat otherwise. (b) shows that spherical gaussians cannot usually deal with high frequency details because they will likely end up in the wrong place and become blurred away.

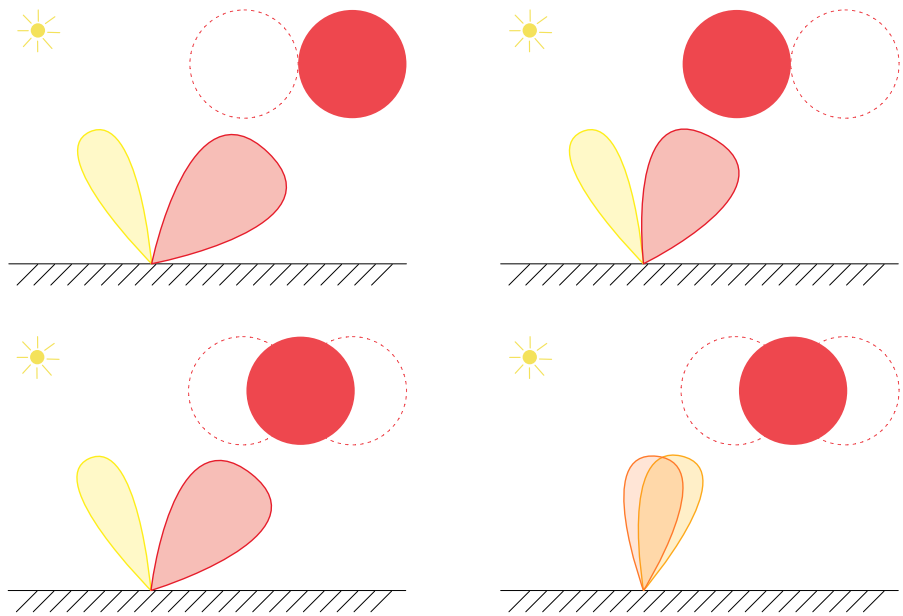


Figure 2.2: When attempting to fit spherical gaussians to a set of neighboring lightfields, the order needs to be considered, or the interpolation will produce undesired results. Top images show gaussians for different points in the scene, bottom-left is the expected interpolation of the gaussians for an intermediate point, while bottom-right is the interpolation at that point if the gaussians in the surrounding points are not kept in the same order.

Contribution

The main contribution of this paper is a method that relies on the ability of neural networks to produce similar outputs for similar inputs to generate the optimised parameters for spherical gaussians that can be interpolated between sample points.

With this method we are able to produce a lightfield map where each texel stores a set of several spherical gaussians with free directions, allowing it to represent some high-frequency details that would not be representable with other methods, and therefore much better visual results than spherical harmonics or fixed-direction spherical gaussians.

We also adapt the work from Heitz et al. [11] to get much higher detail reflections from a distant environment map by encoding the BRDF-dependent visibility function into a separate set of spherical gaussians.

Methodology

We implement a convolutional neural network that will use a set of lightfield images taken from the scene for which we want to compress the lightfields and produces a set of parameters for a predetermined number of spherical gaussians that, when added together, produce images as alike as possible to the input lightfield images.

The input to the network is obtained by building a non-overlapping set of texture coordinates for the whole scene and subdividing the texture space into a grid. Each point in the grid that corresponds to geometry will be the center of a ligthfield texture. Fig. 2.3 exemplifies this.

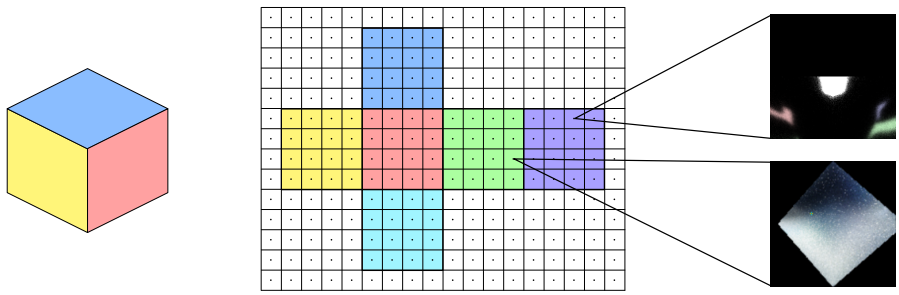


Figure 2.3: Texture coordinates used to create the lightfield map, and the lightfield image that is obtained for each of its cells.

Each lightfield texture is then evaluated through the network and the resulting gaussian parameters are evaluated, added together, and compared with the input to do gradient descent on the magnitude of the difference.

Upon convergence of the network, the parameters of the gaussians for each lightfield are gathered and stored, to be read by the real-time application that will make use of the generated gaussians.

During real-time evaluation, when rendering each point in the scene, the spherical gaussian parameters are sampled with linear interpolation using the texture coordinates used to create them initially. The BRDF of the surface at the point for the viewing direction is approximated with an anisotropic spherical gaussian[1] [20]. Then, each lightfield gaussian is analytically convolved with the approximation of the BRDF to obtain the value that the gaussian contributes to the illumination of the point.

[1] See Appendix B of Paper I for details

The code used to produce the results presented in this paper can be found in `https://gitlab.com/ror3d/spherical-gaussian-lightfields`.

2.2 Real-Time Hair Filtering with Convolutional Neural Networks

Problem

As mentioned in Section 1.3, if we want to render hair so that it looks as realistic as possible, it becomes necessary to represent each hair strand individually.

Since hairs are extremely thin, care needs to be taken to avoid aliasing. This means that rendering each hair strand at its real width is not usually an option, as that would cause severe aliasing, as depicted in Fig. 2.4(b). The naïve method to solve this would be to sample at more points for each pixel, for example using either super-sampling or multi-sample anti-aliasing, but that would require tens or hundreds of samples per pixel to ensure that the hair is properly drawn.

A simple approach is to ignore the problem and render each hair one-pixel thick, while also ignoring the transparency of the hair. That, obviously, makes the hair look too thick and dense, as shown in Fig. 2.4(c).

(a) (b) (c)

Figure 2.4: (a) shows hair rendered at high resolution; (b) is the same mesh of hair at real thickness with 1 sample per pixel; (c) shows the hair rendered with 1-pixel thick strands.

An approximation that avoids the aliasing relies on rendering the hairs at one-pixel thickness, while applying extra transparency proportional to the ratio of the hair's thickness over the width of a pixel, as detailed in Fig. 2.5.

As mentioned in Section 1.2, several techniques have been implemented for order-independent rendering of transparent geometry [1, 24, 23, 16], but in general those do not work well for meshes such as hair, due to the high depth complexity.

Stochastic methods for order independent transparency, such as stochastic transparency [7] or hashed alpha [25], can produce unbiased results for the rendering of the hair, but the results are also quite noisy, since not many samples can be taken in the short period available to render each frame.

Contribution

This paper proposes a method of filtering an image of the hair rendered using stochastic transparency to remove the stochastic spatial noise and produce

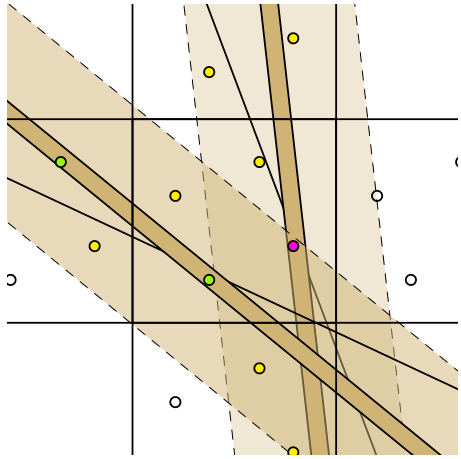


Figure 2.5: Hairs rendered at one-pixel thickness with transparency compensating for real thickness. The pink sample point shows why this approximation is slightly biased — that point will have information from both hairs, when it should only include information from one.

a filtered version that resembles the hair as it would look rendered through downsampling from a much higher resolution.

The filter proposed is implemented using a U-Net [22] neural network architecture, a set of convolutional neural networks followed by a set of transpose convolutional neural networks, that is small enough to be evaluated every frame in a real-time application.

Methodology

First, the color, the illumination, and the transparency of the hair, together with the screen-space depth and tangents, are rendered at 4 samples per pixel using multisample buffers, effectively producing an image with 24 channels. This image is the input to the neural network.

Several convolutional layers are applied on the image, each reducing its size by a factor of 2. Then, the same number of deconvolution layers are applied, which also include information passed through from the original convolutions, to bring that image back to its original size, producing an output with 3 channels representing the color, illumination and transparency of the hair.

These 3 channels output by the network are then composited together to produce the final image.

To train the network we render the hair images at high resolution and reduce them to the same resolution of the images used as the input of the network, so the network will attempt to replicate high quality images.

The training is implemented using the pytorch library, and the real-time application uses CuDNN for evaluation of the networks.

Chapter 3

Discussion and Future Work

In Paper I we show very appealing results, with very high performance, for a static scene; the scene needs to be static because the precomputations, as done currently, take several hours to complete, so an obvious path for improvement would be finding a way to speed-up the training process. One thing that was attempted was to re-use a trained network for previously unseen inputs, or to try to fine-tune an already trained network with new data. Our initial attempts took a similar time to re-train as that of training from scratch, but further research in this direction might prove more fruitful.

The method described works with isotropic spherical gaussians, but it could potentially be made to work with different functions, if the equations to convolve with the BRDF were derived such that they could be used in real-time. This could provide images with much sharper details than the ones created with the spherical gaussians.

On the other hand, as discussed in the paper, compressing the information in some way, such as taking advantage of existing image compression methods, or possibly some hierarchical structure, could allow for using our method with much higher resolution or with the sampling points distributed in space instead of only on the surfaces, which would allow for applying the precomputed global illumination also on dynamic objects (albeit those would not be taken into account for the global illumination of the rest of the scene).

There does not appear to be a straightforward way to have the method work for totally dynamic scenes, since training is too slow to be done in real time, but it would be extremely interesting to develop a method that could take advantage of this work to that effect.

Paper II demonstrates very clean results for very noisy input working in real-time, and very close to the ground-truth, which makes it highly appealing to use. As mentioned in the paper, several restrictions are set on the method that could be improved on, such as the need for the hair to be rendered in a uniform color — multi-colored hair other than shadows or highlights would require the network to handle full colors as input and output, instead of only operating on the brightness value. Training for that to work properly requires a lot more input data for the network to learn different color combinations, but

it might still be feasible and usable in real-time.

Another drawback of the method is the time taken by the initial rendering step for the input of the network. This is mostly caused by the large amount of geometry used. Finding an orthogonal way to the filtering step that could render the stochastic input in a fraction of the time would make the algorithm presented even more appealing.

Bibliography

[1] 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, (30th April 2007), 97–104. ISBN: 978-1-59593-628-8. DOI: 10.1145/1230100.1230117. Retrieved 29/12/2021 from https://doi.org/10.1145/1230100.1230117 (cited on pages 8, 20).

[2] Loren Carpenter. 1984. The A -buffer, an antialiased hidden surface method. *ACM SIGGRAPH Computer Graphics*, 18, 3, (1st January 1984), 103–108. ISSN: 0097-8930. DOI: 10.1145/964965.808585. Retrieved 29/12/2021 from https://doi.org/10.1145/964965.808585 (cited on page 8).

[3] 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, (20th July 2017), 98:1–98:12. ISSN: 0730-0301. DOI: 10.1145/3072959.3073601. Retrieved 29/12/2021 from https://doi.org/10.1145/3072959.3073601 (cited on page 12).

[4] Ken Dahm and Alexander Keller. 2017. Learning light transport the reinforced way. In *ACM SIGGRAPH 2017 Talks* (SIGGRAPH '17). Association for Computing Machinery, New York, NY, USA, (30th July 2017), 1–2. ISBN: 978-1-4503-5008-2. DOI: 10.1145/3084363.3085032. Retrieved 29/12/2021 from https://doi.org/10.1145/3084363.3085032 (cited on page 12).

[5] A. P. Dempster, N. M. Laird and D. B. Rubin. 1977. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39, 1, 1–22. ISSN: 2517-6161. DOI: 10.1111/j.2517-6161.1977.tb01600.x. Retrieved 29/12/2021 from https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1977.tb01600.x (cited on page 12).

[6] Devcore. 2012. 8x8 Discrete Cosine Transform. (2012). Retrieved 02/01/2022 from https://commons.wikimedia.org/wiki/File:DCT-8x8.png (cited on page 7).

[7] Eric Enderton, Erik Sintorn, Peter Shirley and David Luebke. 2011. Stochastic Transparency. *IEEE Transactions on Visualization and Computer Graphics*, 17, 8, (August 2011), 1036–1047. ISSN: 1941-0506. DOI: 10.1109/TVCG.2010.123 (cited on page 20).

[8] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks, John G. Eyles and John Poulton. 1985. Fast spheres, shadows, textures, transparencies, and image enhancements in pixel-planes. *ACM SIGGRAPH Computer Graphics*, 19, 3, (1st July 1985), 111–120. ISSN: 0097-8930. DOI: 10.1145/325165.325205. Retrieved 29/12/2021 from https://doi.org/10.1145/325165.325205 (cited on page 8).

[9] Michaël Gharbi, Jiawen Chen, Jonathan T. Barron, Samuel W. Hasinoff and Frédo Durand. 2017. Deep bilateral learning for real-time image enhancement. *ACM Transactions on Graphics*, 36, 4, (20th July 2017), 118:1–118:12. ISSN: 0730-0301. DOI: 10.1145/3072959.3073592. Retrieved 29/12/2021 from https://doi.org/10.1145/3072959.3073592 (cited on page 15).

[10] Alexandra Haynes. 2021. Female Bob Low-poly Hairstyle. (10th February 2021). Retrieved 31/12/2021 from https://www.cgtrader.com/free-3d-models/character/other/realtime-female-bob-hairstyle-game-ready (cited on page 11).

[11] Eric Heitz, Stephen Hill and Morgan McGuire. 2018. Combining analytic direct illumination and stochastic shadows. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (I3D '18). Association for Computing Machinery, New York, NY, USA, (15th May 2018), 1–11. ISBN: 978-1-4503-5705-0. DOI: 10.1145/3190834.3190852. Retrieved 29/12/2021 from https://doi.org/10.1145/3190834.3190852 (cited on page 18).

[12] Daniel Holden, Taku Komura and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Transactions on Graphics*, 36, 4, (20th July 2017), 42:1–42:13. ISSN: 0730-0301. DOI: 10.1145/3072959.3073663. Retrieved 29/12/2021 from https://doi.org/10.1145/3072959.3073663 (cited on page 12).

[13] Satoshi Iizuka, Edgar Simo-Serra and Hiroshi Ishikawa. 2016. Let there be color! joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Transactions on Graphics*, 35, 4, (11th July 2016), 110:1–110:11. ISSN: 0730-0301. DOI: 10.1145/2897824.2925974. Retrieved 29/12/2021 from https://doi.org/10.1145/2897824.2925974 (cited on page 15).

[14] James T. Kajiya. 1986. The rendering equation. *ACM SIGGRAPH Computer Graphics*, 20, 4, (31st August 1986), 143–150. ISSN: 0097-8930. DOI: 10.1145/15886.15902. Retrieved 29/12/2021 from https://doi.org/10.1145/15886.15902 (cited on page 4).

[15] Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley and Pat Hanrahan. 2003. Light scattering from human hair fibers. In *ACM SIGGRAPH 2003 Papers* (SIGGRAPH '03). Association for Computing Machinery, New York, NY, USA, (1st July 2003), 780–791. ISBN: 978-1-58113-709-5. DOI: 10.1145/1201775.882345. Retrieved 29/12/2021 from https://doi.org/10.1145/1201775.882345 (cited on pages 9, 10).

[16] Marilena Maule, João L. D. Comba, Rafael P. Torchelsen and Rui Bastos. 2011. A survey of raster-based transparency techniques. *Computers & Graphics*, 35, 6, (1st December 2011), 1023–1034. ISSN: 0097-8493. DOI: 10.1016/j.cag.2011.07.006. Retrieved 29/12/2021 from https://www.sciencedirect.com/science/article/pii/S009784931100135X (cited on page 20).

[17] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross and Jan Novák. 2019. Neural Importance Sampling. (3rd September 2019). eprint: 1808.03856 (cs, stat). Retrieved 29/12/2021 from http://arxiv.org/abs/1808.03856 (cited on page 12).

[18] Thomas Müller, Fabrice Rousselle, Jan Novák and Alexander Keller. 2021. Real-time neural radiance caching for path tracing. *ACM Transactions on Graphics*, 40, 4, (19th July 2021), 36:1–36:16. ISSN: 0730-0301. DOI: 10.1145/3450626.3459812. Retrieved 29/12/2021 from https://doi.org/10.1145/3450626.3459812 (cited on page 12).

[19] Rashmi Nanjundaswamy. 2016. Scanning electron microscope image of a human hair. (2016). Retrieved 02/01/2022 from https://www.nisenet.org/catalog/scientific-image-sem-human-hair (cited on page 10).

[20] Matt Pettineo. [n. d.] SG Series Part 4: Specular Lighting From an SG Light Source. The Danger Zone. Retrieved 02/01/2022 from https://therealmjp.github.io/posts/sg-series-part-4-specular-lighting-from-an-sg-light-source/ (cited on page 18).

[21] Thomas Porter and Tom Duff. 1984. Compositing digital images. *ACM SIGGRAPH Computer Graphics*, 18, 3, (1st January 1984), 253–259. ISSN: 0097-8930. DOI: 10.1145/964965.808606. Retrieved 29/12/2021 from https://doi.org/10.1145/964965.808606 (cited on page 8).

[22] Olaf Ronneberger, Philipp Fischer and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (Lecture Notes in Computer Science). Nassir Navab, Joachim Hornegger, William M. Wells and Alejandro F. Frangi, editors. Springer International Publishing, Cham, 234–241. ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28 (cited on pages 15, 21).

[23] 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, (5th August 2011), 119–126. ISBN: 978-1-4503-0896-0. DOI: 10.1145/2018323.2018342. Retrieved 29/12/2021 from https://doi.org/10.1145/2018323.2018342 (cited on page 20).

[24] 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* (I3D '08). Association for Computing Machinery, New York, NY, USA, (15th February 2008), 157–162. ISBN: 978-1-59593-983-8. DOI: 10.1145/1342250.1342275. Retrieved 29/12/2021 from https://doi.org/10.1145/1342250.1342275 (cited on page 20).

[25] Chris Wyman and Morgan McGuire. 2017. Hashed alpha testing. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (I3D '17). Association for Computing Machinery, New York, NY, USA, (25th February 2017), 1–9. ISBN: 978-1-4503-4886-7. DOI: 10.1145/3023368.3023370. Retrieved 29/12/2021 from https://doi.org/10.1145/3023368.3023370 (cited on page 20).

[26] Shilin Zhu, Zexiang Xu, Tiancheng Sun, Alexandr Kuznetsov, Mark Meyer, Henrik Wann Jensen, Hao Su and Ravi Ramamoorthi. 2020. Photon-Driven Neural Path Guiding. (5th October 2020). eprint: 2010.01775 (cs). Retrieved 29/12/2021 from http://arxiv.org/abs/2010.01775 (cited on page 12).

Part II

Appended Papers

Paper I

Spherical Gaussian Light-Field Textures for Fast Precomputed Global Illumination

Roc R. Currius, Dan Dolonius, Ulf Assarsson, Erik Sintorn

Computer Graphics Forum Vol. 39 Issue 2, May 2020
Pages 133-146

10.1111/cgf.13918

EUROGRAPHICS 2020 / U. Assarsson and D. Panozzo
(Guest Editors)

Volume 39 (2020), Number 2

Spherical Gaussian Light-field Textures for Fast Precomputed Global Illumination

R. R. Currius[1], D. Dolonius[1], U. Assarsson[1], and E. Sintorn[1]

[1]Chalmers University of Technology, Sweden

(a) **(b)**

Figure 1: *Two scenes rendered with our method. The local light field for any fragment is available as a precomputed set of 16 Spherical Gaussians in a* light-field texture *(*512×512*, 56MB). A similar texture contains the attenuation factor for a preconvolved environment map. The combined result is images with full global illumination for glossy surfaces rendered in just over a millisecond at 1080p resolution.*

Abstract

We describe a method to use Spherical Gaussians with free directions and arbitrary sharpness and amplitude to approximate the precomputed local light field for any point on a surface in a scene. This allows for a high-quality reconstruction of these light fields in a manner that can be used to render the surfaces with precomputed global illumination in real-time with very low cost both in memory and performance. We also extend this concept to represent the illumination-weighted environment visibility*, allowing for high-quality reflections of the distant environment with both surface-material properties and visibility taken into account. We treat obtaining the Spherical Gaussians as an optimization problem for which we train a Convolutional Neural Network to produce appropriate values for each of the Spherical Gaussians' parameters. We define this CNN in such a way that the produced parameters can be interpolated between adjacent local light fields while keeping the illumination in the intermediate points coherent.*

CCS Concepts

• ***Computing methodologies*** → ***Rendering; Ray tracing;***

1. Introduction

To achieve realistic computer generated images, the *indirect illumination* of each visible surface point must be accounted for. The current de-facto method for rendering such images is *path tracing*, where the Light Transport Equation [Kaj86] is numerically estimated. In real-time applications, even on high-end GPUs with dedicated ray-tracing hardware, only a few samples per pixel and frame are achievable. Recently, several de-noising techniques have been developed that reuse samples from adjacent pixels and frames [CKS*17; MMBJ17]. These techniques show great promise and allow for rendering scenes with fully dynamic lighting and materials. However, they are still much too expensive on mid or low-end hardware.

Therefore, in applications where lighting, geometry, and materials can be considered static, it is often preferable to rely on precomputing the indirect illumination in the scene and using ray tracing

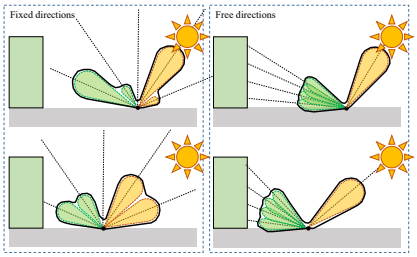


Figure 2: *Left: Using Spherical Gaussians with fixed directions [Pet16] (or Spherical Harmonics [RH01]) the incoming light is projected onto the directions being considered. High frequency changes in the illumination cannot be captured, and there will be visible aliasing as we interpolate between two receiving points. Right: With free directions a much higher quality can be obtained and interpolation can be free from aliasing.*

only for specific effects. When illumination can be pre-computed, the remaining questions are how to store a sufficiently dense sampling within a fixed memory budget, how to reconstruct the local light field, and how to convolve it with the *Bidirectional Reflectance Distribution Function* (BRDF) to get the reflected light.

Common choices of *Spherical Radial Basis Functions* (SRBFs) to store the light field are *Spherical Harmonics* (SHs) [RH01] and *Spherical Gaussians* (SGs) [TS06]. With SHs, a few coefficients are stored that describe a set of orthogonal functions on the sphere that can be combined to approximate the light field. With SGs a sum of gaussian lobes are used instead. Wang et al. [WRG*09] describe how the SVBRDF (Spatially Varying Bidirectional Reflectance Distribution Function) can be described in this form for each vertex, allowing for environment lighting in real time. SGs were used to encode light-field textures in the videogame *The Order 1886*, as described by Pettineo and Neubelt [Pet16]. The authors show that, with 12 SGs with *fixed* direction and sharpness (i.e. 36 floats), they can better represent the original light field than a 3-band SH representation (24 floats). Both methods benefit from expressing the BRDF in the same representation as the light fields, allowing for fast and efficient convolution with the incoming illumination.

Figure 2 illustrates a problem with using either SHs or SGs with fixed directions for approximating the incident illumination. Firstly, since the direction of the basis functions are fixed, the lobes can not be moved to where they are most useful. A much better reconstruction of the local light field can be obtained if lobes are concentrated where they are most needed. Secondly, as a source of illumination moves between two of these directions, the reconstructed illumination can only respond by modifying the amplitude, causing clearly visible aliasing in the reflections.

However, allowing for non-fixed directions is far from trivial. Optimizing only the amplitude can be solved with a linear least square solver. With arbitrary directions and sharpness the problem is much more complex. Additionally, it is imperative that the parameters of the SGs are interpolatable between, e.g., nearby light probes or texels in a light map.

The main contribution of this paper is an alternative approach to solving this optimization problem. Instead of optimizing the SG parameters directly, we train a *Convolutional Neural Network* (CNN) to generate them. Figure 3 shows an overview of our system. We start with a scene with a unique UV parametrization and a precomputed irradiance texture. The goal is to create another texture, the *light-field texture*, where every texel contains the SG parameters (axis, sharpness and amplitude) required to recreate the local light field. We first pathtrace the local light field from every texel's position and store it as a 2D *light-field image* to disk. Next, we train the CNN using these images as input to generate a set of parameters for a number of SGs. The sum of SGs are evaluated to predict the light-field image, and the error is backpropagated through the network. When the training has converged, the output SG parameters for each texel are saved as the light-field texture. A benefit of this approach is that, similarly to how an autoencoder works, the network will produce similar SG parameters for adjacent input light field images, and so a lookup in the light-field texture will produce plausible results when interpolated.

Once the light-field texture is created, it can be used to render the scene with indirect illumination in real time. A fragment shader fetches an interpolated set of gaussian parameters and very efficiently convolves this incident illumination with the BRDF to estimate the reflected radiance towards the camera.

As a second contribution we suggest an algorithm for allowing high-resolution glossy reflections from environment maps while taking visibility into account. A common approximation in real-time applications is to preconvolve the environment map with the *Normal Distribution Function* (NDF) and replace the expensive convolution with a single 3D texture lookup at runtime. The remaining terms of the light transport equation are moved outside of the integral and evaluated only for the perfect specular direction. The error of this estimation will be worse the rougher the material is, but in practice it works well for unoccluded reflection. As illustrated in Figure 4, using a preconvolved environment map is problematic when visibility is to be taken into account. Even if some representation of the local visibility is available, the convolution with the environment map must happen at run-time for correct results.

Inspired by the recent work by Heitz et al. [HHM18], we instead suggest rendering, for each texel and all directions, the preconvolved environment both with and without visibility. By taking their ratio we get a spherical function (represented by a 2D image) which we call the *illumination-weighted environment visibility*. These images are then compressed to spherical gaussians, as described above, and can be easily evaluated for any direction in the shader. Multiplying this result by the pre-convolved environment map gives us a high-quality estimation of the actual reflected light.

Together, these novel contributions allow us to render static, complex, scenes with glossy reflections from any viewpoint using high-resolution precomputed illumination and environment visibility stored as a set of a few spherical gaussians per texel. As shown in Figure 1, with 16 SGs per texel (56MB for a 512x512 light-field

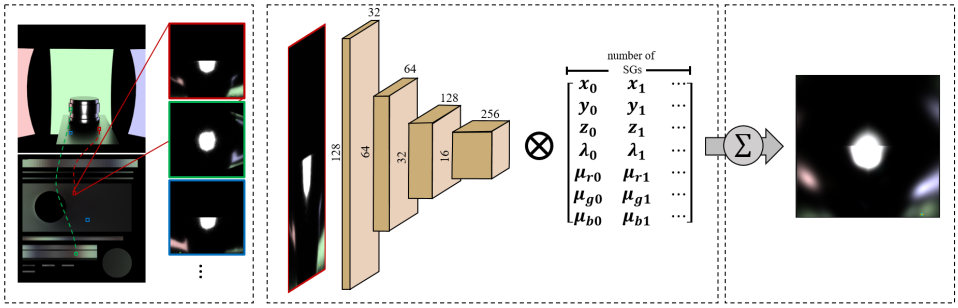


Figure 3: *Training a network to estimate spherical gaussian parameters. a) Given a scene and a UV unwrapping, the local light field (environment map) is pathtraced for every texel in the lightmap and stored as an image. b) These images are then passed through a Convolutional Neural Network where each layer consists of a convolution, max pooling and a ReLU activation. The output of final layer is passed through a fully connected layer to produce the parameters of each SG. c) Finally, the predicted local light field is calculated as the sum of these gaussians, and the error is backpropagated through the network. When the network is fully trained, the local light fields of each texel are run through the network again, and the predicted parameters are stored in the corresponding texel of a light-field texture.*

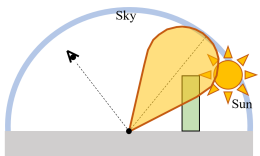


Figure 4: *Evaluating visibility only in the center of the BRDF lobe, to attenuate the preconvolved incoming radiance in that direction, can lead to significant light-leaking, as in this example where the surface should not reflect any sunlight.*

texture with 16-bit floats), we achieve smooth results, comparable to a pathtraced reference, in just over a millisecond on an RTX 2080 graphics card.

2. Previous Work

Image-Based Lighting (IBL). In 1976, Blinn and Newell [BN76] presented their work on *environment maps*, i.e. images representing the incoming radiance for a single point from all directions. For distant illumination, this technique is still in use today, usually extended by preconvolving the incoming radiance with the BRDF to allow for plausible glossy reflections [MS16]. It is common to render several environment maps at several points in the scene, which can then be blended together in an attempt to recreate the light field at an arbitrary point [SZ12]. Unless these *light probes* are extremely densely placed (requiring extensive amounts of memory), such methods will suffer from visibility errors. We refer the reader to a tutorial and survey of image-based lighting by Debevec [Deb06] for more details on image-based lighting.

Irradiance and Precomputed Radiance Transfer. For diffuse or very rough materials, light probes can be compactly described using Spherical Harmonics rather than a full environment map [RH01]. In *Precomputed Radiance Transfer* [SKS02], the *transfer function*, i.e., how the incoming radiance is transferred to a specific direction is precomputed. This allows for relighting of an object without recomputing the radiance transfer. This method has been extended to allow dynamic objects [SLS05] and to represent soft shadows [RWS*06].

Spherical Harmonics require many coefficients not to exhibit ringing artifacts when used to represent high-frequency functions, so they are limited to materials with high roughness. Tsai and Shih [TS06] represent both the transfer functions and the light sources with Spherical Gaussians, which allows for high-frequency lighting environments, but this method cannot easily handle spatially varying BRDFs and detailed reflections for rough materials are difficult to reconstruct. Green et al. [GKMD06] also compress the transfer function using Gaussians. Wang et al. [WRG*09] instead represent the BRDF as SGs and represent environment visibility as a *spherical signed distance function*. For environment lights they sample a preconvolved environment map, which will cause artifacts for rough lobes in certain lighting conditions (see Figure 4). To allow for dynamic scenes, Iwasaki et al. [IFDN12] approximate the geometry using spheres to create a visibility estimation which they can efficiently convolve with the lighting and BRDF.

Xu et al. introduced *Anisotropic Spherical Gaussians*, which are shown to produce better reconstructions of some functions with much fewer lobes [XSD*13]. While we do use anisotropic gaussians to represent the BRDF lobe in our real-time evaluation (see Section 6), we use a sum of isotropic gaussians to represent the local light field to avoid the extra amount of memory required.

None of these methods attempt to capture the local light field,

and thus they are not applicable to interreflections and global illumination rendering. In contrast, we suggest both a method for distant environment lighting with improved quality for rough BRDF lobes, and a method in which the precomputed local light field is reconstructed for every texel, allowing for very fast indirect illumination from any surface in the scene, as long as the scene, lighting, and materials can be considered static.

Xu et al. [XCM*14] derive an expression for a SG representing the reflected radiance from one triangle and of a node in a hierarchical representation of the scene, allowing for diffuse and glossy one-bounce interreflections at interactive frame-rates. In the work by Meder and Bruderlin [MB18], a hierarchy of *Virtual Spherical Gaussian Lights* (VSGLs) is generated by mip-mapping a *Reflective Shadow Map*. When shading, a predetermined number of VSGLs are importance sampled from the hierarchy and convolved with the BRDF, expressed as a SG. This method greatly improves the quality of reflections compared to standard Virtual Point Light sampling.

In several of these works [TS06; WRG*09; XSD*13] a method is required to fit a set of Spherical Gaussians to an environment map, which is achieved in an iterative process by first separately solving for directions and sharpness using the L-BFGS-B algorithm [ZBLN97] and then projecting the amplitude using a least-squares solver. In the work by Vorba et al. [VKŠ*14], the sphere of incoming radiance is projected to a 2D plane and a standard *Gaussian Mixture Model* (GMM) is used, rather than Spherical Gaussians. In their work on Normal Map Filtering [HSRG07], Han et al. instead use the von Mises-Fisher distribution to represent the *Normal Distribution Function* in filtered mipmap levels. Similarly to Green et al. [GKMD06], they add a term to the likelihood function that enforces coherency in directions for neighboring lobes, to allow for interpolation. All three use the *Expectation-Maximization* (EM) algorithm to efficiently estimate the gaussian parameters.

Vorba et al. [VKŠ*14] use bi-variate Gaussians to represent incoming radiance but in an off-line rendering context. They maintain a spatial cache of directional gaussian distributions to approximate a PDF for the incoming radiance. The renderer then uses only the closest cached distribution to importance sample new directions, so no interpolation between distributions is required.

Real-Time Indirect Illumination. The body of work on real-time indirect illumination is vast and spans decades. We refer the reader to the excellent STAR report by Ritschel et al. [RDGK12] for a detailed survey, and will only cover the most relevant works here.

Much recent work relies on rendering a very noisy image using real-time path tracing and denoising the results, e.g. by factoring the LTE and using carefully chosen filters [MMBJ17], or training a recursive autoencoder [CKS*17]. These methods can work very well but are still quite costly even on high-end hardware.

Faster, and more approximate, methods include Voxel Cone Tracing [CNS*11] where a low-resolution voxel representation of the scene is updated and ray-traced every frame, Photon Splatting approaches [ML09; MSK*16], and Light Propagation Volumes [KD10]. Despite often being able to produce very good results, these algorithms are rarely used in the industry due to their relatively high cost. More often, a combination of sparse precomputed illumination and very approximate screen-space methods, e.g., screen-space reflections [MM14] and screen space ambient occlusion [Mit07], are used. The work by McGuire et al. [MMNL17] falls somewhere between; precomputed environment maps, including normal and distance information, are calculated for sparsely placed light probes, which are then ray marched for each pixel to estimate the color of the reflecting surface.

Neural network approaches. Ren et al. [RWG*13] divide the scene into small sub-spaces and store a *Radiance Regression Function* (a small NN) in each, which approximates the outgoing radiance given the viewing direction, surface position, and surface normal. [GvSS17] shows that an image consisting of separate entities can be disentangled into one image per K objects by learning a separate representation vector for each object and a function (a neural network) that allows them to associate each pixel with a specific object. Somewhat similarly, in our method, a CNN learns to map features found in the input light-field images to specific SG parameters.

In the work of Hermosilla et al. [HMRR18], a sparsely sampled point cloud of the scene is processed by a Convolutional Neural Network to obtain abstract features. A second network is trained to process these features, along with the point cloud, to obtain, e.g., AO values for each point. A high-quality shaded image can then be produced, at interactive framerates, by feeding the network the visible points of a 2D image (the GBuffer). The method produces plausible values for points it has not previously seen. View-dependent global illumination is not handled by this method.

Our method is somewhat related to the problem of *inverse graphics* techniques, where the goal is to find scene parameters given observed images. Maximov et al. [MRF18] train a network that describes a *Deep Appearance Map* (DAM) which, given a normal and view direction, outputs the correct radiance for a specific material. They then train a separate network that, given an input photograph, can produce a new DAM very efficiently. Several recent papers have made use of a *differentiable renderer* [LADL18; LHJ19] which can compute derivatives of arbitrary scene parameters from the rendered image to find optimal values. In the work by Chen et al. [CGL*19], a target image is fed through a CNN to predict, e.g., vertex positions which are in turn processed by the differentiable renderer to produce an image. Through back-propagation, the CNN can be updated to improve the estimated vertex positions. Similarly, Wang et al. [WRM17] train a network to reproduce the outgoing radiance given a material, light and view direction. Since it is differentiable, they can then optimize these parameters for a target photograph, allowing for, e.g., inserting new objects in the image with plausible materials and lighting.

Interpolating between environment maps can arguably have similarities to constructing images for novel view points. There, *Deep Neural Network* (DNN) approaches have increasingly gained attraction [FNPS16; ZTF*18; KWR16; SWS*17; FBD*19]. However, these methods do not directly lend themselves for efficiently compressed light-field representations and, when applicable, real-time evaluation is much more expensive than our proposed method. In these methods, neural networks are used to predict the result, which is costly even with hardware acceleration. We only use a network to compute the SG parameters, which are then trivially

interpolated at run-time. DNNs have also been used for other related tasks, such as real-time light field reconstruction [CWZ*18; MKU13], approximate global illumination [TF17], and BRDF estimation from photos [AAL16], to mention a few.

3. Light-field Images

We store the gaussian parameters that approximate the local light field for each texel in a *light-field texture*, so all surfaces in the scene need a unique UV mapping. We follow a method similar to Rakhteenko's [Rak18] to obtain the positions and normals for each texel in the *light-field texture* while avoiding artifacts at seams and at points that lie inside other objects. Using these positions and normals, we compute a *light-field image*, a 2D image with the incident radiance projected from the sphere. For this we use a GPU-accelerated path tracer implemented using *Optix* [PBD*10]. We found an environment map size of 128×128 to be sufficient for the fidelity we can reconstruct and have used that size throughout the project. These light-field images are saved to disk in an uncompressed 16 bit float format, and sum up to tens of GBs for each of our test scenes.

4. Optimizing the SG parameters

A single spherical gaussian has the form: $G(\mathbf{v};\mathbf{u},\lambda,\mu) = \mu e^{\lambda(\mathbf{v}\cdot\mathbf{u}-1)}$, where $\mathbf{u}$ is the *axis* of the gaussian lobe, μ is the *amplitude*, and λ is the *sharpness*. For each texel, t, we want to approximate each channel, c, of each pixel, i, in each light-field image, $T_i(\mathbf{v})$, as a sum of N spherical gaussians:

$$P_{ic} = \sum_{j}^{N} G(\mathbf{v}_i;\mathbf{u}_j,\lambda_j,\mu_{jc}) = \sum_{j}^{N} \mu_{jc} e^{\lambda_j(\mathbf{v}_i\cdot\mathbf{u}_j-1)}, \tag{1}$$

where $\mathbf{v}_i$ is a direction corresponding to pixel i and depends on the spherical projection used. Therefore, the problem is to optimize all SG parameters such that the L_2 loss is minimized:

$$\sum_{t}^{\text{all texels}} \sum_{i}^{\text{all pixels}} \sum_{c}^{\text{all channels}} \left(\sum_{j}^{N} \mu_{tcj} e^{\lambda_j(\mathbf{v_i}\cdot\mathbf{u_{tj}}-1)} - T_{tc}(\mathbf{v_i}) \right)^2. \tag{2}$$

The number of parameters to optimize scales with the number of texels in the light-field texture. Even with a very small light-field texture of 128×128 texels and 16 SGs, the number of free parameters to optimize is 1.8M. Additionally, if the gaussians for all texels in the light-field texture are optimized independently, the converged parameters can differ very much between neighboring texels in the light-field texture, resulting in severe visual artifacts when interpolated.

To enforce locally coherent sets of SGs to solve this, previous work has suggested explicitly aligning the axis of adjacent SGs during the optimization task [GKMD06; HSRG07]. We show in Section 7 that, for our problem, this slows convergence and either blurs the resulting reflections, or leaves undesirable artifacts along lines where the optimizer could not resolve conflicting axes.

Instead, we propose a novel formulation of the problem. Rather than trying to optimize the SG parameters directly, we train a Convolutional Neural Network to produce good SG parameters given an input light-field image (see Figure 3). The motivation for our approach is twofold: First, by making the parameters a function of the input image, we encourage similar images to produce similar parameters. This is not guaranteed but, just as an autoencoder will cluster similar images in latent space, our network will tend to make the SG parameters' trajectories locally continuous in the light-field texture, allowing for interpolation. Secondly, rather than training all SG parameters in isolation, the CNN is shared among all texels. Therefore, updating the network to perform better for one texel is likely to improve the result for similar inputs. As will be shown in Section 7, this improves convergence dramatically.

An overview of our network is provided in Figure 3. The input is a 2D light-field image obtained as above, to allow for 2D convolution. We use the octahedron projection suggested by Meyer et al., due to its simplicity [MSS*10]. Note that this projection does not give equal projected area in all directions, which must be accounted for during training. Each layer of the CNN consists of a convolution, max pooling, and ReLU activation. The output of the last layer is the input of a fully connected layer with $N \times M$ outputs, where N is the number of SGs used and M is the number of parameters per SG. Next, the output image is generated by evaluating the sum of gaussians defined by these parameters (Eq 1). The predicted light-field image is compared to the input image and the loss is propagated backward through the network. In the following paragraphs we will go through each of these steps in detail.

Encoder Network. During one epoch of training, each light-field image is passed through the CNN to produce the predicted SG parameters. Each CNN layer performs a convolution of 3×3-pixel spatial support and a ReLU activation, followed by 2×2 max pooling to produce a new image of half the size. The first convolution layer produces an image with 32 channels and each of the three subsequent layers doubles the number of channels, resulting in a final image of 8×8 feature vectors with 256 channels. This is then used as input to a fully connected layer, without activation function, that outputs an $N \times M$ matrix of real numbers, each taken to represent one of the M parameters in one of the N SGs. The hyperparameters of the network were found empirically, and kept as low as possible without introducing a visual degradation of the result of our more challenging scenes.

Loss Function. Once the constrained SG parameters are available, we can run a final kernel to reconstruct the predicted light-field image. For each pixel and channel we evaluate the sum of the predicted spherical gaussians using Eq 1. Since the input and the predicted images represent radiance, which may be of high dynamic range, we minimize the L_2 log loss function: $(\log(T_{ic}+1) - \log(P_{ic}+1))^2$. This ensures that very high energy values (e.g., directly visible light-sources or specular highlights) are not given too much importance compared to darker areas. The details of back-propagating the gradient of the L_2 log loss with respect to each parameter are given in the Appendix A.

At this step we take into account that the projection used is not area-preserving: to avoid some pixels having more weight, their gradient contribution needs to be scaled relative to their unprojected solid angle.

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

Figure 5: *Two of our scenes with: a) Only diffuse component from irradiance map, b) Reflections from preconvolved environment map, c) Environment visibility using 16 SGs, d) Interreflections using 16 SGs. e) Is a path-traced reference.*

Constraints. During backpropagation we enforce constraints on the generated parameters by modifying their gradients depending on the type of parameter: the axis of the SG are constrained to be of unit length, and the amplitude and sharpness are constrained to be positive. We will note that, while the axis *could* be expressed using only two values, e.g. spherical coordinates, this would cause discontinuities both for the training network and for the real-time renderer when interpolating between directions. Also, while a negative amplitude is not necessarily erroneous, we found that enforcing strictly positive amplitudes consistently improved our results.

Optimizations. There are two non-obvious optimizations we have employed in the training. First, a pixel of the input image contains the average incoming radiance from a small set of directions, rather than a single direction. This must be accounted for when training, otherwise the network can overtrain and produce unwanted artifacts. However, evaluating Eq 1 for several directions for each pixel would be very costly so, instead, we randomly jitter the direction used for evaluation and take a single sample, which we found to be sufficient to avoid overtraining. Secondly, the path-traced input images only have valuable information in the hemisphere centered on the normal. Therefore, we do not let directions, $\mathbf{v}$, below the normal, $\mathbf{n}$, contribute to the gradient at all.

5. Illumination Weighted Environment Visibility

Ignoring visibility, mirror reflections from an environment map can be achieved with a single texture lookup. For glossy materials, modern applications usually employ some kind of Torrance Sparrow BRDF [TS92], making the light reflected to the camera from the environment be:

$$L_o(\omega_o) = \int_\Omega \frac{D(\omega_h)G(\omega_i,\omega_o)F(\omega_o)}{4\,|\omega_o\cdot\mathbf{n}|\,|\omega_i\cdot\mathbf{n}|} L_E(\omega_i)V_E(\omega_i)\,|\omega_i\cdot\mathbf{n}|\,d\omega_i, \tag{3}$$

where ω_o is the direction to the camera, ω_h is the half vector, Ω is all directions on the hemisphere, D is the Microfacet Distribution Function, G describes attenuation due to masking and shadowing, F is the fresnel term, and $\mathbf{n}$ is the surface normal. The visibility term, V, is often ignored. To achieve the look of a glossy material, without sampling the environment map excessively, a common trick is to preconvolve the $D(\omega_h)L_E(\omega_i)$ term for varying material roughnesses into a 3D texture, assuming a surface facing ω_o, and then approximate the reflected light as:

$$L_o(\omega_o) = \left(\int_\Omega D(\omega_h)L_E(\omega_i)d\omega_i\right) \frac{G(\omega_r,\omega_o)F(\omega_o)}{4\,|\omega_o\cdot\mathbf{n}|\,|\omega_r\cdot\mathbf{n}|}\,|\omega_r\cdot\mathbf{n}|, \tag{4}$$

where ω_r is ω_o reflected around the normal. This approximation is increasingly incorrect for rougher materials, and for grazing viewing directions, but often looks plausible and is commonly used in practice. Alternatively, a *dominant* reflection vector can be calculated by shifting the specular reflection vector towards the normal at grazing angles [Seb14]

While our light-field texture could contain illumination from the environment, that would be the same for every point in the scene so, rather than spending SGs on reconstructing the environment map at every texel, it is preferable to make use of the existing high-resolution pre-convolved environment map. The most obvious approach might be to use a texture of sums of SGs to approximate the visibility function, $V_E(\omega_i)$, but for rough materials this would be insufficient, as illumination is contributed from a larger cone of directions.

Instead, we extend an idea recently published by Heitz et

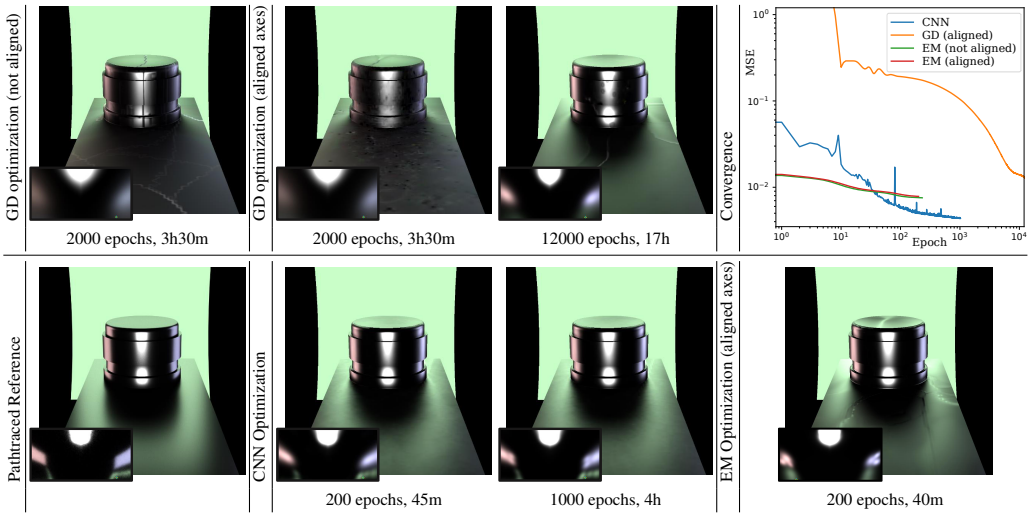


Table 1: *With no additional constraint, optimizing the SG parameters directly (EM or GD) causes disturbing artifacts between texels with very different solutions. Adding a regularization constraint (aligned axes) can alleviate this problem, but dampens the system, causing it to converge with a far from optimal result. By using a single CNN to produce all sets of SG parameters, local coherence between sets of SGs is enforced and the final MSE, which compares the input lightfield image with its obtained SG representation, is much smaller. The insets show the predicted light-field image for one texel (compare to the ground-truth light-field image in the inset of the pathtraced image).*

al. [HHM18], where correct soft shadows are computed by combining analytic area-light illumination and denoised, raytraced visibility. They suggest estimating the *illumination-weighted shadow*

$$W_S(\omega_o) = \frac{\int_\Omega R(\omega)L(\omega)V(\omega)d\omega}{\int_\Omega R(\omega)L(\omega)d\omega}, \tag{5}$$

where R is the cosine weighted BRDF, L is the incoming radiance from the light-source and V is the visibility. This term is stochastically estimated and then multiplied by the exact analytical estimation of the unshadowed illumination, $\int_\Omega R(\omega)L(\omega)d\omega$.

In our case, we consider the incoming radiance from an environment map, rather than an area light, and we have no means of evaluating that analytically. We can, however, preconvolve the unoccluded environment map:

$$U(\omega_o) = \int_\Omega D(\omega_h)L_E(\omega_i)d\omega_i, \tag{6}$$

and store the result in a 3D texture. We then precompute the *illumination-weighted environment visibility*:

$$W_E(\omega_o) = \frac{\int_\Omega D(\omega_h)L_E(\omega_i)V(\omega_i)d\omega_i}{\int_\Omega D(\omega_h)L_E(\omega_i)d\omega_i}, \tag{7}$$

for each light-field texel using a path tracer. This function we also represent using SGs, trained as described above.

Finally, in the real-time shader, we can multiply the preconvolved environment illumination with this estimation and, again, approximate the remainder of the LTE using the perfect specular reflection direction:

$$L_o(\omega_o) = U(\omega_o)W_E(\omega_o)\frac{G(\omega_r,\omega_o)F(\omega_o)}{4\,|\omega_o\cdot\mathbf{n}|\,|\omega_r\cdot\mathbf{n}|}\,|\omega_r\cdot\mathbf{n}| \tag{8}$$

As shown in Table 3, this way we can achieve plausible, high-resolution, glossy reflections from an environment map with visibility at the small cost of one environment lookup and evaluating a sum of spherical gaussians. This technique can be used on its own or in combination with the method described in the previous section. Note that while we compute the full, three-channel, illumination-weighted environment visibility, it would also in many cases be sufficient to use a monochrome result, reducing the amount of memory traffic.

6. Real-time Algorithm

To render the images shown in this paper we have used a deferred shading pipeline and applied the lighting from our light-field textures and environment visibility in the global lighting pass.

The light-field and environment visibility textures containing the gaussian parameters are read from disk and stored in *texture arrays*. Although our light-field texture could be used for diffuse reflections as well, we instead use the existing precomputed irradiance light map, and use the light-field texture only for glossy reflections. All illumination in the scenes comes from emissive surfaces or the environment.

The steps to apply the illumination from the light-field texture, for each pixel in the fragment shader, are:

1. Fetch position, normal, uv-coordinates, and material properties from the G-buffer.
2. Look up irradiance in the precomputed texture and calculate diffuse reflection.
3. Calculate the (anisotropic) SG that represents the $D(\omega_h)$ term from the material properties.
4. Fetch one SG at a time from the light-field texture (7 parameters, i.e., two texture lookups per SG) and convolve it with D.
5. Calculate the other BRDF terms F, G, and the dot products in the divisor for the perfect specular direction.
6. Multiply the obtained terms to obtain the glossily reflected radiance for this SG and accumulate it to the total glossy reflected radiance from the light-field texture.

To evaluate and convolve the spherical gaussians, we follow Pettineo [Pet16], the relevant definitions from which have been included in Appendix B, and we refer the reader to the paper by Wang et al. [WRG*09] for a full derivation.

The steps to render the environment map reflections using the illumination-weighted environment visibility method are:

1. Fetch position, normal, uv-coordinates, and material properties from the G-buffer (re-use the information already obtained when applying light-field texture).
2. Based on material roughness, look up $U(\omega_o)$ from the preconvolved environment map.
3. Fetch each SG (two texture lookups per SG) from the visibility factor texture and evaluate it in the reflected direction. Accumulate the evaluated value to obtain the visibility factor $W_E(\omega_o)$ for that pixel.
4. Calculate glossily reflected radiance from the environment according to Eq 8.

7. Results

The evaluation of our method was performed on an Intel core i7-8700 with an RTX 2080 graphics card. The training is implemented using nVidia's CUDA and cuDNN, and the real-time renderer is implemented in OpenGL. All scenes are lit only by our proposed method, either from emitting surfaces or environment maps. Direct lighting can be orthogonally added with any standard method.

Direct optimization of SG parameters. We primarily compare our suggested method of using a CNN to generate the SG parameters to a direct optimization of the parameters using Gradient Descent (GD). We have chosen a Gradient Descent solver, as that lets us train using the same initialization, loss function, and parameter gradients, allowing us to evaluate the benefit of using a CNN in isolation. We have additionally performed one comparison with directly optimizing the gaussian parameters using *Expectation Maximization* (EM) [HSRG07; HZE*19], by normalizing the amplitude of the SGs so the integral over the sphere adds up to 1, letting us treat the sum of them as a von Mises-Fisher mixture. We have observed that EM can converge much faster and to a better MSE result than GD (see Table 1), even though it's goal is not to optimize for MSE.

As expected, adjacent sets of SG parameters can not be smoothly interpolated if the parameters are optimized in isolation. To remedy this, we add the regularization constraint suggested by Green et al. [GKMD06] to GD optimization, and an alignment term [HSRG07] to EM. In both algorithms, the axes of the SGs are pushed towards the average of adjacent texels' axes. Introducing a constraint alleviates the problems slightly but, when converged, the reflections still show strong artifacts along lines where one SG changes too quickly. If we increase the weight of the constraint further, it dampens the system and the training converges at a much higher MSE. The result is very blurry reflections.

In contrast, when training a CNN to generate the parameters, coherence between nearby sets of gaussians is maintained indirectly, still allowing parameters to change quickly when doing so does not affect the MSE. This leads to a much better MSE for the converged result and the images obtained when using the SGs for reflection are much closer to the pathtraced reference.

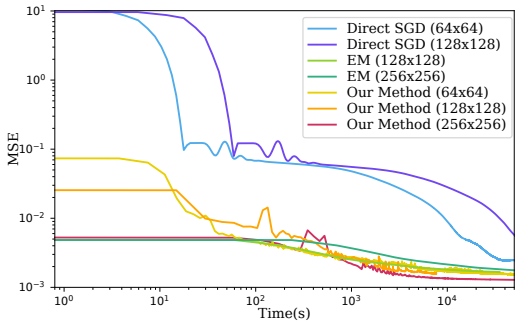


Figure 6: *Convergence for a simple scene with light-field textures of varying size.*

Using a CNN to produce the SG parameters also scales very well with the resolution of the light-field texture, as illustrated in Figure 6. Here, we can see that, while each epoch of training will take time proportional to the number of input light-field images, the time to convergence is essentially unaffected. Using our method, training is converged after approximately three hours, regardless of the light-field texture resolution. With a direct optimization of parameters (here without enforcing coherence between sets of SGs), the time to convergence is proportional to the number of input images. The average MSE of the final predicted light-field image is also much better with our method and, interestingly, improves with increased resolution.

Quality. Table 2 shows a comparison of our method for rendering images with precomputed light fields, using varying numbers of SGs per texel, and a path-traced reference. The scene is a simple test scene containing objects with a material of increasing roughness (0.2, 0.3, 0.4, and 0.5, GGX BRDF [WMLT07]) from left to right. The scene is illuminated by a number of emitting arcs that can be seen in the background. In the right column we see the light field as it was reconstructed for one of the pixels. The resolution of the light-field texture is rather small (256×256), to show that the gaussians can be interpolated with very plausible results.

For the three rightmost objects (roughness $>= 0.3$), the reconstructed light-fields are sufficient to produce an image that matches

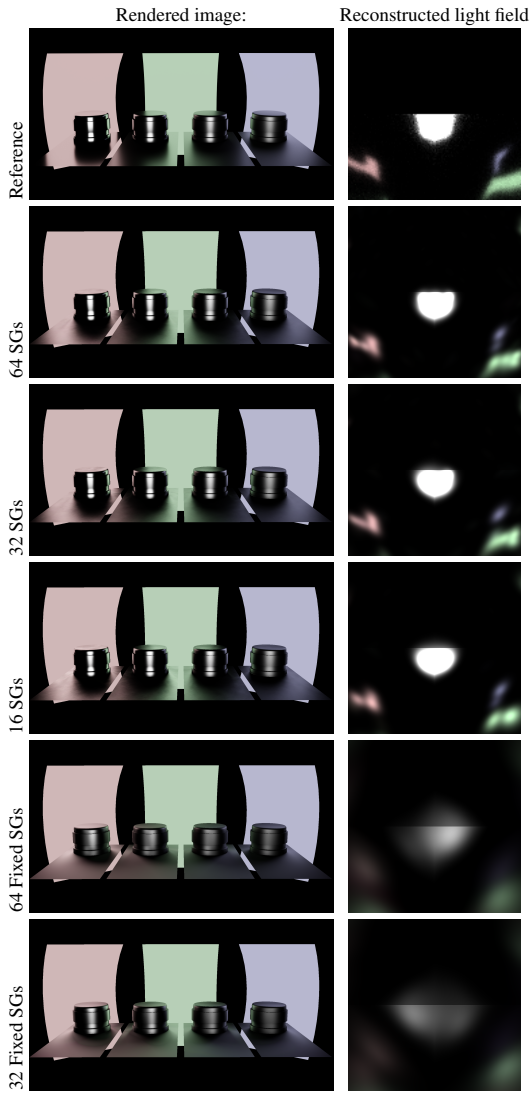


Table 2: *Quality of reflections compared to a path-traced reference for varying numbers of SGs per texel. In the right column is the reconstructed light field for one pixel.*

the path-traced reference quite well, even with 16 SGs. At lower roughness levels, the remaining errors become more obvious and on the clear, flat plane the reflections might not be acceptable even with 64 SGs. On a curved object, or a textured material (see, e.g., Figure 1b), the quality of highly glossy reflections can be quite sufficient with as few as 16 SGs. Looking at the leftmost plane, We can identify two main sources of error. First, since the reconstructed light-field image consists only of a sum of gaussians, straight, hard lines are difficult to reconstruct, leading to somewhat smudgy reflections. Secondly, in some places we can see what looks like folds in the flat plane. These artifacts appear when a SG changes direction quickly over a few pixels, which the network might deem necessary to reduce the overall error.

The two bottom rows show the results when using fixed directions as in previous work [Pet16]. Here, even the most rough material is clearly not comparable to the path-traced reference, and the errors are even more visible in motion, as can be seen in the accompanying video. This is not surprising when looking at the corresponding light-field image. The available SGs are necessarily spread uniformly over the sphere and the majority of them do not contribute at all.

In Table 3, we show a similar scene, but illuminated by an environment map, and using our precomputed illumination-weighted environment visibility method. While reflections are not quite as sharp as in the path-traced reference, our method works as a very convincing visibility estimator for any direction even with this quite challenging, high frequency, HDR environment map. Note in the right column that the reconstructed illumination weighted environment visibility is not a simple visibility map, but an attenuation factor for the preconvolved environment map.

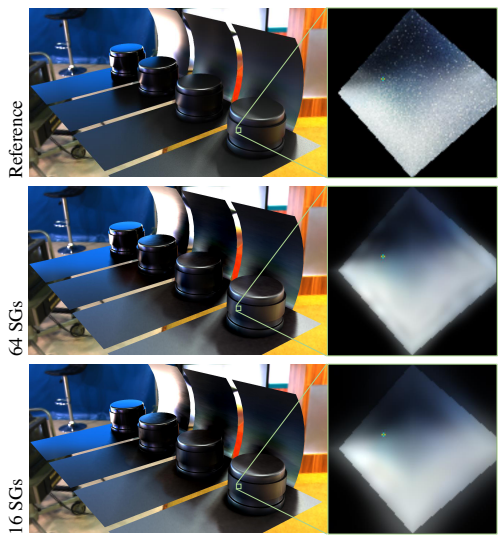
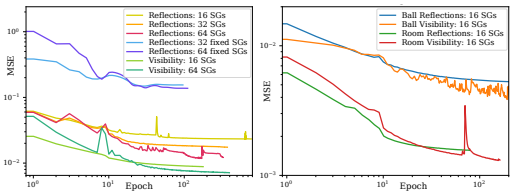


Table 3: *Quality of illumination weighted environment visibility compared to a path-traced reference for varying numbers of SGs per texel. In the right column is the reconstructed visibility for one pixel.*

Convergence. In Figures 7a and 7b, we show the loss as a function of the number of epochs the network has been trained. In all



(a) (b)

Figure 7: *Convergence of the networks trained for Tables 2 and 3, and for Figure 1.*

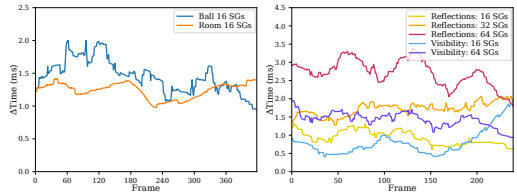


Figure 9: *The time taken for our deferred shading pass, including evaluation of indirect illumination from light-field textures, for each frame of the accompanying videos.*

of our tests, the MSE improves only very slightly after 128 epochs, and in general, a good result is obtained after 30 epochs. In the first graph, each epoch took approximately one minute to train, and in Figure 7b, which has a larger light-field texture (512×512), each epoch took approximately five minutes. In Figure 7a, we also show the convergence when training for fixed directions. Here, we could reach convergence by directly optimizing the parameters with gradient descent.

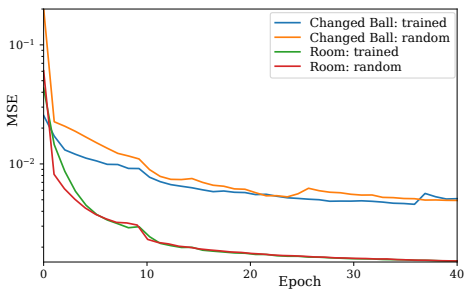


Figure 8: *Convergence when starting with a pre-trained network. The network is initialized with the values obtained from training for Figure 1b and then trained for two different scenes. The Changed Ball scene is very similar to the one the network is trained for, and the Room scene is the scene shown in 1a.*

To evaluate how general the trained network is, we have experimented with initializing the network weights with the converged weights for a different scene. The results are shown in Figure 8. We trained the scene shown in Figure 1b to obtain an initial network state and then trained two different scenes. One of these scenes was obtained by moving objects around in the original scene, and the other is the scene shown in Figure 1a. Although the MSE obtained after the first few epochs was slightly better than for random initialization, we did not find that a pre-trained network improved convergence in either case. We believe the old input images, although similar, do not contain sufficiently similar features for the network to generalize.

Performance. Finally, in Figure 9, we show the time taken to render each frame of the accompanying videos. All images are rendered at a resolution of 1920×1080, and the times shown are the time taken for the *Deferred Shading* pass, which evaluates all SGs (the total frametime includes an additional 0.2ms for rendering the GBuffer). Not unexpectedly, the performance is mostly proportional to the number of SGs evaluated. Since evaluating and convolving spherical gaussians is very cheap, the costly part of our approach is the number of texture fetches required. That memory is the bottleneck is further evidenced by the fact that performance improves significantly when we use 16 bit floating point values rather than 32 bit to describe our SG parameters. Since using 16 bit floats has no visible impact, that is what we have used in all meassurements in this paper. We also attempted to further reduce the size of our light-field textures by converting them to 8 bit values. This had a significant impact on quality, however, and did not improve performance much. To reduce the memory footprint further, it might instead be possible to use any of the compressed texture formats available in hardware, but we have not yet explored this further.

8. Conclusion and Future Work

We have shown that the quality of light-field textures, represented by Spherical Gaussians, can be greatly increased by allowing for arbitrary axes. We suggest training a Convolutional Neural Network to produce appropriate parameters for these SGs, rather than optimizing the spherical gaussians' parameters directly, and show that good results are obtained, for complex scenes, within a few hours of training. Additionally, we suggest a novel method for approximating environment visibility, by precomputing the *illumination weighted environment visibility*, and show that the same network can be used to create the SGs describing this function. Our real-time indirect illumination algorithm is extremely fast on modern high-end hardware and should perform well within real-time even on much older hardware or even portable devices.

Generating one of the converged ground-truth images shown in Figure 5e takes about 5 minutes on our RTX 2080 card, with an Optix renderer. A noisy, but recognizable, picture can be rendered within seconds. By significantly simplifying the allowed types of light-transport and scene-geometry, and making heavy use of temporal denoising filters, a pathtraced image can be obtained at interactive framerates (see e.g, QuakeRTX). For high-quality scenes and arbitrary lighting, fully dynamic solutions are still not available, however. Our method admittedly requires hours of baking and training as a preprocess (around 8 hours of baking and 6 of training), but allows for good quality global-illumination images for a time budget of 1-2 milliseconds per frame (see Figure 9).

In this work, we have concentrated on storing the SG parameters in two-dimensional textures, but another promising area would be to approximate densely placed light probes, which would allow dynamic objects to reflect the static scene. Although our examples do not require much memory for the light-field textures, a larger scene might require much higher resolution and then the memory cost of our method would naturally grow. Therefore, another interesting area of future work is to further compress the light-field data. This could be achieved as simply as using hardware compression for the textures, or it might be possible to take advantage of the coherency between texels in the light-field texture.

9. Acknowledgments

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

References

[AAL16] AITTALA, MIIKA, AILA, TIMO, and LEHTINEN, JAAKKO. "Reflectance Modeling by Neural Texture Synthesis". *ACM Trans. Graph.* 35.4 (2016) 5.

[BN76] BLINN, JAMES F. and NEWELL, MARTIN E. "Texture and Reflection in Computer Generated Images". *Commun. ACM* 19.10 (Oct. 1976), 542–547. ISSN: 0001-0782. DOI: 10.1145/360349.360353. URL: http://doi.acm.org/10.1145/360349.360353 3.

[CGL*19] CHEN, WENZHENG, GAO, JUN, LING, HUAN, et al. "Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer". *ArXiv* abs/1908.01210 (2019) 4.

[CKS*17] CHAITANYA, CHAKRAVARTY R. ALLA, KAPLANYAN, ANTON S., SCHIED, CHRISTOPH, et al. "Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder". *ACM Trans. Graph.* 36.4 (July 2017), 98:1–98:12. ISSN: 0730-0301. DOI: 10.1145/3072959.3073601. URL: http://doi.acm.org/10.1145/3072959.3073601 1, 4.

[CNS*11] CRASSIN, CYRIL, NEYRET, FABRICE, SAINZ, MIGUEL, et al. "Interactive Indirect Illumination Using Voxel Cone Tracing: A Preview". *Symposium on Interactive 3D Graphics and Games*. I3D '11. San Francisco, California: ACM, 2011, 207–207. ISBN: 978-1-4503-0565-5. DOI: 10.1145/1944745.1944787. URL: http://doi.acm.org/10.1145/1944745.1944787 4.

[CWZ*18] CHEN, ANPEI, WU, MINYE, ZHANG, YINGLIANG, et al. "Deep Surface Light Fields". *Proc. ACM Comput. Graph. Interact. Tech.* 1.1 (July 2018), 14:1–14:17. ISSN: 2577-6193. DOI: 10.1145/3203192. URL: http://doi.acm.org/10.1145/3203192 5.

[Deb06] DEBEVEC, PAUL. "Image-based Lighting". *ACM SIGGRAPH 2006 Courses*. SIGGRAPH '06. Boston, Massachusetts: ACM, 2006. ISBN: 1-59593-364-6. DOI: 10.1145/1185657.1185686. URL: http://doi.acm.org/10.1145/1185657.1185686 3.

[FBD*19] FLYNN, JOHN, BROXTON, MICHAEL, DEBEVEC, PAUL E., et al. "DeepView: View Synthesis With Learned Gradient Descent". *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), 2362–2371 4.

[FNPS16] FLYNN, JOHN, NEULANDER, IVAN, PHILBIN, JAMES, and SNAVELY, NOAH. "Deep Stereo: Learning to Predict New Views from the World's Imagery". *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 5515–5524 4.

[GKMD06] GREEN, PAUL, KAUTZ, JAN, MATUSIK, WOJCIECH, and DURAND, FRÉDO. "View-dependent precomputed light transport using nonlinear gaussian function approximations". *In ACM Symposium on Interactive 3D graphics*. 2006, 7–14 3–5, 8.

[GvSS17] GREFF, KLAUS, van STEENKISTE, SJOERD, and SCHMIDHUBER, JÜRGEN. "Neural Expectation Maximization". *NIPS*. 2017 4.

[HHM18] HEITZ, ERIC, HILL, STEPHEN, and MCGUIRE, MORGAN. "Combining Analytic Direct Illumination and Stochastic Shadows". *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '18. Montreal, Quebec, Canada: ACM, 2018, 2:1–2:11. ISBN: 978-1-4503-5705-0. DOI: 10.1145/3190834.3190852. URL: http://doi.acm.org/10.1145/3190834.3190852 2, 7.

[HMRR18] HERMOSILLA, PEDRO, MAISCH, SEBASTIAN, RITSCHEL, TOBIAS, and ROPINSKI, TIMO. "Deep-learning the Latent Space of Light Transport". *Comput. Graph. Forum* 38 (2018), 207–217 4.

[HSRG07] HAN, CHARLES, SUN, BO, RAMAMOORTHI, RAVI, and GRINSPUN, EITAN. "Frequency Domain Normal Map Filtering". *ACM Trans. Graph.* 26.3 (July 2007). ISSN: 0730-0301. DOI: 10.1145/1276377.1276412. URL: http://doi.acm.org/10.1145/1276377.1276412 4, 5, 8.

[HZE*19] HERHOLZ, SEBASTIAN, ZHAO, YANGYANG, ELEK, OSKAR, et al. "Volume Path Guiding Based on Zero-Variance Random Walk Theory". *ACM Trans. Graph.* 38.3 (June 2019). ISSN: 0730-0301. DOI: 10.1145/3230635. URL: https://doi.org/10.1145/3230635 8.

[IFDN12] IWASAKI, KEI, FURUYA, WATARU, DOBASHI, YOSHINORI, and NISHITA, TOMOYUKI. "Real-time Rendering of Dynamic Scenes under All-frequency Lighting using Integral Spherical Gaussian". *Comput. Graph. Forum* 31 (2012), 727–734 3.

[Kaj86] KAJIYA, JAMES T. "The Rendering Equation". *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), 143–150. ISSN: 0097-8930. DOI: 10.1145/15886.15902. URL: http://doi.acm.org/10.1145/15886.15902 1.

[KD10] KAPLANYAN, ANTON and DACHSBACHER, CARSTEN. "Cascaded Light Propagation Volumes for Real-time Indirect Illumination". *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '10. Washington, D.C.: ACM, 2010, 99–107. ISBN: 978-1-60558-939-8. DOI: 10.1145/1730804.1730821. URL: http://doi.acm.org/10.1145/1730804.1730821 4.

[KWR16] KALANTARI, NIMA KHADEMI, WANG, TING-CHUN, and RAMAMOORTHI, RAVI. "Learning-based View Synthesis for Light Field Cameras". *ACM Trans. Graph.* 35.6 (Nov. 2016), 193:1–193:10. ISSN: 0730-0301. DOI: 10.1145/2980179.2980251. URL: http://doi.acm.org/10.1145/2980179.2980251 4.

[LADL18] LI, TZU-MAO, AITTALA, MIIKA, DURAND, FRÉDO, and LEHTINEN, JAAKKO. "Differentiable Monte Carlo ray tracing through edge sampling". *ACM Trans. Graph.* 37 (2018), 222:1–222:11 4.

[LHJ19] LOUBET, GUILLAUME, HOLZSCHUCH, NICOLAS, and JAKOB, WENZEL. "Reparameterizing discontinuous integrands for differentiable rendering". *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38.6 (Dec. 2019). DOI: 10.1145/3355089.3356510 4.

[MB18] MEDER, JULIAN and BRÜDERLIN, BEAT D. "Hemispherical Gaussians for Accurate Light Integration". *ICCVG*. 2018 4.

[Mit07] MITTRING, MARTIN. "Finding Next Gen: CryEngine 2". *ACM SIGGRAPH 2007 Courses*. SIGGRAPH '07. San Diego, California: ACM, 2007, 97–121. ISBN: 978-1-4503-1823-5. DOI: 10.1145/1281500.1281671. URL: http://doi.acm.org/10.1145/1281500.1281671 4.

[MKU13] MIANDJI, EHSAN, KRONANDER, JOEL, and UNGER, JONAS. "Learning Based Compression of Surface Light Fields for Real-time Rendering of Global Illumination Scenes". *SIGGRAPH Asia 2013 Technical Briefs*. 24. ACM, 2013, 24:1–24:4 5.

[ML09] MCGUIRE, MORGAN and LUEBKE, DAVID. "Hardware-accelerated Global Illumination by Image Space Photon Mapping". *Proceedings of the Conference on High Performance Graphics 2009*. HPG '09. New Orleans, Louisiana: ACM, 2009, 77–89. ISBN: 978-1-60558-603-8. DOI: 10.1145/1572769.1572783. URL: http://doi.acm.org/10.1145/1572769.1572783 4.

[MM14] MCGUIRE, MORGAN and MARA, MICHAEL. "Efficient GPU Screen-Space Ray Tracing". *Journal of Computer Graphics Techniques (JCGT)* 3.4 (Dec. 2014), 73–85. ISSN: 2331-7418. URL: http://jcgt.org/published/0003/04/04/ 4.

[MMBJ17] MARA, MICHAEL, MCGUIRE, MORGAN, BITTERLI, BENEDIKT, and JAROSZ, WOJCIECH. "An Efficient Denoising Algorithm for Global Illumination". *ACM SIGGRAPH / Eurographics High Performance Graphics*. HPG 2017. July 2017, 7. URL: https://casual-effects.com/research/Mara2017Denoise/index.html 1, 4.

[MMNL17] MCGUIRE, MORGAN, MARA, MIKE, NOWROUZEZAHRAI, DEREK, and LUEBKE, DAVID. "Real-time Global Illumination Using Precomputed Light Field Probes". *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '17. San Francisco, California: ACM, 2017, 2:1–2:11. ISBN: 978-1-4503-4886-7. DOI: 10.1145/3023368.3023378. URL: http://doi.acm.org/10.1145/3023368.3023378 4.

[MRF18] MAXIMOV, MAXIM, RITSCHEL, TOBIAS, and FRITZ, MARIO. "Deep Appearance Maps". *ArXiv* abs/1804.00863 (2018) 4.

[MS16] MANSON, JOSIAH and SLOAN, PETER-PIKE. "Fast Filtering of Reflection Probes". *Comput. Graph. Forum* 35.4 (July 2016), 119–127. ISSN: 0167-7055. DOI: 10.1111/cgf.12955. URL: https://doi.org/10.1111/cgf.12955 3.

[MSK*16] MOREAU, P., SINTORN, E., KÄMPE, V., et al. "Photon Splatting Using a View-sample Cluster Hierarchy". *Proceedings of High Performance Graphics*. HPG '16. Dublin, Ireland: Eurographics Association, 2016, 75–85. ISBN: 978-3-03868-008-6. DOI: 10.2312/hpg.20161194. URL: https://doi.org/10.2312/hpg.20161194 4.

[MSS*10] MEYER, QUIRIN, SUSSMUTH, JOCHEN, SUSSNER, GERD, et al. "On Floating-point Normal Vectors". *Proceedings of the 21st Eurographics Conference on Rendering*. EGSR'10. Saarbrücken, Germany: Eurographics Association, 2010, 1405–1409. DOI: 10.1111/j.1467-8659.2010.01737.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2010.01737.x 5.

[PBD*10] PARKER, STEVEN G., BIGLER, JAMES, DIETRICH, ANDREAS, et al. "OptiX: A General Purpose Ray Tracing Engine". *ACM Trans. Graph.* 29.4 (July 2010), 66:1–66:13. ISSN: 0730-0301. DOI: 10.1145/1778765.1778803. URL: http://doi.acm.org/10.1145/1778765.1778803 5.

[Pet16] PETTINEO, MATT. *The Danger Zone: SG Series*. 2016. URL: https://mynameismjp.wordpress.com/2016/10/09/sg-series-part-1-a-brief-and-incomplete-history-of-baked-lighting-representations/ (visited on 05/07/2019) 2, 8, 9, 14.

[Rak18] RAKHTEENKO, ARTHUR. *Baking artifact-free lightmaps on the GPU*. 2018. URL: https://ndotl.wordpress.com/2018/08/29/baking-artifact-free-lightmaps/ (visited on 05/07/2019) 5.

[RDGK12] RITSCHEL, TOBIAS, DACHSBACHER, CARSTEN, GROSCH, THORSTEN, and KAUTZ, JAN. "The State of the Art in Interactive Global Illumination". *Comput. Graph. Forum* 31.1 (Feb. 2012), 160–188. ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.02093.x. URL: https://doi.org/10.1111/j.1467-8659.2012.02093.x 4.

[RH01] RAMAMOORTHI, RAVI and HANRAHAN, PAT. "An Efficient Representation for Irradiance Environment Maps". *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: ACM, 2001, 497–500. ISBN: 1-58113-374-X. DOI: 10.1145/383259.383317. URL: http://doi.acm.org/10.1145/383259.383317 2, 3.

[RWG*13] REN, PEIRAN, WANG, JIAPING, GONG, MINMIN, et al. "Global illumination with radiance regression functions". *ACM Trans. Graph.* 32 (2013), 130:1–130:12 4.

[RWS*06] REN, ZHONG, WANG, RUI, SNYDER, JOHN, et al. "Real-time Soft Shadows in Dynamic Scenes Using Spherical Harmonic Exponentiation". *ACM Trans. Graph.* 25.3 (July 2006), 977–986. ISSN: 0730-0301. DOI: 10.1145/1141911.1141982. URL: http://doi.acm.org/10.1145/1141911.1141982 3.

[Seb14] SEBASTIEN LAGARDE, CHARLES DE ROUSIERS. *Moving Frostbite to PBR, SIGGRAPH 2014 Course: Physically Based Shading in Theory and Practice.* Aug. 2014. URL: https://seblagarde.wordpress.com/2015/07/14/siggraph-2014-moving-frostbite-to-physically-based-rendering/ 6.

[SKS02] SLOAN, PETER-PIKE, KAUTZ, JAN, and SNYDER, JOHN. "Precomputed Radiance Transfer for Real-time Rendering in Dynamic, Low-frequency Lighting Environments". *ACM Trans. Graph.* 21.3 (July 2002), 527–536. ISSN: 0730-0301. DOI: 10.1145/566654.566612. URL: http://doi.acm.org/10.1145/566654.566612 3.

[SLS05] SLOAN, PETER-PIKE, LUNA, BEN, and SNYDER, JOHN. "Local, Deformable Precomputed Radiance Transfer". *ACM Trans. Graph.* 24.3 (July 2005), 1216–1224. ISSN: 0730-0301. DOI: 10.1145/1073204.1073335. URL: http://doi.acm.org/10.1145/1073204.1073335 3.

[SWS*17] SRINIVASAN, PRATUL P., WANG, TONGZHOU, SREELAL, ASHWIN, et al. "Learning to Synthesize a 4D RGBD Light Field from a Single Image". *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), 2262–2270 4.

[SZ12] SÉBASTIEN, LAGARDE and ZANUTTINI, ANTOINE. "Local Image-based Lighting with Parallax-corrected Cubemaps". *ACM SIGGRAPH 2012 Talks*. SIGGRAPH '12. Los Angeles, California: ACM, 2012, 36:1–36:1. ISBN: 978-1-4503-1683-5. DOI: 10.1145/2343045.2343094. URL: http://doi.acm.org/10.1145/2343045.2343094 3.

[TF17] THOMAS, MANU MATHEW and FORBES, ANGUS GRAEME. "Deep Illumination: Approximating Dynamic Global Illumination with Generative Adversarial Network". *CoRR* abs/1710.09834 (2017). arXiv: 1710.09834. URL: http://arxiv.org/abs/1710.09834 5.

[TS06] TSAI, YU-TING and SHIH, ZEN-CHUNG. "All-frequency Precomputed Radiance Transfer Using Spherical Radial Basis Functions and Clustered Tensor Approximation". *ACM Trans. Graph.* 25.3 (July 2006), 967–976. ISSN: 0730-0301. DOI: 10.1145/1141911.1141981. URL: http://doi.acm.org/10.1145/1141911.1141981 2–4.

[TS92] TORRANCE, K. E. and SPARROW, E. M. "Radiometry". Ed. by WOLFF, LAWRENCE B., SHAFER, STEVEN A., and HEALEY, GLENN. USA: Jones and Bartlett Publishers, Inc., 1992. Chap. Theory for Off-specular Reflection from Roughened Surfaces, 32–41. ISBN: 0-86720-294-7. URL: http://dl.acm.org/citation.cfm?id=136913.136924 6.

[VKŠ*14] VORBA, JIŘÍ, KARLÍK, ONDŘEJ, ŠIK, MARTIN, et al. "On-line Learning of Parametric Mixture Models for Light Transport Simulation". *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)* 33.4 (Aug. 2014) 4.

[WMLT07] WALTER, BRUCE, MARSCHNER, STEPHEN R., LI, HONGSONG, and TORRANCE, KENNETH E. "Microfacet Models for Refraction Through Rough Surfaces". *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. Grenoble, France: Eurographics Association, 2007, 195–206. ISBN: 978-3-905673-52-4. DOI: 10.2312/EGWR/EGSR07/195-206. URL: http://dx.doi.org/10.2312/EGWR/EGSR07/195-206 8.

[WRG*09] WANG, JIAPING, REN, PEIRAN, GONG, MINMIN, et al. "All-frequency Rendering of Dynamic, Spatially-varying Reflectance". *ACM Trans. Graph.* 28.5 (Dec. 2009), 133:1–133:10. ISSN: 0730-0301. DOI: 10.1145/1618452.1618479. URL: http://doi.acm.org/10.1145/1618452.1618479 2–4, 8.

[WRM17] WANG, TUANFENG Y., RITSCHEL, TOBIAS, and MITRA, NILOY JYOTI. "Joint Material and Illumination Estimation from Photo Sets in the Wild". *2018 International Conference on 3D Vision (3DV)* (2017), 22–31 4.

[XCM*14] XU, KUN, CAO, YAN-PEI, MA, LI-QIAN, et al. "A practical algorithm for rendering interreflections with all-frequency BRDFs". *ACM Trans. Graph.* 33 (2014), 10:1–10:16 4.

[XSD*13] XU, KUN, SUN, WEI-LUN, DONG, ZHAO, et al. "Anisotropic Spherical Gaussians". *ACM Trans. Graph.* 32.6 (2013), 209:1–209:11 3, 4.

[ZBLN97] ZHU, CIYOU, BYRD, RICHARD H., LU, PEIHUANG, and NOCEDAL, JORGE. "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization". *ACM Trans. Math. Softw.* 23.4 (Dec. 1997), 550–560. ISSN: 0098-3500. DOI: 10.1145/279232.279236. URL: https://doi.org/10.1145/279232.279236 4.

[ZTF*18] ZHOU, TINGHUI, TUCKER, RICHARD, FLYNN, JOHN, et al. "Stereo Magnification: Learning View Synthesis Using Multiplane Images". *ACM Trans. Graph.* 37.4 (July 2018), 65:1–65:12. ISSN: 0730-0301. DOI: 10.1145/3197517.3201323. URL: http://doi.acm.org/10.1145/3197517.3201323 4.

Appendix A: Partial Derivatives for back propagation

We use the L_2 log loss function when training, defined as follows:

$$L(P) = (\log(T+1) - \log(P+1))^2, \tag{9}$$

where P is the predicted value for a pixel in a light-field image, and T is the target value. The gradient of the L_2 log loss function with respect to this pixel is:

$$\frac{dL(P)}{dP} = -2\frac{\log(1+T) - \log(1+P)}{1+P} \tag{10}$$

However, we need to backpropagate the gradient of the L_2 loss with respect to each *parameter*. The chain rule gives us that the gradient of a specific parameter $\mathbf{p}_k$ is:

$$\frac{dL}{d\mathbf{p}_k} = \frac{1}{IC}\sum_{i,c}\frac{dP_{ic}}{d\mathbf{p}_k}\frac{dL(P_{ic})}{dP_{ic}}, \tag{11}$$

where I is the total number of pixels and C is the number of channels. Hence, for each pixel and channel, we must find the partial derivative of Eq 1 with respect to each of the parameters and add that contribution to the parameter's gradient. For the different parameter types, these derivatives are:

$$\frac{dP_{ic}}{d\mu_{jc}} = e^{\lambda_j(\mathbf{v}_i\cdot\mathbf{p}_j - 1)} \tag{12}$$

$$\frac{dP_{ic}}{d\lambda_j} = \mu_{jc}e^{\lambda_j(\mathbf{v}_i\cdot\mathbf{p}_j - 1)}(\mathbf{v}_i\cdot\mathbf{p}_j - 1) \tag{13}$$

$$\frac{dP_{ic}}{dx} = \mu_{jc}e^{\lambda_j(\mathbf{v}_i\cdot\mathbf{p}_j - 1)}\lambda_{jx} \text{ (identically for } y \text{ and } z) \tag{14}$$

Appendix B: BRDF as Spherical Gaussians and SG Convolution

In the real-time rendering of the reflection SGs we treat the BRDF function as a spherical gaussian to be able to convolve it with the spherical gaussians describing the incoming light field for a texel. These formulas have been adapted from [Pet16], where an in-depth explanation on how they are derived is also given.

We use the Cook-Torrance BRDF:

$$f(\omega_i,\omega_o) = \frac{F(\omega_o,\omega_h)\,G(\omega_i,\omega_o,\omega_h)\,D(\omega_h)}{4(\mathbf{n}\cdot\omega_i)(\mathbf{n}\cdot\omega_o)}, \tag{15}$$

and the following definition of a Spherical Gaussian:

$$G(\mathbf{v};\mu,\lambda,a) = ae^{\lambda(\mu\cdot\mathbf{v}-1)}. \tag{16}$$

The SG approximation to the D term that we use is defined as follows:

$$D(\omega_h) = e^{-(arccos(\omega_h\cdot\mathbf{n})/r)^2} \approx G(\omega_h;\mathbf{n},\frac{2}{r^2},\frac{1}{\pi r^2}), \tag{17}$$

where r is the roughness of the material.

This gaussian is defined in the half-vector domain, and we need to convert it to the same domain as the SG it will be convolved with. To better represent the BRDF from directions approaching the surface plane we use an anisotropic transformation of the previous lobe in this step, as suggested by [Pet16]. For that, the following transformations are used:

$$\begin{aligned} \mu_w &= 2(\omega_o\cdot\mu_d)\mu_d - \omega_o \\ \lambda_w^x &= \frac{\lambda_d}{8\,max(\mu_d\cdot\omega_o, 0.0001)^2} \\ \lambda_w^y &= \frac{\lambda_d}{8} \\ a_w &= a_d \end{aligned} \tag{18}$$

This anisotropic spherical gaussian can be evaluated with:

$$G(\mathbf{v};[\mu_x,\mu_y,\mu_z],[\lambda_x,\lambda_y],a) = \tag{19}$$

$$= a\cdot max(\mathbf{v}\cdot\mu_z,0)e^{-\lambda_x(\mathbf{v}\cdot\mu_x)-\lambda_y(\mathbf{v}\cdot\mu_y)} \tag{20}$$

Where μ_x and μ_y are two orthogonal vectors that form a basis with $\mu_z = \mu_w$. Any will do, and they need to be transformed together with μ_z when applying equation 18.

We can convolve two spherical gaussians $G_1(\mathbf{v})$ and $G_2(\mathbf{v})$ as follows:

$$\int_\Omega G_1(\mathbf{v})G_2(\mathbf{v})d\mathbf{v} = \frac{4\pi a_1 a_2}{e^{\lambda_m}}\frac{sinh(||\mu_m||)}{||\mu_m||} \tag{21}$$

Where

$$\lambda_m = \lambda_1 + \lambda_2 \tag{22}$$

$$\mu_m = \frac{\lambda_1\mu_1 + \lambda_2\mu_2}{\lambda_1 + \lambda_2} \tag{23}$$

To convolve each isotropic SG from the light-field texture with the anisotropic SG approximating the D term, we use:

$$\begin{aligned} &\int_\Omega G_1(\mathbf{v};\mu_1,\lambda_1,a_1)\cdot G_2^A(\mathbf{v};\mu_2,[\lambda_2^x,\lambda_2^y],a_2)d\mathbf{v} = \\ &= \frac{a_1 a_2\pi}{\sqrt{(\frac{\lambda_1}{2}+\lambda_2^x)(\frac{\lambda_1}{2}+\lambda_2^y)}}max(\mu_2^z\cdot\mathbf{v})e^{-(\lambda_2^x(\mathbf{v}\cdot\mu_2^x)^2+\lambda_2^y(\mathbf{v}\cdot\mu_2^y)^2)}. \end{aligned} \tag{24}$$

Paper II

Real-Time Hair Filtering with Convolutional Neural Networks

Roc R. Currius, Erik Sintorn

Submitted, under review

Real-Time Hair Filtering with Convolutional Neural Networks

Roc R. Currius
Erik Sintorn
roc.ramon@chalmers.se
erik.sintorn@chalmers.se
Chalmers Institute of Technology
Gothenburg, Sweden

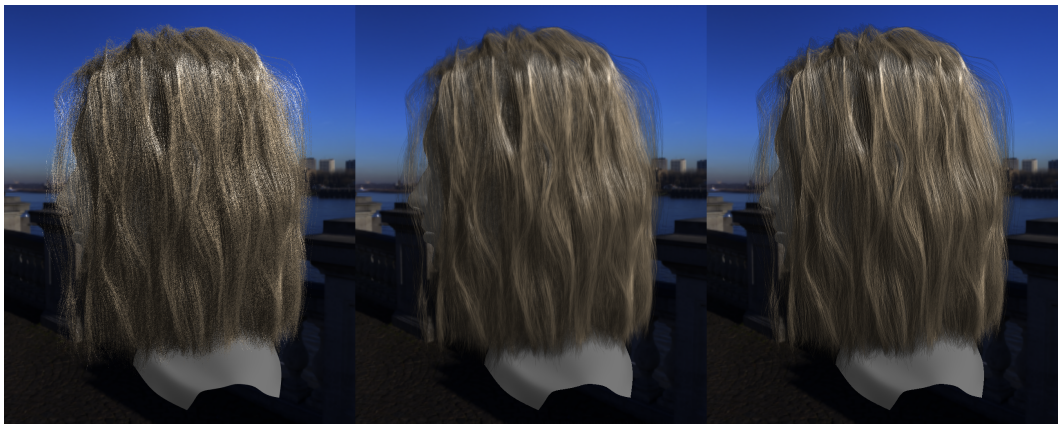


Figure 1: From left to right: hair rendered with stochastic transparency, filtered using our method, and the ground truth.

ABSTRACT

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**.

KEYWORDS

hair, real-time, transparency, filtering, neural networks

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 [33]. Hair fibers, being semi-transparent, scatter light in a complex manner[17], and while approximations exist[39], rendering hair with correct indirect lighting is still only possible in off-line renderers.

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; it 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[6, 33], 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[35] 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 [12, 23, 28].

Following Enderton et al.[7], many stochastic transparency methods have hair fragments randomly sampled by discarding fragments based on their transparency, 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[5], we train a U-Net[26] 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[9] suggested a first attempt at approximating a transfer function for hair. More physically accurate models have since been suggested by, e.g., Marschner et al. [17], Zinke et al. [39], and Sadeghi et al. [27]. 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 [11, 29]. As our suggested method for transparency is mostly orthogonal to the shading model used, we use the simple phenomenological model proposed by Scheuermann[29].

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 [32]. 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 [31] which, however, can be inefficient for complex hair geometry. The same authors later suggest approximating a per-pixel visibility function [32], but this method only works when all fragments have the same opacity, and quality deteriorates when the fragments are unevenly distributed. Adaptive Transparency [28] is a similar method, but 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. [23] also estimates a visibility function but with power, or trigonometric, moments. This results in a low frequency visibility function and is found by Kern et al. [12] to frequently over or under estimate visibility. For a survey of OIT methods we recommend the article by Maule et al. [18].

Rendering shadows cast by transparent objects is similarly difficult. An early method, intended for off-line rendering, is Deep Shadow Maps [15], in which an A-buffer is compressed to a piecewise linear visibility function per pixel. A real-time alternative, Opacity Shadow Maps [13], stores discrete functions in a 3D texture, and in Deep Opacity Maps [38] 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 [33], where alpha blending is avoided and MSAA is used to reduce aliasing. 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. [8], 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 [3]. The voxelized volume can also be used for self shadowing.

Enderton et al.[7] propose a method for rendering transparent objects in any order by randomly discarding each fragment based on its transparency (extending the idea of *Screen-door Transparency* [22]). 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 [19]. Laine and Karras[14] show that variance can be reduced by applying stratification techniques, 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[36] use a hash based on the discretized model-space position of each fragment to determine whether it should be discarded. 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 ground truth, alpha-blended, result.

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.[4] targets 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. In the work by Wei et al.[34], 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. 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.[25] instead use for input a style reference image and a processed hand drawing representing the desired shape of the hair. Similarly, Qiao and Kanai[24] apply a GAN to transfer the hairstyle from a reference image to an arbitrarily rendered image. They also target reproducing realistic lighting on the hair based on the scene.

NeRF-Tex [1] is a method for rendering fur by randomly distributing volumetric patches over a mesh. A *Neural Radiance Field* (NeRF) [21] 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

Much work has been done in attempting to filter out noise from stochastically sampled images and especially those produced from path tracing. Some recent methods have used neural networks to

improve the process. Kalantari et al.[10] use a network to decide filter parameters to use at each point of an input with few samples, together with scene information such as surface normals, positions, textures, etc., and obtain very accurate results. Similarly, Bako et al.[2] use a convolutional neural network to decide the best filter kernel for each point of their input features, treating diffuse and specular information in separate networks. These methods are intended for offline use, and so expect a higher quality input and include more complex networks than is feasible in real time.

With the advent of hardware accelerators for raytracing, several works in real-time denoising for pathtraced images have appeared. Notably, Chaitanya et al.[5] present a method to filter a very small number of samples of a pathtraced image with a convolutional neural network, similar to our work albeit with a rather expensive recurrent topology, and achieve interactive performance. Although the problem solved by this paper (reconstructing contigous geometry with sparsely sampled illumination) is quite different from ours (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. Mara et al.[16] suggest another method for filtering Monte Carlo pathtraced images with real-time performance where, in a similar manner, they separate "glossy" from "matte" terms during filtering and apply an algorithm based on cross-bilateral filters. They obtain results comparable to offline methods in less than 10ms of computation. More recently, Meng et al.[20] use a convolutional neural network to embed the noisy input in bilateral grids and then use bilateral filtering on these to produce high-quality results with real-time performance.

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.

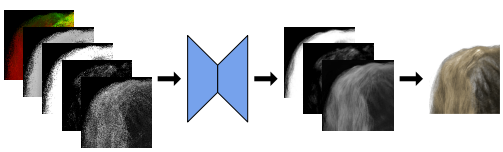


Figure 2: Overview of our method. Stochastically sampled color factor, highlight, alpha, depth and tangents are filtered with a CNN to obtain the filtered color factor, highlight, and alpha, which are composited to produce the final image.

3.2 Input Rendering

The true light transport in hair is very complex and indirect illumination effects are important [17]. In this paper, we consider only direct illumination and use a simplified approach described by Scheuermann[29, 30]. The formulas are shown in Figure 3. Real-time estimates to achieve indirect illumination exist [39] but are orthogonal to our work.

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

Figure 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 described by Scheuermann[29]; 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 Marschner et al.[17].

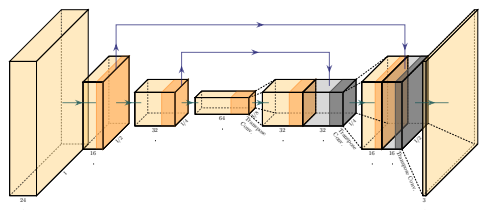


Figure 4: Diagram of the network used to filter the hair. Convolutions are followed by a ReLU. Encoding steps are downsampling convolutions, while decoding steps are transposed convolutions set up to obtain outputs double the size of the input. All convolutions are followed by ReLU.

To make the network capable of handling arbitrary hair colors, we separate the outgoing radiance of the hair into two components, one which is dependent on the color of the hair, $(F_d + F_{TRT})$ in Figure 3, and one that only depends 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 get more than one sample per pixel in a single pass, we apply the method from [7], using the coverage mask for a multisample buffer by setting or unsetting each of the bits with a probability of *alpha*.

The average of these samples is still an unbiased estimate but, as can be seen in Figure 5(c), it is still much too noisy even at high number of samples. Since rendering the hair at its actual size would produce aliasing, even at high sample rates, we render it 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, but 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[36], 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), 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, but we have found that, if removing either of our chosen features, the gains in performance do not compensate the loss in quality of the result. 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 alpha output is clipped to be between 0 and 1; the color factor output is calculated as $y = x^3/32 + x/5 + 1$, so as to provide more precision around 1; the specular highlight is only clipped to be positive.

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 - see Figure 9), taken from randomised distances and orientations of the camera and light. 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.

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.

Figure 6 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 5, 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

(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)

Figure 5: 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.

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.

(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.

9 shows the two hairstyles used for training, as well as the results when using a different hairstyle not used in the training. We see that our network generalizes well for that hairstyle as well.

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.

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 for each color, or to train the network to receive and produce RGB values for color and highlight.

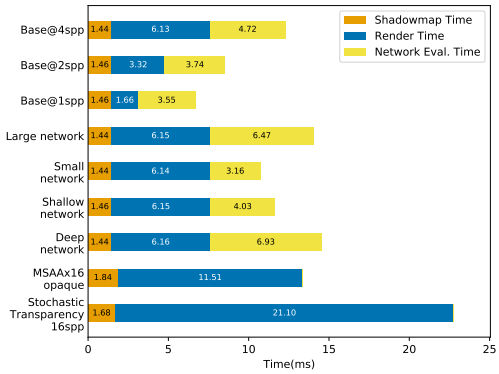


Figure 6: Performance timings for the execution of the network during real-time evaluation (*Network Eval. Time*), and the time to render the input for different network sizes (*Render Time*). Also included is the time taken for 16 samples of pure stochastic transparency, and for rendering with only MSAA, disregarding transparency. *Shadowmap Time* is the time taken to render the stochastic shadowmap at a resolution of 1024x1024, 1SPP.

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

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

Similarly, the presented algorithm is only intended to work with hair of uniform color. Training for 3-color channels for the input and output of the network, which would allow for multi-colored hair, increases complexity, and we haven't found it to work straightforwardly for RGB color space, 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. [5], 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.

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

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

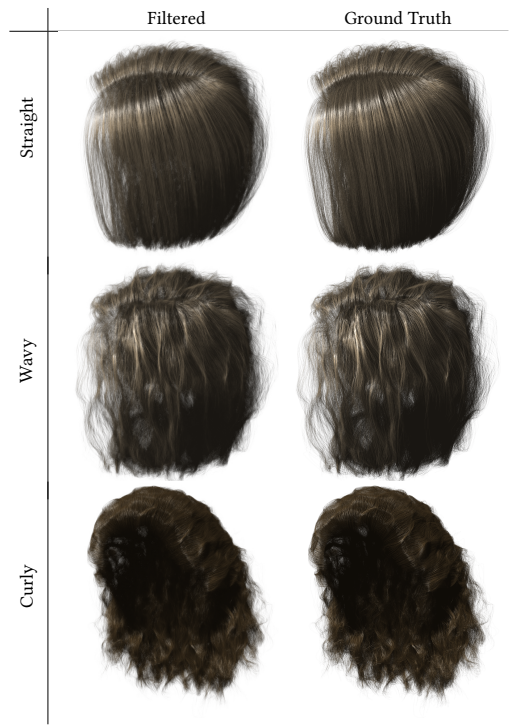


Figure 9: Results of our network for different hairstyles. The network was trained with images of both straight and wavy hair, but it also produces good results for hairstyles not used in the training.

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

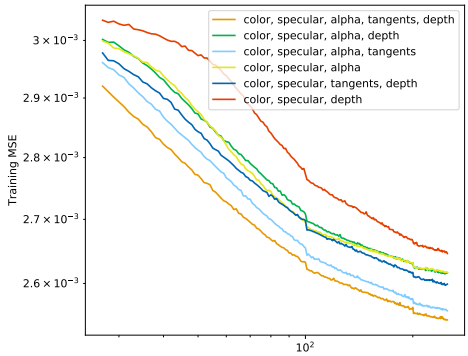


Figure 10: Plot of MSE during training up to 250 epochs for different sets of the input features to the network.

light, including color gradients in the hair, by using RGB channels or some intermediate color space.

The rendering of the stochastic input constitutes a large part of the total frame time (Figure 6). In order to achieve higher frame times, as well as less memory usage, the mesh could potentially be simplified while having the network still produce good-looking results.

ACKNOWLEDGMENTS

We would like to thank Yuksel for the hair meshes used throughout this paper[37].

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

REFERENCES

[1] 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

[2] Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan NovákK, 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

[3] 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

[4] 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

[5] 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

[6] 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.

[7] 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

[8] 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

[9] 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

[10] Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. *ACM Trans. Graph.* 34, 4 (July 2015), 1–12. https://doi.org/10.1145/2766977

[11] Brian Karis. 2016. Physically Based Hair Shading in Unreal.

[12] 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

[13] 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

[14] 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

[15] 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

[16] Michael Mara, Morgan McGuire, Benedikt Bitterli, and Wojciech Jarosz. 2017. An Efficient Denoising Algorithm for Global Illumination. In *Proceedings of High Performance Graphics*. ACM, Los Angeles California, 1–7. https://doi.org/10.1145/3105762.3105774

[17] 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

[18] 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

[19] 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

[20] Xiaoxu Meng, Quan Zheng, Amitabh Varshney, Gurprit Singh, and Matthias Zwicker. 2020. *Real-Time Monte Carlo Denoising with the Neural Bilateral Grid*. The Eurographics Association. https://doi.org/10.2312/sr.20201133

[21] 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]

[22] 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

[23] 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

[24] 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

[25] 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

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

[27] 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

[28] 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

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

[30] Thorsten Scheuermann. 2004. 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

[31] 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

[32] 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

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

[34] 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

[35] 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

[36] 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

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

[38] 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

[39] 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