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 typically 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 collects and summarises the following papers:

[**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.

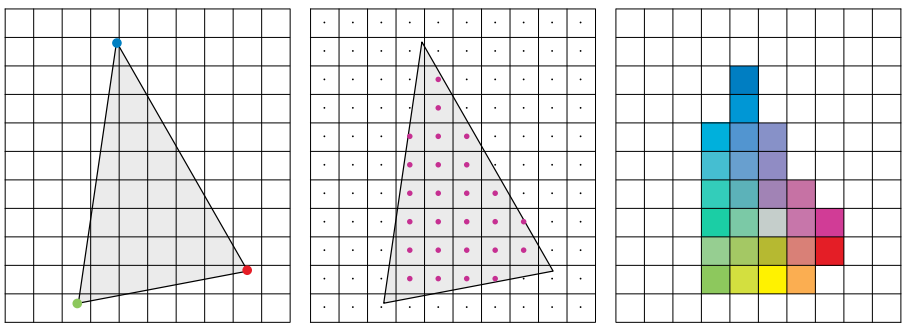


Figure 1.1: Rasterisation works by determining which pixels' centers are inside of the triangle, and then interpolating the values of the surface properties at the vertices.

Rendering of realistic scenes in real-time applications has improved im-

mensely 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:

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

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.

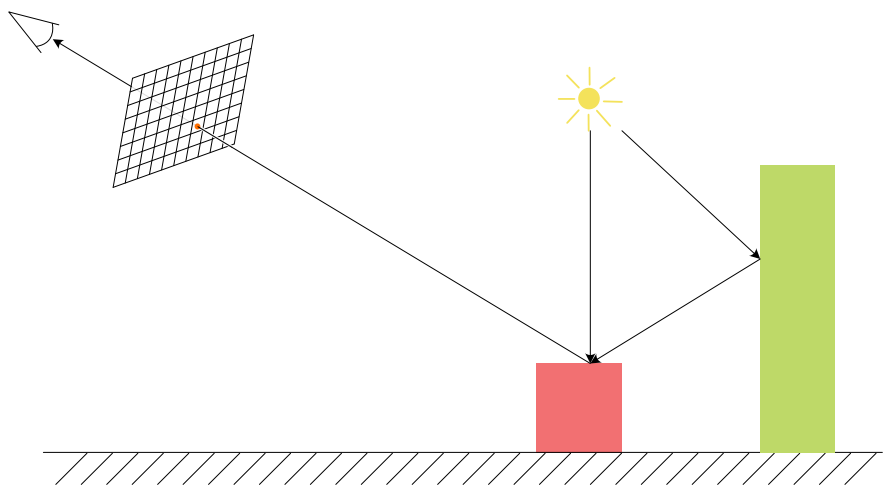


Figure 1.2: Raytracing follows the trajectory that a ray would travel from a point in the scene to the viewer's eye through the screen, and paints the pixel it crosses with the color from the point in the scene.

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.

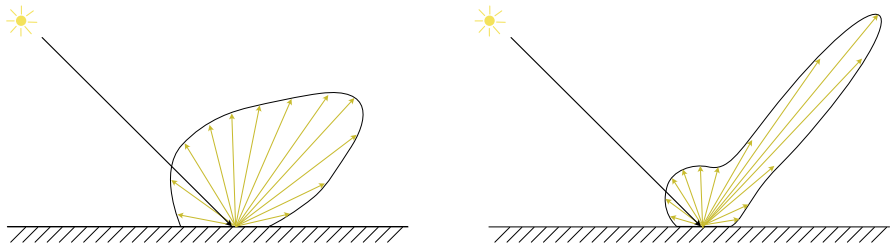


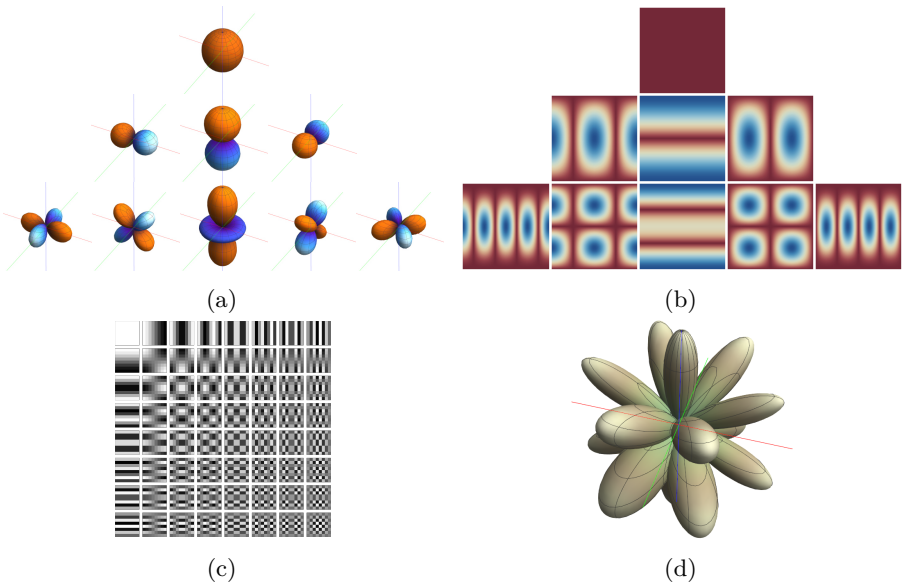
Figure 1.3: Side view of the BRDF directions for an incoming ray on rough (left) and smooth (right) materials.

A common approximation for indirect illumination in rasterised scenes uses *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.



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

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.

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 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.

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

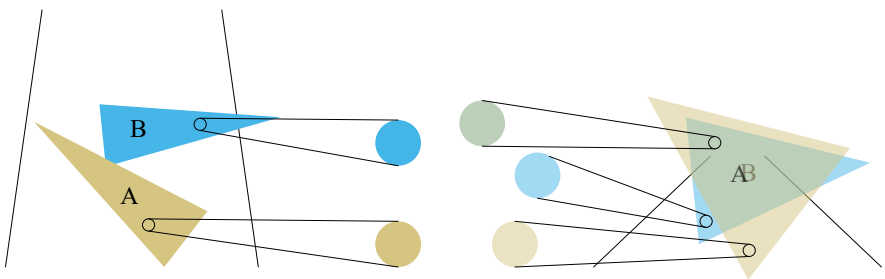


Figure 1.6: Semi-transparent object A is in front of object B, both with 50% transparency. Equation 1.3 can be used to calculate the color seen in the overlapping area.

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.

[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.

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 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.

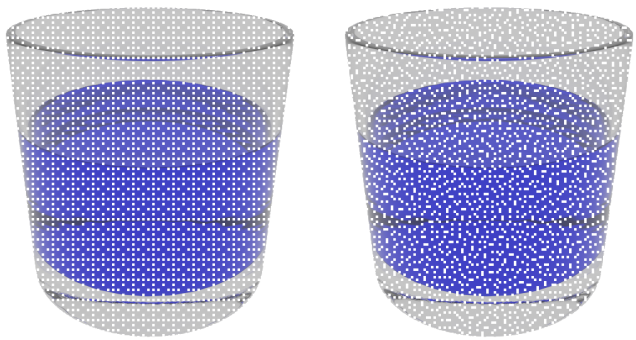


Figure 1.7: The basic screen-door transparency (*left*) uses a predefined pattern to decide which pixels are on and which off depending on the surface's transparency. Stochastic transparency (*right*) instead decides randomly for every pixel, eliminating patterning artifacts.

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.

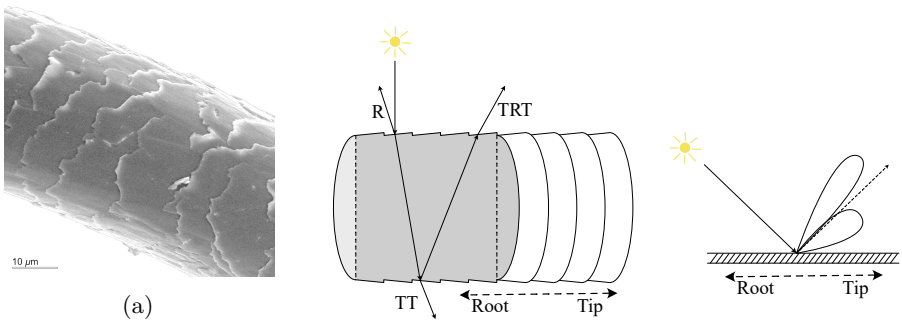


Figure 1.8: Surface of a hair fiber used for the lighting model. (a) is an electron micrograph of the surface of a hair fiber (*image by Nanjundaswamy [19] under Creative Commons BY-NC-SA 3.0 US*). (b) shows the simplification of the surface and the BRDF lobes by Marschner et al. [15].

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.

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.

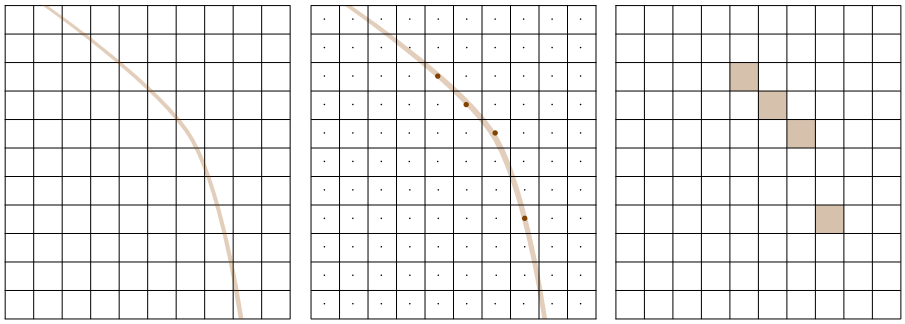


Figure 1.9: Aliasing when rendering hairs that are significantly thinner than a pixel — most pixels along the hair's path will not be shown because they do not cross the sampling point at the center of the pixel.

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.

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.

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*).

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.

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

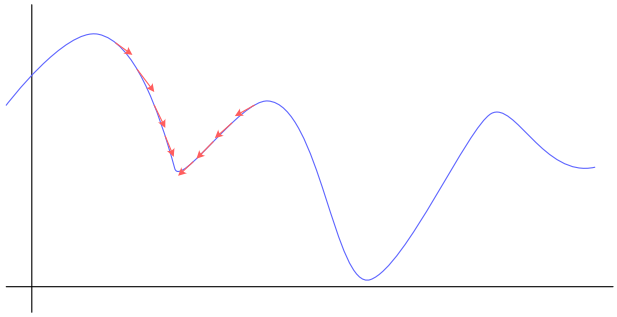


Figure 1.11: Gradient Descent algorithm: Following the direction of the gradient vector at a point converges to a local minima of the function.

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 distribution 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.

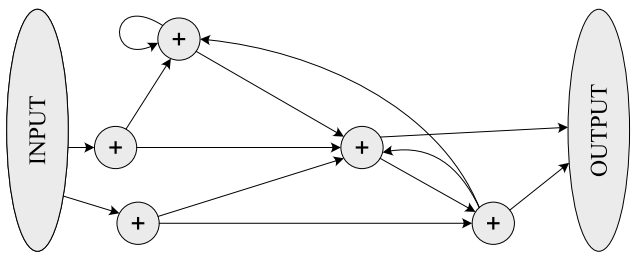


Figure 1.12: Several networks with several connections between them, representing the way the data from each will flow to each other. When updating them, the incoming values for each neuron will be added together to create its new value.

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.

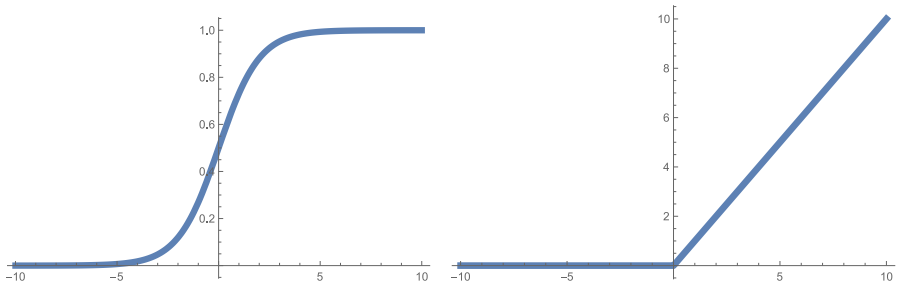


Figure 1.13: Plot of a sigmoid (left) and a ReLU (right) activation functions, typically used in artificial neural networks.

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.

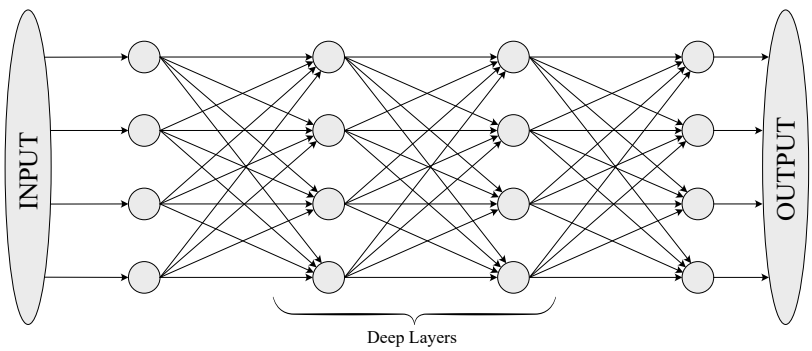


Figure 1.14: Multiple layers of neurons with full connectivity between every two adjacent layers.

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.

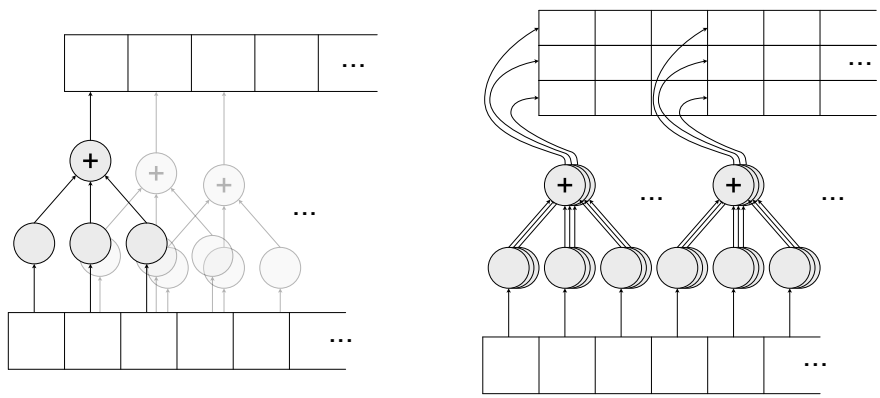


Figure 1.15: representation of a 1-dimensional convolutional neural network, where a network with fewer inputs than the total amount of data points is *convolved* with the data, i.e. the input is advanced and the network is applied again, storing all the results in a new list. Each different filter produces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

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 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

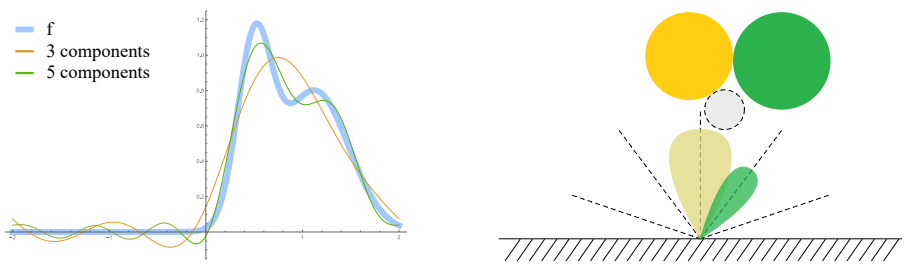
Summary of Included Papers

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.



(a) (b)

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

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.

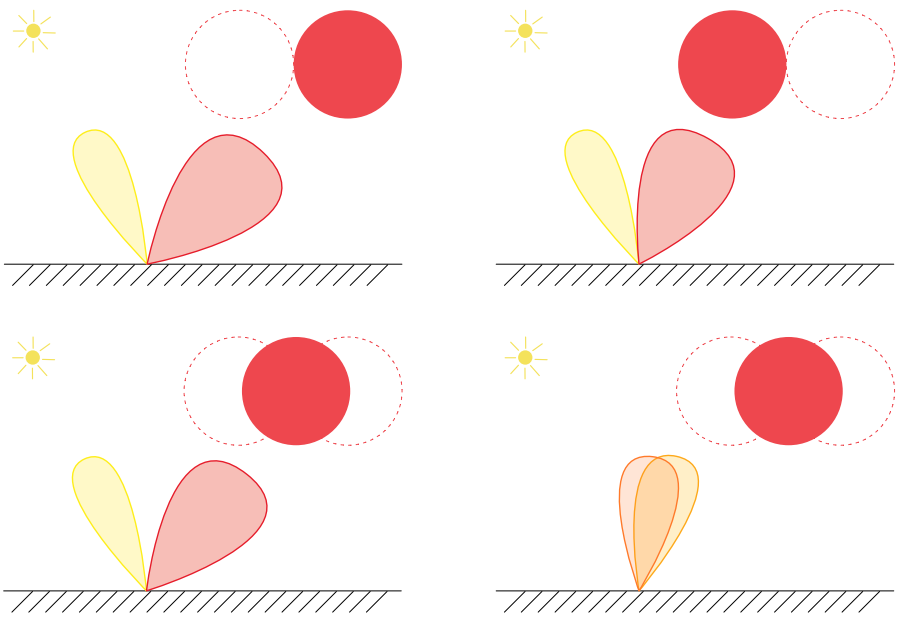


Figure 2.2: When attempting to fit spherical gaussians to a set of neighboring lightfields, the order needs to be considered, or the interpolation will produce undesired results. Top images show gaussians for different points in the scene, bottom-left is the expected interpolation of the gaussians for 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.

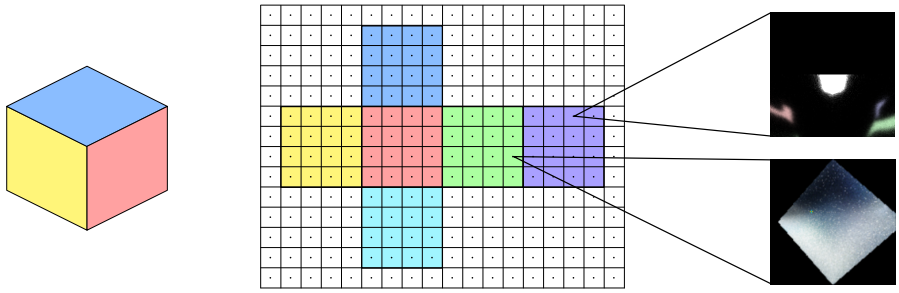


Figure 2.3: Texture coordinates used to create the lightfield map, and the lightfield image that is obtained for each of its cells.

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.

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

[1] See Appendix B of Paper I for details

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.

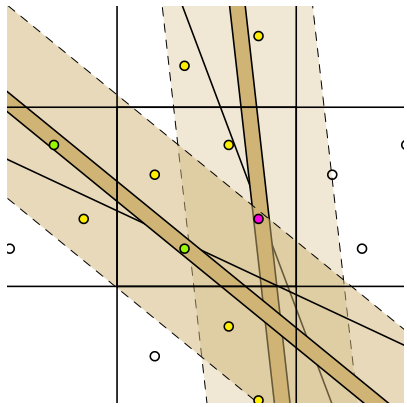


Figure 2.5: Hairs rendered at one-pixel thickness with transparency compensating for real thickness. The pink sample point shows why this approximation is slightly biased — that point will have information from both hairs, when it should only include information from one.

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 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 obtained 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 9, 21).

[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 9).

[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 21).

[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 9).

[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 16).

[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 12).

[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 19).

[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 16).

[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 page 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 21).

[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. 2016. SG Series Part 4: Specular Lighting From an SG Light Source. The Danger Zone. (10th October 2016). Retrieved 02/01/2022 from https://therealmjp.github.io/posts/sg-series-part-4-specular-lighting-from-an-sg-light-source/ (cited on page 20).

[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 16, 22).

[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 21).

[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 21).

[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 21).

[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).