Deep Shading: Convolutional Neural Networks for Screen-Space Shading

Oliver Nalbach
Max-Planck-Institut für Informatik

Elena Arabadzhiyska
Max-Planck-Institut für Informatik

Dushyant Mehta
Max-Planck-Institut für Informatik

Hans-Peter Seidel
Max-Planck-Institut für Informatik

Tobias Ritschel
University College London

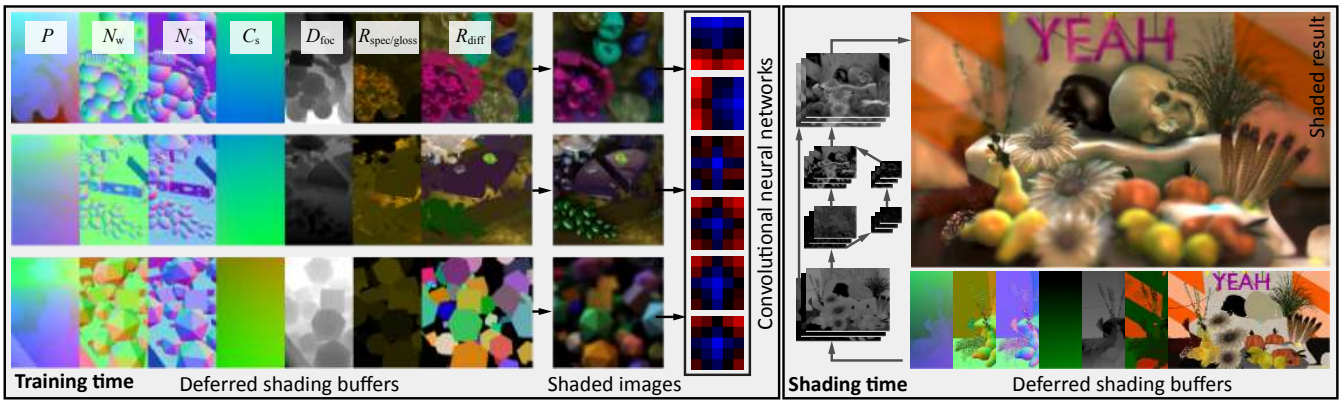

Figure 1: *In a training phase* (left)*, our approach learns a mapping from attributes in deferred shading buffers, e. g., positions, normals, reflectance, to RGB colors using a convolutional neural network (CNN). At runtime* (right)*, the CNN is used to jointly shade with environment lighting and shadows as well as depth-of-field at interactive rates (*512×384 *px, 1 ms rasterizing attributes, 209 ms network execution).*

Abstract

In computer vision, Convolutional Neural Networks (CNNs) have recently achieved new levels of performance for several inverse problems where RGB pixel appearance is mapped to attributes such as positions, normals or reflectance. In computer graphics, screen-space shading has recently increased the visual quality in interactive image synthesis, where per-pixel attributes such as positions, normals or reflectance of a virtual 3D scene are converted into RGB pixel appearance, enabling effects like ambient occlusion, indirect light, scattering, depth-of-field, motion blur, or anti-aliasing. In this paper we consider the diagonal problem: synthesizing appearance from given per-pixel attributes using a CNN. The resulting Deep Shading simulates all screen-space effects as well as arbitrary combinations thereof at competitive quality and speed while not being programmed by human experts but learned from example images.

Keywords: global illumination, convolutional neural networks, screen-space

Concepts: •**Computing methodologies** → **Neural networks; Rendering;** *Rasterization;*

1 Introduction

The move to deep architectures in Machine Learning has precipitated unprecedented levels of performance on various computer vision tasks, with several applications having the inverse problem of mapping image pixel RGB appearance to attributes such as positions, normals or reflectance as an intermediate or end objective. Deep architectures have further opened up avenues for several novel applications. In computer graphics, screen-space shading has been instrumental in increasing the visual quality in interactive image synthesis, employing per-pixel attributes such as positions, normals or reflectance of a virtual 3D scene to render RGB appearance that captures effects such as ambient occlusion, indirect light, scattering, depth-of-field, motion blur, and anti-aliasing.

In this paper we turn around the typical flow of information through Computer Vision Deep Learning pipelines to synthesize appearance from given per-pixel attributes, making use of Deep Convolutional architectures (CNNs). The resulting approach, which we call Deep Shading, can simulate all screen-space effects individually, as well as arbitrary combinations thereof at competitive quality and speed while not being explicitly programmed by human experts and rather learned from example images.

2 Previous Work

Previous work comes, on the one hand, from a computer graphics background where attributes have to be converted into appearance and, on the other hand, from a computer vision background where appearance has to be converted into attributes.

Attributes-to-appearance The rendering equation [Kajiya 1986] is a reliable forward model of appearance in the form of radiance incident at a virtual camera sensor when a three-dimensional description of the scene in form of attributes like positions, normals and reflectance is given. Several contrived simulation methods for solving it exist, such as Finite Elements, Monte-Carlo Path Tracing and Photon Mapping. The high-quality results these achieve come at the cost of significant computational effort. Interactive performance is only possible through advanced parallel implementations in specific shader languages [Owens et al. 2007], which not only demands a substantial programming effort, but the proficiency as well. By choosing to leverage Deep Learning architectures, we seek to overcome those computational costs by focusing computation on converting attributes into appearance according to example data rather than using physical principles.

Our approach is based on screen-space shading that has been demonstrated to approximate many visual effects at high performance, such as ambient occlusion [Mittring 2007], indirect light [Ritschel et al. 2009], scattering [Jimenez et al. 2009], participating media [Elek et al. 2013], depth-of-field [Rokita 1993] and motion blur [McGuire et al. 2012]. Anti-aliasing too can be understood as a special form

of screen-space shading, where additional depth information allows to post-blur along the "correct" edge to reduce aliasing in FXAA [Lottes 2011]. All of these approaches proceed by transforming a deferred shading buffer [Saito and Takahashi 1990], i. e., a dense map of pixel-attributes, into RGB appearance. We will further show how a single CNN allows combining all of the effects above at once.

Even though screen-space shading is subject to limitations like missing light and shadow from surfaces not contained in the image, several desirable properties make it an attractive choice for interactive applications such as computer games: computation is focused only on what is visible on screen; no pre-computations are required making it ideal for rich dynamic worlds; it is independent of a geometric representation, allowing to shade range images or iso-surface ray-castings; it fits the massive fine-grained parallelism of current GPUs and many different effects can be computed from the same input representation.

Until now, image synthesis, in particular in screen-space, has considered the problem from a pure simulation point of view. In this paper, we demonstrate competitive results achieved by learning from data, mitigating the need for mathematical derivations from first principles. This has the benefit of avoiding any effort that comes with designing a mathematical simulation model. All that is required is one general but slow simulation system, such as Monte Carlo, to produce exemplars. Also, it adapts to the statistics of the visual corpus of our world which might not be congruent to the one a shader programmer assumes.

Applications of machine learning to image synthesis are limited, with a few notable exceptions. A general overview of how computer graphics could benefit from machine learning, combined with a tutorial from a CG perspective, is given by Hertzmann [2003]. The CG2Real system [Johnson et al. 2011] starts from simulated images that are then augmented by patches of natural images. It achieves images that are locally very close to real world example data, but it is founded in a simulation system, sharing all its limitations and design effort. Recently, CNNs were used to transfer artistic style from a corpus of example images to any new exemplar [Gatys et al. 2015]. Our work is different as shading needs to be produced in real-time and in response to a great number of guide signals encoding the scene features instead of just locally changing RGB structures when given other RGB structures. Dachsbacher [2011] has used neural networks to reason about occluder configurations. Neural networks have also been used as a basis of pre-computed radiance transfer [Ren et al. 2013] (PRT) by running them on existing features to fit a function valid for a single scene. They share the limitations of PRT, such as static geometry and limited spatial resolution, which have prevented its wide-spread use in the industry, that routinely uses screen-space shading. Earlier, neural networks were used to learn a mapping from character poses to visibility for PRT [Nowrouzezahrai et al. 2009]. Without end-to-end learning of convolutions and a deep architecture, all approaches mentioned do not achieve a generalization between scenes, but remain limited to a specific room, character, etc. Kalantari et al. [2015] have used example data to learn optimal parameters for filtering Monte Carlo noise. Our approach also learns from labeled reference images, but learns shading itself instead of filtering the noise produced by a different shading method.

For image processing, Convolution Pyramids [Farbman et al. 2011] have pursued an approach that optimizes over the space of filters to the end of fast and large convolutions. Our approach optimizes over pyramidal filters as well, but allows for a much larger number of internal states and much more complex filters defined on much richer input. Similar to Convolutional Pyramids, our network is based on a pyramidal CNN, allowing for fast but large filters to produce long-range effects such as distant shadows or strong depth-of-field.

Appearance-to-attributes The inverse problem of turning image appearance into semantic and non-semantic attributes lies at the heart of computer vision. Of late, Deep Networks, particularly CNNs, have shown unprecedented advances in typical inverse problems such as detection [Krizhevsky et al. 2012], segmentation and detection [Girshick et al. 2014], or depth [Eigen et al. 2014], normal [Wang et al. 2015] or reflectance estimation [Narihira et al. 2015]. These advances are underpinned by three developments: availability of large training datasets, deep but trainable (convolutional) learning architectures, and GPU accelerated computation. Another key contributor to these advances has been the ability to train end-to-end, i. e., going from input to desired output without having to devise intermediate representations and special processing steps.

One recent advance is of importance in applying CNNs to high-quality shading: The ability to produce dense per-pixel output even for high resolutions. Recently, (de-convolutional) CNNs that do not only decrease, but also increase resolution were proposed [Long et al. 2015; Hariharan et al. 2015], resulting in fine per-pixel solutions.

For the problem of segmentation, Ronneberger et al. [2015] even apply a fully symmetric U-shaped architecture where each down-sampling step is matched by a corresponding up-sampling step which is re-using earlier intermediate results of the same resolution level.

CNNs have also been employed to replace certain graphics pipeline operations such as changing the viewpoint [Dosovitskiy et al. 2015; Kulkarni et al. 2015]. Here, appearance would be already known, it is just required to manipulate it to achieve a novel view. In our work, we do not seek to change a rendered image but to create full high-quality shading from the basic output of a GPU pipeline such as geometry transformation, visible surface determination, culling, direct light, and shadows.

We seek to circumvent the need to manually concoct and combine convolutions into screen-space shaders that have to be programmed, and ultimately benefit from the tremendous advances in optimizing over deep convolutional networks to achieve a single screen-space über-shader that is optimal in the sense of certain training data.

3 Background

Here we briefly summarize some aspects of Machine Learning, Neural Networks, Deep Learning and training of Convolutional Networks, only to the extent necessary for immediate application to the computer graphics problem of shading.

For our purposes, it suffices to view (supervised) learning as simply fitting a sufficiently complex and high-dimensional function $\tilde{f}$ to data samples generated from an underlying, unknown function f, without letting the peculiarities of the sampling process from being expressed in the fit. In our case, the domain of f consists of all instances of a per-pixel deferred shading buffer for images of a given resolution (containing per-pixel attributes such as position, normals and material parameters) and the output is the per-pixel RGB image appearance of the same spatial resolution. We are given the value $f(\mathbf{x}_i)$ of the function applied to $\mathbf{x}_i$, the ith of n example inputs. From this we would like to find a good approximation $\tilde{f}$ to f, with the quality of the fit quantified by a cost/loss function that defines some measure of difference between $\tilde{f}(\mathbf{x}_i)$ and $f(\mathbf{x}_i)$. Training examples can be produced in arbitrary quantity, by mere path tracing or any other sufficiently powerful image synthesis algorithm.

Neural Networks (NNs) are a particularly useful way of defining arbitrary non-linear approximations $\tilde{f}$. A Neural Network is typically comprised of computational *units* or *neurons*, each with a set of inputs and a singular scalar output that is a non-linear function of some affine combination of its inputs governed by a vector of

weights $\mathbf{w}_k$ for each unit k. This affine combination per *unit* is what is learned during training. The *units* are arranged in a hierarchical fashion in *layers*, with the outputs from one *layer* serving as the inputs to the *layers* later in the hierarchy. There are no connections between *units* of the same layer. The *fan-in* of each unit can either connect to all outputs of the previous layer (*fully-connected*), or only *sparsely* to a few, typically nearby ones. Furthermore, units can also be connected to several preceding layers in the hierarchy.

The non-linearity applied to the affine combination per unit is called the *activation function*. These are often smooth functions, such as the Sigmoid. In our networks, we make use of *Rectified Linear Units* (ReLUs) that simply clamp each unit's output to the non-negative range.

Defining $\mathbf{w}$ as the set of weights for the entire network, the function $\tilde{f}(\mathbf{x}_i)$ can be expressed as $\tilde{f}_\mathbf{w}(\mathbf{x}_i)$. A typical choice of *loss* is the squared $\mathscr{L}_2$-norm: $||\tilde{f}_\mathbf{w}(\mathbf{x}_i) - f(\mathbf{x}_i)||_2^2$. Alternatively, a perceptual loss function based on a combination of $\mathscr{L}_1$-norm and structural similarity (SSIM) index may be used [Zhao et al. 2015]. Optimizing weights with respect to the loss is a non-linear optimization process, and *Stochastic Gradient Descent* or its variants are the usual choice of learning algorithm. The method makes a computational time - run time trade-off between computing loss gradients with respect to weights at all exemplars at each gradient descent step and computing gradients with one sample at a particular gradient descent step, by choosing to compute it for subsets of exemplars in mini-batches. The gradient with respect to $\mathbf{w}$ is computed by means of *back-propagation*, i. e., the error is first computed at the output layer and then propagated backwards through the network. From this, the corresponding update to each unit's weight can be computed.

Convolutional Networks are a special case of Neural Networks, with a semblance of regular spatial arrangement of the units within layers. Within each layer, units are arranged in multiple regular-grid slices of the same size. Each unit in layer $i+1$ connects to the outputs of the units from all slices of the layer i lying within a certain local spatial extent defined as the (spatial) kernel size of the unit, centered at the unit. The units within each slice share their weights, with the consequence that the operation of each slice can be seen as a 3D convolution with a kernel that is as large as the spatial fan-in of the unit along two dimensions, and extends as large as the number of slices in the previous layer along the third dimension. We will interchangeably use *spatial kernel size* and *kernel size*, and the third kernel dimension is implicit.

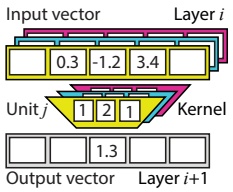

Figure 2: *Terminology.*

CNNs typically stack multiple such convolutional layers, with spatial resolution being reduced between consecutive layers as a trick to achieve translation invariance. However *de-convolutional* networks, allow us to increase the resolution back again [Long et al. 2015], which is critical for our task, where per-pixel appearance i. e., high-quality shading needs to be produced quickly.

4 Deep Shading

Here, we detail the training data we produced for our task, the network architecture proposed and the process of training it.

4.1 Training data

Our training data starts out with about 50,000 pairs of deferred shading buffers and corresponding shaded reference images. Images and buffers are computed in a resolution of $256 \times 256\,px$ using perspective projection from random camera positions within a cube covering the 3D scene, with a fixed field-of-view of 35°. Several scenes of different nature (Fig. 1 left) are sampled to avoid over-fitting to a particular scene. To increase the robustness of the training set in an easy way, only 8,000 unique views of the scene are rendered. We call these *base images*. For each base image three rotated (in steps of 90 deg) as well as horizontally and vertically flipped versions are used. Special care has to be taken when transforming attributes which are with respect to the view space, here the respective positions and vectors have to be transformed themselves by applying rotations or mirroring.

We composed the training scenes with objects from publicly available sources with consideration for the effects to be learnt. For instance, to learn image-based lighting, where the effect depends on the surface normal at each pixel, the scene should contain smooth objects so that as many different normals as possible are covered. Contrarily, to learn an anti-aliasing filter, a lot of fine geometry with sharp edges is necessary to create sufficiently many exemplars for the network. The left of Fig. 1, shows examples of ground truth images for each instance of the network. Sec. 5 contains additional information on the training sets for each application.

Attributes The deferred shading buffers are computed using OpenGL's rasterization without anti-aliasing of any form. They contain per-pixel geometry, material and lighting information. All labels are scaled and biased to fit into 8 bit images.

Positions are only stored with respect to the camera space (P_s) while normals might be stored with respect to both, camera space and world space (N_s and N_w). Normals are represented as unit vectors in Cartesian coordinates and positions are first scaled uniformly by dividing them by the diameter of a bounding sphere around the scene to ensure that they are in the range $[-1,1]^3$. Additionally, depth alone ($D_\mathrm{s} = P_{\mathrm{s},3}$) and distance to the focal plane (D_focal) are provided to also capture sensor parameters. To be able to compute view-dependent effects, the normalized direction to the camera ($C_{\mathrm{w/s}}$) is an additional input.

Material parameters (R) combine surface and scattering properties. For surfaces, we use the set of parameters to the Phong [1975] reflection model, i. e., RGB diffuse and specular colors (denoted as R_diff and R_spec) as well as scalar glossiness (R_gloss). For scattering we use a simplified model with four components (R_scatt): The first three are the screen-space variance that the best single Gaussian approximating the BSSRDF at each RGB channel and at unit distance would have and the fourth is a general scalar strength parameter, controlling how much SSS contributes to the material over all.

Direct lighting (denoted by L or L_diff for diffuse-only) is not computed by the network but provided as an input to it, as is the case with all corresponding screen-space shaders we are aware of. Fortunately, it can be quickly computed at runtime and fed into the network. Direct light is computed using the Phong reflection model with shadow mapping and stored as RGB maps.

Finally, the per-pixel object motion F is encoded as a two-dimensional polar coordinate in each pixel to support motion blur, with the assumption that the motion during exposure time is small enough to be approximated well by a translation. The first component holds the direction between 0 and 2π, the second component holds the distance in that direction.

In summary, each pixel is labeled by a high-dimensional value, where the dimensions are partially redundant and correlated, e. g., normals are derivatives of positions and camera space differs from world space only by a linear transformation. Nonetheless, those

attributes are the output of a typical deferred shading pass in a common interactive graphics application, produced within milliseconds from complex geometric models. Redundant attributes are almost free but improve the performance of certain networks for certain effects. At the same time, for some effects that do not need certain labels, they can be manually removed to increase speed.

Appearance The reference images store per-pixel RGB appearance resulting from shading. They are produced using a reference rendering method. Path traced images, paintings or even real photos would represent valid sample data as well, but their acquisition is significantly more time-consuming than that of the approximate references we use, of which massive amounts can be produced in a reasonable time. Therefore, appearance is computed independent for every pixel as follows. Every pixel iterates all neighboring pixels and computes shading using the attributes. For motion blur and depth of field, 100 individual images from random time and lens samples are computed and averaged to arrive at a reference image combining complex shading and distribution effects. For anti-aliasing, reference images are computed with $10\times$ super-sampling relative to the label images. We do not apply any gamma or tone mapping to our reference images used in training. It therefore has to be applied as a post-process after executing the network.

Typically, screen-space shading is faded out based on a distance term and only accounts for a limited spatial neighborhood. As we train on one resolution but later apply the network also to different ones, the effective size of the neighborhood changes. As a solution, when applying the network at a resolution which is larger by factor of N compared to the training resolution, we also divide the screen space effect radius of the reference image by N. This order is chosen to make the CNN more independent of the final output resolution.

In practice, some effects like Ambient Occlusion (AO) and Direct Occlusion (DO) do not compute final appearance in terms of RGB radiance, but rather a quantity which is later multiplied with albedo. We found the networks that do not emulate this obvious multiplication to be substantially more efficient while also requiring less input data and therefore opt for a manual multiplication. However, some networks that go beyond this simple case need to include the albedo in their input and calculations. The result section will get back to where albedo is used in detail. Tbl. 1 provides an overview in the column "albedo".

In a similar vein, we have found that some effects are best trained for a single color channel, while others need to be trained for all channels at the same time. In the first case, the same network is executed for all three input channels simultaneously using vector arithmetic after training it on scalar images showing only one of the color channels. In the second case, one network with different weights for the three channels is run. We refer to the first case as "mono" networks, to the latter as "RGB" networks (Tbl. 1).

4.2 Network

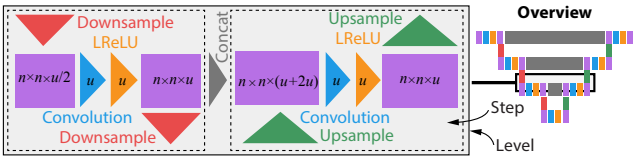

Figure 3: Left: *One level of our network.* Right: *The big picture.*

Our network is U-shaped, with a left and a right *branch*. The first and left branch is reducing spatial resolution (*down* branch) and the second and right branch is increasing it again (*up* branch). We refer to the layers producing outputs of one resolution as a *level*. Fig. 3 shows an example of one such level. Overall, up to 5 levels with corresponding resolutions ranging from 256×256 px to 16×16 px are used. Further, we refer to all layers of a particular level and branch (i. e., left or right) as a *step*. Every step is comprised of a convolutional as well as a subsequent activation layer. The convolutions (blue in Fig. 3) have a fixed extent, which is the same for all convolutions, in the spatial domain, which varies depending on the effect to compute. The consecutive activation layers (orange in Fig. 3) consist of *leaky ReLUs* as described by Maas et al. [2013], which multiply negative values by a small constant instead of zero.

The change in resolution between two steps on different levels is performed by re-sampling layers. These are realized by 2×2 mean-pooling on the down (red in Fig. 3) and by bilinear up-sampling (green in Fig. 3) on the up branch.

The layout of this network is the same for all our effects, but the number of kernels on each level and the number of levels vary. All designs have in common that the number of kernels increases by a factor of two on the down part to decrease by the same factor again on the up part. We denote the number of kernels used on the first level by u_0. A typical start value is $u_0 = 16$, resulting in a 256-dimensional feature vector for every pixel in the coarsest resolution. The coarsest level consists of only one step, i. e., one convolution and one activation layer, as depicted in Fig. 3. Additionally, the convolution steps in the up-branch access the outputs of the corresponding step of the same output resolution in the down part (gray arrow in Fig. 3). This allows to retain fine spatial details.

A typical network has about three million degrees-of-freedom i. e., weights and bias terms which are learned (see Tbl. 1 for details). We call the CNN resulting from training on a specific input and specific labels a *Deep Shader*.

Training Caffe [Jia et al. 2014], an open-source CNN implementation, is used to implement and train our network. To produce the input cube to the first step, all input attributes are loaded from image files and their channels are concatenated forming input vectors with 3 to 18 components per pixel. To allow networks of varying complexity which demand different learning rates in order to keep optimization using stochastic gradient descent fast and stable, an adaptive learning rate method (ADADELTA [Zeiler 2012]) is used.

We use a loss function based on the structural similarity (SSIM) index [Zhao et al. 2015] which compares two image patches in a perceptually motivated way. The loss between the output of the network and the ground truth is computed by first tiling the two images into patches of 8×8 px and computing the SSIM between corresponding patches for each channel. SSIM ranges from -1 to 1, where a higher value means a higher similarity. It is therefore subtracted from 1 and halved to compute the structural dissimilarity (DSSIM). The sum of the DSSIM values for all patch pairs across all channels defines the final loss.

Testing To evaluate the performance of the network, we compute the test error as the average loss over a set of 100 test images. These are produced in the same way as the training data, randomly sampling viewpoints and attributes like diffuse reflectance. In addition to the common requirement that the testing set should not be part of the training data, we also only use scenes not used to produce training data when preparing the test data. The resulting SSIM values are listed in Tbl. 1.

Implementation While Caffe is useful for training the network, it is inconvenient for executing it in the setting of an interactive

rendering application. We therefore re-implemented the forward pass of the network using plain OpenGL shaders operating on array textures. In our application, the result of applying the Deep Shader can be interactively explored as seen in the supplemental video.

5 Results

5.1 Effects

Here we will analyze our learned Deep Shaders for different shading effects. Tbl. 1 provides an overview over their input attributes, structural properties and the resulting SSIM achieved on a test set, together with the time necessary to execute the network using our own OpenGL implementation on an NVIDIA GeForce GTX 980 Ti GPU. For visual comparison, we show examples of Deep Shaders applied to new (non-training) scenes compared to the reference implementations used to produce the training sets in Fig. 4.

Table 1: *Structural properties of the networks for different effects, resulting degrees of freedom, SSIM on the test set and time for executing the network using our OpenGL implementation on* 512×384 *px input as seen in Fig. 4. In case of mono networks, the time refers to the simultaneous execution of three networks. The SSIM is always with respect to the raw output of the network, e. g., indirect irradiance for GI. The final image might show even better SSIM.*

Effect	Attributes	Albedo	Mono	u_0.	Lev.	Ker.	Size	SSIM	Time
IBL	N_w, C_w, R	✓	✗	128	1	1×1	2 K	.982	7.2 ms
AO	N_s, P_s	✗	✓	8	5	3×3	244 K	.891	26 ms
DO	N_w, N_s, P_s	✗	✗	16	5	3×3	1.0 M	.694	78 ms
GI	N_s, P_s, L_{diff}	✓	✓	16	5	5×5	2.7 M	.648	254 ms
SSS	P_s, R_{scatt}, L	✓	✓	8	5	3×3	244 K	.977	35 ms
DoF	D_{focal}, L	✓	✓	8	5	5×5	171 K	.926	28 ms
MB	F, L, D_s	✓	✓	8	5	3×3	244 K	.973	36 ms
AA	D_s, L	✓	✓	8	1	5×5	609	.882	2.2 ms
Full	All	✓	✗	16	5	5×5	2.7 M	.527	209 ms

Ambient Occlusion Ambient occlusion (AO) simulates darkening in corners and creases due to a high number of blocked light paths and is typically defined as the percentage of directions in the hemisphere around the surface normal at a point which are not blocked within a certain distance. Our ground truth images are computed from screen-space positions and normals using screen-space ambient occlusion (SSAO) [Mittring 2007] with a constant effect range defined in world space units. In an actual application, the AO term is typically multiplied with the ambient lighting term before adding it to the image.

The trained network faithfully reproduces darkening in areas with nearby geometry (Fig. 4). The most noticeable difference to images produced using a reference is a lack of contrast. This might be due to the training set not containing enough examples of empty spaces, which should be rendered purely white.

We made AO – which is a prototypic screen-space effect – the subject of further in-depth analysis of alternative network designs described in Sec. 5.2 and seen in Fig. 6, a) and b).

Image-based Lighting In image-based lighting (IBL) a scene is shaded by sampling directions in an environment map to determine incoming radiance, assuming all directions are unblocked. The network is trained to render a final image, so that no post-multiplications are necessary, using diffuse and specular colors as well as gloss strengths. It can operate on all color channels simultaneously.

As can be seen from the vehicles in Fig. 4, the network handles different material colors and levels of glossiness well. The two main limitations are a slight color shift compared to a reference, as seen in the tires of the tractors, and an upper bound on the representable level of glossiness. The latter is not surprising as the extreme here is a perfect mirror which would need a complete encoding of the environment map used in training, which has a resolution of several megapixels, into as few as 128 convolution kernels.

Directional Occlusion Directional occlusion (DO) is a generalization of ambient occlusion where each sample direction is associated with a radiance sample taken from an environment map and we sum up light only from unblocked directions [Ritschel et al. 2009]. A DO Deep Shader is specific to the environment map it was trained on and operates on all channels simultaneously, like IBL. The DO result is applied in the same way as AO.

While AO works well, Deep Shading struggles more with DO. The increased difficulties come from indirect shadows now having different colors and appearing only for certain occlusion directions. As can be seen in Fig. 4, the color of the light from the environment map and the color of shadows match the reference but occlusion is weakened in several places. This is due to the fact that the indirect shadows resulting from DO induce much higher frequencies than unshadowed illumination or the indirect shadows in AO which assume a constant white illumination from all directions, which are harder to encode in a network.

Diffuse Indirect Light A common challenge in rasterization-based real-time rendering is how to add indirect light. To simplify the problem, the set of relevant light paths is often reduced to a single "indirect bounce" and diffuse reflection [Tabellion and Lamorlette 2004]. For this specialization, screen-space global illumination (SSGI) is a possible approximation. The ground truth in our case consists of the "indirect radiance", i. e., the light arriving at each pixel after one interaction with a surface in the scene. From this, the final indirect component can be computed by multiplying with the diffuse color. We compute our ground truth images in screen space as well [Ritschel et al. 2009]. As we are assuming diffuse reflections, the direct light input to the network is computed using only the diffuse reflectance of the material. In the absence of advanced effects like fluorescence or dispersion, the light transport in different color channels is independent from each other. We therefore apply a monochromatic network. The network successfully learns to brighten areas in shadow, which do not appear pitch-black anymore, rather the color of nearby lit objects (Fig. 4).

Anti-aliasing While aliasing on textures can be reduced by applying proper pre-filtering, this is not possible for sharp features produced by the geometry of a scene itself. Classic approaches compute several samples of radiance per pixel which typically comes with a linear increase in computation time. This is why state-of-the-art applications like computer games offer simple post-processing filters like Fast Approximate Anti-Aliasing (FXAA) [Lottes 2011] as an alternative, which operate on the original image and auxiliary information such as depth values. We let our network learn such a filter on its own, independently for each channel.

Applying our network to an aliased image (Fig. 4) replaces jagged edges by smooth ones. While it cannot be expected to reach the same performance as the $10\times$ Multi Sample Anti-Aliasing (MSAA) we use for our reference, which can draw from orders of magnitude of additional information, the post-processed image shows fewer disturbing artefacts. At the same time, the network learns to not over-blur interior texture areas that are properly sampled, but only blurs along depth discontinuities.

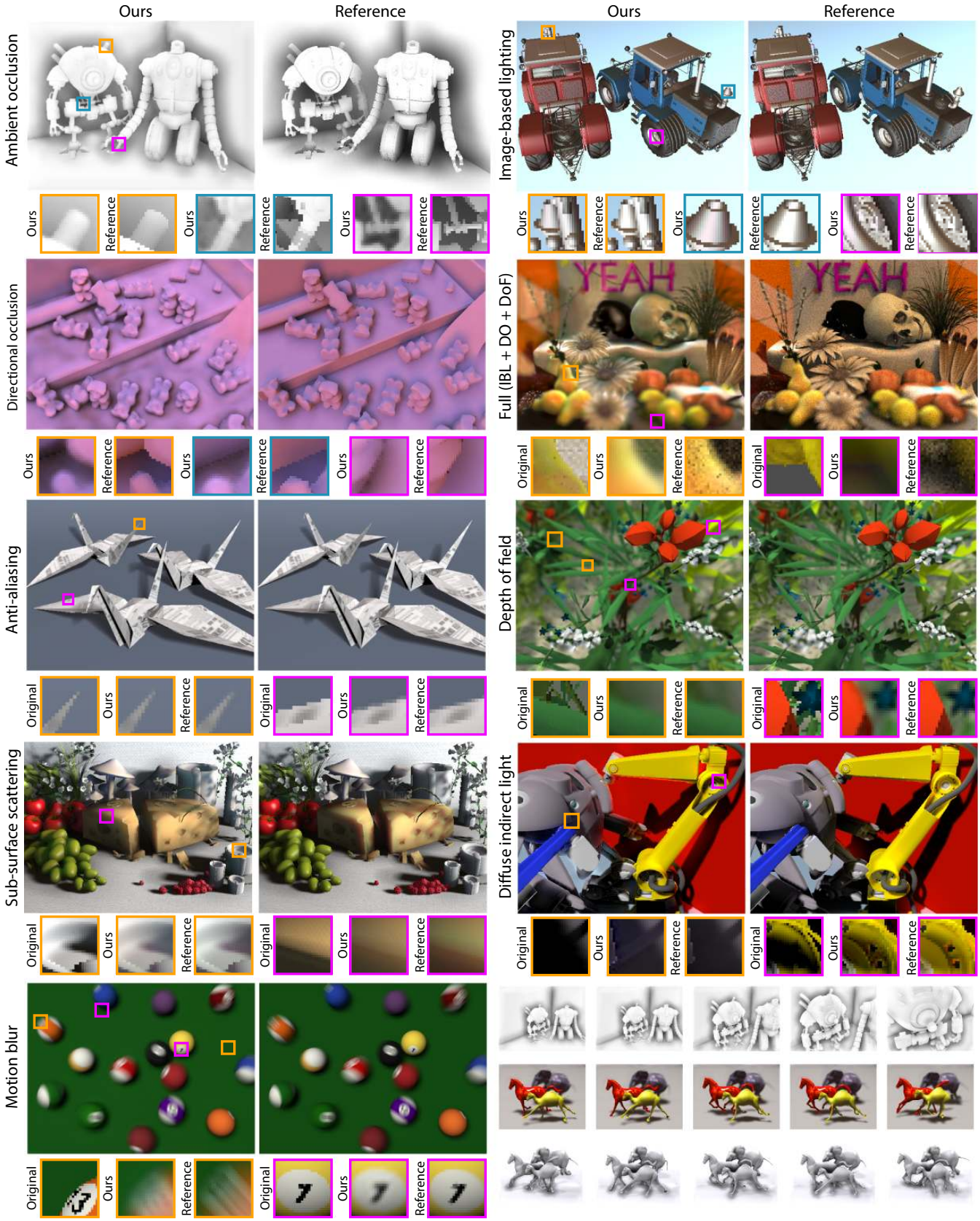

Figure 4: *Results of different Deep Shaders as discussed in Sec. 5.1. "Original" is the input RGB image without any effect. The lower right panel shows animate variants of the Deep Shader seen above from the supplemental video, running at interactive rates.*

Depth-of-field As a simple rasterization pass can only simulate a pinhole camera, the appearance of a shallow depth of field (DoF) has to be faked by post-processing when multiple rendering passes are too costly. In interactive applications, this is typically done by adaptive blurring of the sharp pinhole-camera image. We learn our own depth-of-field blur from sample data which we actually generate in an unbiased way, by averaging renderings from multiple positions on the virtual camera lens. The amount of blurriness depends on the distance of each point to the focal plane. As the computation of the latter does not come with any additional effort compared to the computation of simple depth, we directly use it as an input to the Deep Shader. While the training data is computed using a fixed aperture, the shallowness of the depth of field, as well as the focusing distance, are easy to adjust later on by simply scaling and translating the distance input. The Deep Shader again is trained independently for each channel, assuming a non-dispersive lens.

The Deep DoF Shader blurs things in increasing distance from the focal plane by increasing extents. In Fig. 4, the red blossoms appear sharper than e. g., the blades of grass in the background. It turned out to be fruitful to use textured objects in training to achieve a sufficient level of sharpness in the in-focus areas.

The biggest challenge for Deep Shading of all distribution effects (DoF, MB, AA) is to avoid over-blurring in the "zero areas": at the focal plane for DoF, static objects for MB or off-geometry edges for AA. The network tends to get this difference right in many places and by-large, but fails to fully avoid blurring, resulting in slight haloes or blur that reduces fine spatial details.

Sub-surface Scattering Simulating the scattering of light inside an object is crucial for achieving realistic appearance for translucent materials like wax or skin. A popular approximation to this is screen-space sub-surface scattering (SSSS) [Jimenez et al. 2009] which essentially applies a spatially-varying blurring kernel to the different color channels of the image.

While the original method uses a sum of Gaussian kernels to approximate a physically-motivated diffusion profile derived from scattering and absorption coefficients, we opted for a single Gaussian with a different variance for each color channel to keep the number of input attributes low. An automatic translation of physical parameters into the right diffusion profile represents further work.

To compute the final images on which the network is trained, we linearly interpolate between the blurred image and the original one depending on a per-object scalar which is supplied to the network as well. In a physically correct solution, this would be handled by applying the Fresnel equations depending on the refractive indices of the materials, which determine which portion of the light is transmitted into the material and which is directly reflected.

After training the Deep Shader independently for all RGB channels on randomly textured training images with random variances for the applied Gaussian blurring kernel and with random scattering strengths, we achieve images which are almost indistinguishable from our reference method.

Motion Blur Motion blur is the analog to depth-of-field in the temporal domain. Images of objects moving with respect to the camera appear to be blurred along the motion vectors of the objects for non-infinitesimal exposure times. The direction and strength of the blur depends on the speed of the object in the image plane [McGuire et al. 2012].

For training, we randomly moved objects inside the scene for random distances. Motions are restricted to those which are parallel to the image plane, so that the motion can be encoded by an angle and magnitude alone. We also provide the Deep Shader with a depth image to allow it to account for occlusion relations between different objects correctly, if possible. Our Deep Shader performs motion blur in an convincing way that manages to convey a sense of movement and comes close to a reference image (Fig. 4).

Full Shading Finally, we learn a Deep Shader that combines several shading effects at once and computes a scene shaded using image-based lighting, with directional occlusion to produce soft shadows, and additional shallow depth-of-field and anti-aliasing. As DO and IBL are part of the effect, the network can again make use of all channels simultaneously.

An image generated using the network (Fig. 4) exhibits all of the effects present in the training data. The scene is shaded according to the environment map, working for both diffuse (cloth) and moderately glossy (fruit) materials. Furthermore, occlusion is visible, e. g., around the skull, and the background and very front show a subtle depth-of-field effect.

Animations Please see the supplemental video for view changes inside those scenes, and dynamic characters. The network might overblur at times, but we found Deep Shading to almost never produce any flickering as the network is built from smooth functions.

Typical artefacts In networks, where light transport becomes too complex and the mapping was not fully captured, what looks plausible in a static image may start to look wrong in a way that is hard to compare to common errors in computer graphics: spatio-temporal patterns resembling the correct patterns manifest, but are inconsistent with the laws of optics and with each other, adding a painterly and surrealistic touch. We show example artefacts in Fig. 5. Capturing high frequencies is a key challenge of Deep Shaders (Fig. 5, a). If the network does not have enough capacity or was not train enough, the results might overblur with respect to the reference. We consider this a graceful degradation compared to typical artefacts of man-made shaders such as ringing or Monte Carlo noise, which are unstable over time and unnatural with respect to natural image statistics. Sometimes, networks trained on RGB tend to produce color shifts (Fig. 5, b). CNN-learned filters may also introduce high frequencies manifesting as ringing (Fig. 5, c). At image boundaries, Deep Shaders may behave differently and produce incorrect patterns (Fig. 5, d). At attribute discontinuities, the SSIM loss lacking an inter-channel prior gives rise to color ringing (Fig. 5, e).

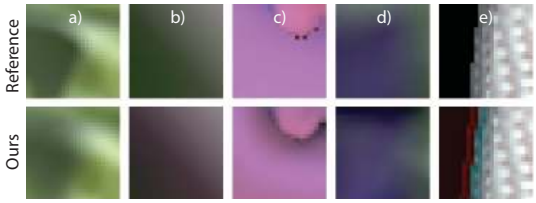

Figure 5: *Typical artefacts of our approach:* a): *Blur.* b): *Color shift.* c): *Ringing.* d): *Image boundaries.* e): *Attribute discontinuities.*

5.2 Analysis

Deep Learning architectures, with their vast number of trainable parameters, have a propensity to overfit even when presented with a large corpus to learn from. Of concern to us is the trade-off between the expressiveness of the network in approximating a certain effect and its computational demands. To understand this, we investigate two modes of variation of the number of parameters of the network,

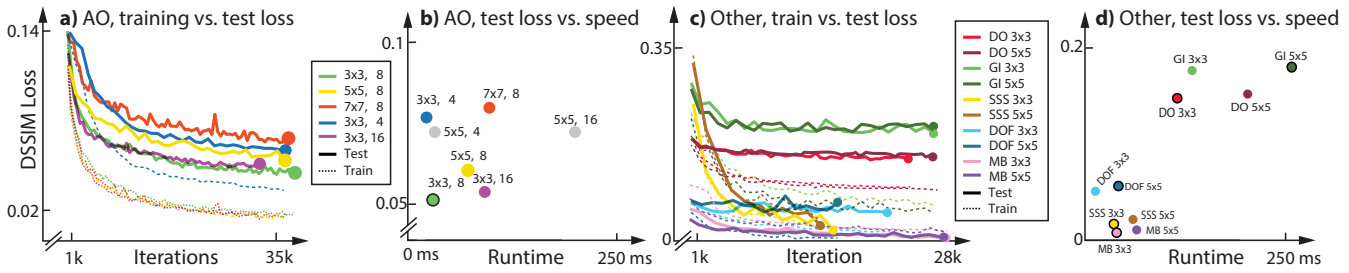

Figure 6: *Analysis of different network structures. We here compare different design choices for different effects in terms of compute time and DSSIM loss. The vertical axes on all plots corresponds to DSSIM loss (less is better). The horizontal axes of the line plots range over the number of learning iterations. The scatter plots have computation time of the Deep Shader as the horizontal axis.* a) *Loss as a function of iterations for different designs of AO (curves).* b) *Relation of final loss and compute time for different designs for AO.* d) *Loss as a function of iterations for different designs for other effects (curves).* d) *Relation of final loss and compute time for different designs for other effects.*

choosing to vary the spatial extent of the kernels as well as the number of kernels on the first level u_0. Our objective for each effect being finding the smallest network with sufficient capacity to learn, that generalizes well on previously unseen data. The results are summarized in Fig. 6 for the example of AO.

Spatial Kernel Size Fig. 6, (a) (green, yellow and orange lines) shows the evolution of training and test error with an increasing number of training iterations, with the number of kernels fixed to a medium value of $u_0 = 8$ and varying the spatial extent of the kernels. We see that all three configurations have a nearly identical training error profile, which means that they all possess sufficient capacity to approximate the mapping induced by the training set equally well. However, the test error actually becomes worse with increasing spatial kernel size, indicating that the networks with kernel sizes of 5×5 and 7×7 are overfitting and hence not generalizing well. While an increased kernel size can increase the computational power of the network, it also increases the need for an extensive training set. Thus, for the given training set, 3×3 is the optimal choice and it also represents the fastest-to-execute of the three options as shown in Fig. 6, (b).

Initial Number of Kernels The orthogonal mode of variation is u_0, the number of kernels on the first level, with the number of kernels in subsequent layers expressed as multiples of u_0. Again, we plot the training and test error, this time for different values of u_0 (Fig. 6, a, blue and purple lines). On the training set, $u_0 = 16$ performs only slightly better than $u_0 = 8$, while both perform equally well on the test set. With 16 kernels, the network is slightly overfitting to the training set. Additionally, increasing the number of kernels contributes to increased memory consumption, both in the way of increased number of parameters and increased size of intermediate representations. Varying the spatial kernel size in isolation does not affect the size of intermediate representations. Reducing the number to $u_0 = 4$ however evinces a clear loss of expressiveness. We therefore choose $u_0 = 8$ for our application, as a doubling of u_0 to 16 also comes with a huge hit in performance (Fig. 6, b).

Structural Choices for other Effects The detailed analysis for AO yields an expedient direction to proceed in for the choice of kernel size and u_0 for the other effects. We start off with spatial extents of 3×3 and 5×5, with $u_0 = 8$, and proceed to increase or decrease u_0 in accordance with overfit / underfit characteristics exhibited by the the train-test error curves. Tbl. 1 indicates the final choice of the network structure for each effect. Additionally, the train-test error curves for the top two contenders for each effect are shown in Fig. 6, (c), with their test loss-vs.-speed characteristics captured in Fig. 6, (d). u_0 for all pairs of curves in Fig. 6, (c) are as listed in Tbl. 1.

From Fig. 6, (d) (red and dark red), the choice of kernel size for DO is clearly 3×3. For GI (green and dark green) however, even though the losses are quite similar, we go with the larger kernel 5×5 because it produced visually better results. In case of DOF (blue and dark blue), it again is a close call both in terms of execution time and loss, so we pick the larger kernel size. For SSS (yellow and brown) as well as MB (pink and purple) it is a close call and we go with 3×3 for both.

6 Conclusion

We have proposed Deep Shading, a system to perform shading using a CNN. Different from previous applications in computer vision using appearance to infer attributes, Deep Shading leverages deep learning to turn attributes of virtual 3D scenes into appearance. It is also the first example of performing complex shading purely by learning from data and removing all considerations of light transport simulation derived from first principles of optics.

We have shown that CNNs can actually model any screen-space shading effect such as ambient occlusion, indirect light, scattering, depth-of-field, motion blur, or anti-aliasing as well as arbitrary combinations of them at competitive quality and speed. Our main result is a proof-of-concept of image synthesis that is not programmed by human experts but learned from data without human intervention.

The main limitation of Deep Shading is the one inherent to all screen-space shading, namely missing shading from objects not contained in the image due to occlusion, clipping or culling. At the same time, screen-space shading is well-established in the industry due to its ability to handle large and dynamic scenes in an output-sensitive manner. We would also hope, that in future refinements, the Deep Shader might even learn to fill in this information, e. g., it might recognize the front of a sphere and know that in a natural scene the sphere will have a symmetric back that will cast a certain shadow. In future work, we would like to overcome the limitation to screen space effects by working on a different scene representation, such as surfels, patches or directly in the domain of light paths. Some shading effects like directional occlusion and indirect lighting are due to very complex relations between screen space attributes. Consequently, not all configurations are resolved correctly by a network with limited capacity, such as ours which runs at interactive rates. We have however observed that the typical artefacts are much more pleasant than for human-designed shaders. Typical ringing and over-shooting often produces patterns the network has learned from similar configurations, and what appears plausible to the network is

often visually plausible as well. A perceptual study could look into the question whether Deep Shaders, in addition to their capability to learn shading, also produce more visually plausible errors than the typical simulation-type errors which are patterns that never occur in the data. Screen-space excels in handling complex dynamic scenes, and Deep Shading does as well. Deep Shaders that result in a low final test error (Fig. 6, c) are almost free of temporal artefacts as seen in the supplemental video.

Deep Shading of multiple effects can currently achieve performance en-par with human-written code, but not exceed it. We would hope that more and improved training data, advances in learning methods and new types of deep representations will allow surpassing human shader programmer performance in a not-so-distant future.

References

DACHSBACHER, C. 2011. Analyzing visibility configurations. *IEEE Trans. Vis. and Comp. Graph. 17*, 4, 475–86.

DOSOVITSKIY, A., TOBIAS SPRINGENBERG, J., AND BROX, T. 2015. Learning to generate chairs with convolutional neural networks. In *Proc. CVPR*, 1538–1546.

EIGEN, D., PUHRSCH, C., AND FERGUS, R. 2014. Depth map prediction from a single image using a multi-scale deep network. In *Proc. NIPS*, 2366–74.

ELEK, O., RITSCHEL, T., AND SEIDEL, H.-P. 2013. Real-time screen-space scattering in homogeneous environments. *IEEE Computer Graph. and App.*, 3, 53–65.

FARBMAN, Z., FATTAL, R., AND LISCHINSKI, D. 2011. Convolution pyramids. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 6, 175:1–175:8.

GATYS, L. A., ECKER, A. S., AND BETHGE, M. 2015. A neural algorithm of artistic style. *arXiv 1508.06576*.

GIRSHICK, R., DONAHUE, J., DARRELL, T., AND MALIK, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR*, 580–7.

HARIHARAN, B., ARBELÁEZ, P., GIRSHICK, R., AND MALIK, J. 2015. Hypercolumns for object segmentation and fine-grained localization. In *Proc. CVPR*.

HERTZMANN, A. 2003. Machine learning for computer graphics: A manifesto and tutorial. In *Proc. Pacific Graphics*.

JIA, Y., SHELHAMER, E., DONAHUE, J., KARAYEV, S., LONG, J., GIRSHICK, R., GUADARRAMA, S., AND DARRELL, T. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proc. ACM Multimedia*, 675–8.

JIMENEZ, J., SUNDSTEDT, V., AND GUTIERREZ, D. 2009. Screen-space perceptual rendering of human skin. *ACM Trans. Applied Perception 6*, 4, 23.

JOHNSON, M. K., DALE, K., AVIDAN, S., PFISTER, H., FREEMAN, W. T., AND MATUSIK, W. 2011. CG2Real: Improving the realism of computer generated images using a large collection of photographs. *IEE Trans. Vis. and Comp. Graph. 17*, 9, 1273–85.

KAJIYA, J. T. 1986. The rendering equation. In *ACM SIGGRAPH*, vol. 20, 143–50.

KALANTARI, N. K., BAKO, S., AND SEN, P. 2015. A machine learning approach for filtering Monte Carlo noise. *ACM Trans. Graph. (Proc. SIGGRAPH)*.

KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, 1097–105.

KULKARNI, T. D., WHITNEY, W., KOHLI, P., AND TENENBAUM, J. B. 2015. Deep convolutional inverse graphics network. In *Proc. NIPS*.

LONG, J., SHELHAMER, E., AND DARRELL, T. 2015. Fully convolutional networks for semantic segmentation. In *Proc. CVPR*.

LOTTES, T., 2011. FXAA. Nvidia White Paper.

MAAS, A. L., HANNUN, A. Y., AND NG, A. Y. 2013. Rectifier nonlinearities improve neural network acoustic models. *Proc. ICML 30*.

MCGUIRE, M., HENNESSY, P., BUKOWSKI, M., AND OSMAN, B. 2012. A reconstruction filter for plausible motion blur. In *Proc. ACM i3D*, 135–42.

MITTRING, M. 2007. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 Courses*, 97–121.

NARIHIRA, T., MAIRE, M., AND YU, S. X. 2015. Direct intrinsics: Learning albedo-shading decomposition by convolutional regression. In *Proc. CVPR*, 2992–3.

NOWROUZEZAHRAI, D., KALOGERAKIS, E., AND FIUME, E. 2009. Shadowing dynamic scenes with arbitrary BRDFs. In *Comp. Graph. Forum*, vol. 28, 249–58.

OWENS, J. D., LUEBKE, D., GOVINDARAJU, N., HARRIS, M., KRÜGER, J., LEFOHN, A. E., AND PURCELL, T. J. 2007. A survey of general-purpose computation on graphics hardware. In *Comp. Graph. Forum*, vol. 26, 80–113.

PHONG, B. T. 1975. Illumination for computer generated pictures. *Communications of the ACM 18*, 6, 311–317.

REN, P., WANG, J., GONG, M., LIN, S., TONG, X., AND GUO, B. 2013. Global illumination with radiance regression functions. *ACM Trans. Graph. (Proc. SIGGRAPH) 32*, 4, 130.

RITSCHEL, T., GROSCH, T., AND SEIDEL, H.-P. 2009. Approximating dynamic global illumination in image space. In *Proc. ACM i3D*, 75–82.

ROKITA, P. 1993. Fast generation of depth of field effects in computer graphics. *Computers & Graphics 17*, 5, 593–95.

RONNEBERGER, O., FISCHER, P., AND BROX, T. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *Proc. MICAI*. 234–41.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-d shapes. In *ACM SIGGRAPH Computer Graphics*, vol. 24, 197–206.

TABELLION, E., AND LAMORLETTE, A. 2004. An approximate global illumination system for computer generated films. *ACM Trans. Graph. (Proc., SIGGRAPH) 23*, 3, 469–76.

WANG, X., FOUHEY, D. F., AND GUPTA, A. 2015. Designing deep networks for surface normal estimation. *Proc. CVPR*.

ZEILER, M. D. 2012. ADADELTA: an adaptive learning rate method. *CoRR abs/1212.5701*.

ZHAO, H., GALLO, O., FROSIO, I., AND KAUTZ, J. 2015. Is $\updownarrow_2$ a good loss function for neural networks for image processing? *arXiv:1511.08861*.