DOI: 10.1111/cgf.13225

Eurographics Symposium on Rendering 2017
P. Sander and M. Zwicker
(Guest Editors)

Volume 36 (2017), Number 4

Deep Shading: Convolutional Neural Networks for Screen Space Shading

O. Nalbach[1], E. Arabadzhiyska[1,2], D. Mehta[1], H.-P. Seidel[1], and T. Ritschel[3]

[1]Max Planck Institute for Informatics, Germany
[2]MMCI / Saarland University, Germany
[3]University College London, United Kingdom

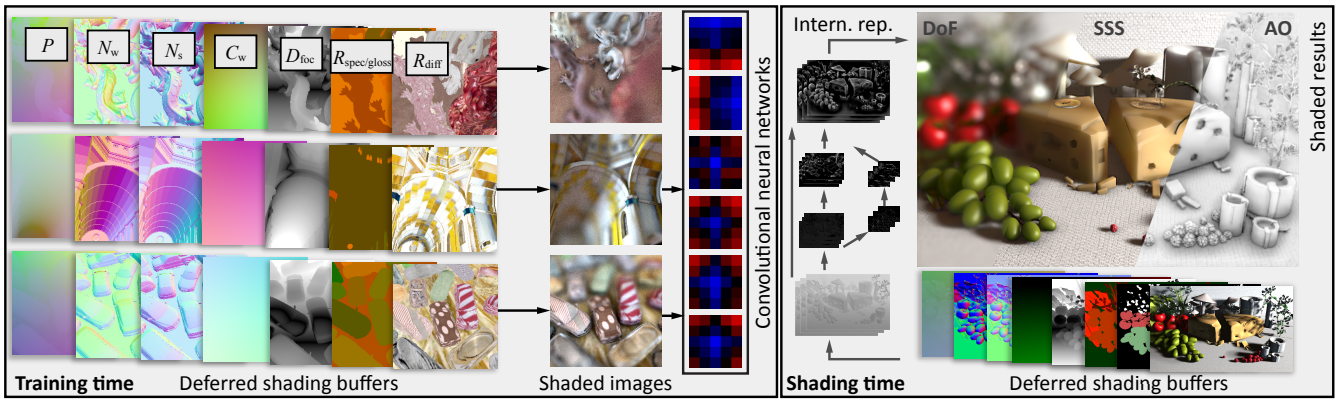

Figure 1: *In training* (left)*, our approach learns a mapping from attributes in deferred shading buffers, e.g., positions, normals, reflectance, to RGB colors using a convolutional neural network (CNN). At run-time* (right)*, the CNN is used to produce effects such as depth-of-field, sub-surface scattering or ambient occlusion at interactive rates (*768×512 *px, 1 ms rasterizing attributes, 27/27/9 ms network execution).*

Abstract

In computer vision, convolutional neural networks (CNNs) achieve unprecedented performance for inverse problems where RGB pixel appearance is mapped to attributes such as positions, normals or reflectance. In computer graphics, screen space shading has boosted the quality of real-time rendering, converting the same kind of attributes of a virtual scene back to appearance, enabling effects like ambient occlusion, indirect light, scattering and many more. In this paper we consider the diagonal problem: synthesizing appearance from given per-pixel attributes using a CNN. The resulting Deep Shading renders screen space effects at competitive quality and speed while not being programmed by human experts but learned from example images.

CCS Concepts

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

1. Introduction

Deep learning achieves unprecedented performance on many computer vision tasks, with several applications revolving around mapping image appearance to attributes such as positions, normals or reflectance. In computer graphics, screen space shading has increased the visual quality in interactive image synthesis, employing per-pixel attributes such as positions, normals or reflectance of a virtual scene to render effects such as ambient occlusion (AO), directional occlusion (DO), indirect light (GI), sub-surface scattering (SSS), depth-of-field (DOF), motion blur (MB), image-based lighting (IBL) or anti-aliasing (AA). In this paper we turn the typical flow of information through computer vision deep learning pipelines around to synthesize appearance from given per-pixel attributes, using deep convolutional architectures. We call this Deep Shading [NAM*16]. Its main benefit is that it can achieve quality and performance similar to human-written shaders by only learning from example data which may be rendered or come from actual photographs. Unlike previous approaches using learning for better filtering of noisy shad-

ing estimates [KBS15], we directly hallucinate the shading without any preceding Monte Carlo sampling. This avoids human effort in shader programming and ultimately allows to for a deep "multi-shader" that combines previously separate screen space effects in one single CNN.

2. Previous Work

Previous work comes from a computer graphics background where attributes are converted into appearance and from a computer vision background where appearance has to be converted into attributes.

Attributes-to-appearance The rendering equation [Kaj86] is a reliable forward model of appearance in the form of radiance reaching a virtual sensor when a 3D scene description is given. Several methods for solving it exist, such as finite elements, Monte Carlo path tracing and photon mapping, but high-quality results come at significant computational cost. Interactive performance is only possible through advanced parallel implementations in specific GPU programming languages [OLG*07], demanding substantial programming effort and proficiency. By deep learning, we seek to overcome these costs by focusing computation on converting attributes into appearance according to sample data rather than using physics.

Our approach is based on screen space shading which can approximate many visual effects at high performance, such as ambient occlusion [Mit07], indirect light [RGS09], sub-surface scattering [JSG09], participating media [ERS13], depth-of-field [Rok93] and motion blur [MHBO12]. Anti-aliasing too can be seen as a special form of screen space shading, where additional depth information allows to post-blur along depth discontinuities to reduce aliasing as in FXAA [Lot11]. All of these transform a deferred shading buffer [ST90], i.e., a dense map of pixel-attributes, into RGB appearance. We show how a single CNN can combine different effects at once.

Although screen space shading has limitations like missing light or shadows from surfaces not part of the image, several properties make it attractive for interactive applications like games: computation is focused only on what is visible on screen; no precomputations are required making it ideal for rich dynamic worlds; it is independent of the geometric representation, allowing to shade range images or ray-casted iso-surfaces; it fits the massive fine-grained parallelism of GPUs and many different effects can be computed from the same input representation.

Until now, image synthesis has considered rendering from a pure simulation point of view. In this paper, we achieve competitive results by learning from data, mitigating the need for mathematical derivations from first principles. This avoids effort that comes with designing a mathematical simulation model. We only require a general but slow simulation system, such as Monte Carlo, to produce exemplars. Also, learning adapts to the statistics of real world renders which might differ from the ones a programmers assume.

Applications of machine learning to image synthesis are limited, with a few notable exceptions. An overview how computer graphics could benefit from learning was given by Hertzmann [Her03]. The CG2Real system [JDA*11] starts from simulated images that are then augmented by patches of real ones. It achieves images that are locally very close to natural images but is founded in a simulation system, sharing its limitations and design effort. Recently, CNNs have been used to transfer artistic style from a corpus of example images to any new exemplar [GEB15]. While style transfer performs local changes of RGB image structures to resemble different RGB structures, we compute shading from diverse physically-based scene features. Also, results need to be produced in real-time. Dachsbacher [Dac11] has used neural networks to reason about occluder configurations. They have also been used as a basis of pre-computed radiance transfer [RWG*13] (PRT) by running them on existing features to fit a function valid for a single scene. In a similar spirit, Ren et al. [RDL*15] have applied machine learning to re-lighting: a network learns how image pixels change color in response to modified lighting. Both works [RWG*13, RDL*15] demonstrate high-quality results when generalizing over light conditions but share the limitation to static 3D scenes, resp. 2D images, without showing generalization to new geometry or animations, such as we do. Such generalization is critical for applications where geometry is dynamic, resulting in a more demanding problem that is worth addressing using advanced (i.e., deep) learning. We would argue that Deep Shading achieves this generalization required to make learning a competitive image-synthesis solution in practice, in the same way that screen space shading is highly adopted by the gaming industry for its inherent support for dynamic scenes.

Earlier, neural networks were used to learn a mapping from character poses to visibility for PRT [NKF09]. Without the end-to-end learning made possible by deeper architectures the approach does not achieve generalization between scenes but remains limited to a specific room, character, etc. Kalantari et al. [KBS15] have used sample data to learn optimal parameters for filtering Monte Carlo Noise. Our input, i.e., screen space attributes, might appear similar but does not include a noisy estimate of the shading to be computed. Also, the range of the machine learning is different. While Kalantari et al. [KBS15] learn optimal parameters for filtering the noisy input using a pre-defined filter, we directly hallucinate the final image. Not much is known about the complexity of the mapping from attributes to filter settings and what the effect of sub-optimal learning is. In our case, the mapping from input attributes to appearance labels is as complex as shading itself. At the same time, the stakes are high: learning this mapping results in an entirely different form of interactive image synthesis, not building on anything such as Monte Carlo ray-tracing that can be slow to compute.

For image processing, convolution pyramids [FFL11] have pursued an approach that optimizes over a space of nested filters to perform fast and large convolutions. We optimize over pyramidal filters as well, but for much more complex filters defined on much richer input. Similar to Cconvolution pyramids, our network is based on a "pyramidal" CNN 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 (non-)semantic attributes lies at the heart of computer vision. Of late, deep neural networks, particularly CNNs, have shown unprecedented advances in classic problems such as detection [KSH12], segmentation [GDDM14], and depth [EPF14], normal [WFG15] or reflectance estimation [NMY15]. These advances were enabled by three developments: availability of large

training datasets, deep but trainable (convolutional) architectures, and GPU acceleration. Another key contributor has been the ability to train end-to-end, i.e., going from input to output without having to devise intermediate representations or processing.

One recent advance would be of importance applying CNNs to high-quality shading: The ability to produce dense output, even for high resolutions, by CNNs that do not only decrease but also increase resolutions as proposed by [LSD15, HAGM15]. For the problem of segmentation, Ronneberger et al. [RFB15] even apply a fully symmetric U-shaped net where each down-sampling step is matched by a corresponding up-sampling step that may also re-use earlier intermediate results of the same resolution level.

CNNs have also been employed to replace certain graphics pipeline operations such as changing the viewpoint [DTSB15, KWKT15]: Appearance is known but manipulated to achieve a novel view. 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 manual programming of efficient screen space shaders and elude the need to come up with analytic approaches by instead learning from examples and optimizing over deep convolutional networks to achieve a single general screen space shader that is optimal in the sense of certain training data.

3. Background

Here we briefly summarize relevant aspects of machine learning to the extent necessary for immediate application to the computer graphics problem of shading. For our purposes, it suffices to view (supervised) learning as fitting a sufficiently complex and high-dimensional function $\tilde{f}$ to data samples generated by an underlying, unknown function f in a robust way. The quality of the fit is quantified by a *loss function*. In our case, the domain of f consists of deferred shading buffers of a given (spatial) resolution which contain per-pixel attributes such as position, normal and material parameters, while the range of f covers per-pixel RGB image appearance of the same resolution. We are given n function values $f(\mathbf{x}_i)$ for n exemplary inputs $\mathbf{x}_i$. These can be produced in arbitrary quantity, e.g., by path tracing or other image synthesis algorithms as well as (depth) sensor imagery.

(Convolutional) Neural Networks Neural networks are a way of defining powerful non-linear approximations $\tilde{f}$. A neural network is comprised of computational *units* or *neurons* mapping multi-dimensional inputs to scalar outputs. These are computed by applying a non-linear function, called *activation function*, to an affine combination of the inputs governed by a vector of *weights* $\mathbf{w}_k$ for each unit k. The weights $\mathbf{w}_k$ are what is learned during training. As activation function, we use *Rectified Linear Units* (ReLUs) which are defined by $r(x) = \max(0, x)$

Multiple units are arranged in a hierarchical fashion and grouped into *layers*, with the outputs of one layer serving as inputs to following layers. There are usually no connections between units of the same layer. The *fan-in*, i.e., the vector of inputs of each unit, can either cover all outputs of the previous layer (*fully-connected*) or only a few. Units may also connect to several preceding layers in the hierarchy. Defining $\mathbf{w}$ as the set of weights for the entire network, the function $\tilde{f}(\mathbf{x}_i)$ can be expressed as $\tilde{f}_{\mathbf{w}}(\mathbf{x}_i)$.

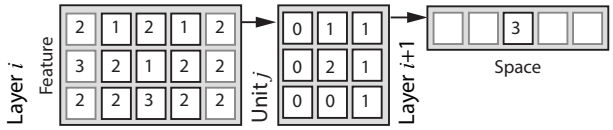

Figure 2: *CNN convolution in the simplified case of one spatial dimension.* Left: *At layer i, the data has a spatial extent of five (horizontal) and three features (vertical).* Middle: *Unit j has kernel width three in the spatial dimension and takes all features present at layer i as input. The fan-in is highlighted by a black outline.* Right: *Unit j produces one feature at one spatial location in layer* $i+1$.

Convolution layers are a particular type of layer defining a regular spatial arrangement of the units: units are arranged into multiple regular and same-sized grid slices. Each unit in the convolution layer $i+1$ connects to the units of all slices of layer i within a certain local spatial extent centered at the respective unit as depicted in Fig. 2. All units of a slice share their weights, so that the computation happening for each slice can be seen as a 3D convolution with a *kernel* which is as "high and wide" as the spatial fan-in of the units and as "deep" as the number of slices in the previous layer. We will refer to the *spatial kernel size* simply as *kernel size*.

Convolutional neural networks stack multiple convolution layers and often also reduce the spatial resolution between consecutive layers to achieve translation invariance and computational efficiency for richer features. CNNs are complemented by de-convolutional (or up-sampling) networks which additionally allow to increase the resolution again [LSD15] which is critical for our task where we want to produce per-pixel appearance in the original resolution.

Training The network's weights are optimized in a non-linear fashion using *stochastic gradient descent* (SGD) so that the network reproduces the training data. The distance between the actual training value and the network output is quantified by the *loss*. A common choice of loss function is the $\mathscr{L}_2$-norm but for networks with images as output, a perceptual loss based on the structural similarity (SSIM) has proven useful [ZGFK17].

4. Deep Shading

Here, we detail the training data we produced for our task, the network architecture proposed and the process of training it. We will use the term *attributes* when referring to the inputs of our networks, i.e., the screen space information such as positions, normals or material parameters. Furthermore, we will refer to the outputs, with which the the inputs are supposed to be labeled by the network, with the general term *appearance*.

4.1. Data Generation

Structure of the Data Our data sets consist of 61,000 pairs of deferred shading buffers and corresponding shaded reference images in a resolution of $512 \times 512\,px$ for AO and $256 \times 256\,px$ for all other

Figure 3: *Selection of images showing training and testing scenes with random textures and lighting.*

effects. Of these 61,000 pairs, we use 54,000 images to train the network, 6,000 for validation and the remaining 1,000 for testing (Sec. 6.2). Train and validation images share the same set of 10 scenes, while the test images come from 4 different scenes not used in training or validation.

To generate the 60,000 train/validation images, we first render 1,000 pairs for each of the set of ten scenes of different nature (Fig. 3, left part). These *base images* are then rotated (in steps of 90°) as well as flipped horizontally and vertically to increase the robustness and size of the training set in an easy way. Special care has to be taken when transforming attributes stored in view space, here the respective positions and vectors have to be transformed themselves by applying rotations or mirroring. For the test set, we proceed analogously but using the distinct set of four scenes and appropriately less base images per scene. Generating one set, i.e., rendering and subsequent data augmentation, takes up to about 170 hours of computation on a single high-end GPU. We plan to make our network definitions and data sets available for use by other research groups.

The base images all show unique and randomly sampled views of the respective scene seen through a perspective camera with a fixed field-of-view of 50°. View positions are sampled from a box fitted to the scenes' spatial extents. Sec. 5 contains additional information on the training sets for each application. Fig. 3 shows samples of typical ground truth images for a combination of image-based lighting and depth-of-field.

About half of our scenes are common scenes from the computer graphics community such as Crytek Sponza or Sibenik Cathedral and other carefully modeled scenes from sources such as BlendSwap. The remaining scenes were composed by ourselves using objects from publicly available sources to cover as many object categories as possible, e.g., vehicles, vegetation or food. Procedurally generated scenes would be another more sophisticated option.

Attributes The deferred shading buffers are computed using OpenGL's rasterization. They contain per-pixel geometry, material and lighting information. All labels are stored as 16 bit float images.

Positions are stored in camera space (P_s) while normals are stored in camera and world space (N_s and N_w). Camera space is chosen as, for our shading purposes, absolute world positions do not contain more information than the former and would encourage the network to memorize geometry. Normals are represented as unit vectors in Cartesian coordinates. Additionally, depth ($D_s = P_{s,3}$), distance to the focal plane (D_{focal}) and a high-level parameter B corresponding to the radius of the circle of confusion of the lens system are provided to capture camera sensor-related parameters. To be able to compute view-dependent effects, the normalized direction to the camera (C_w) is an additional input. Material parameters (R) combine surface and scattering properties. For surfaces, we use the set of parameters to the Phong [Pho75] 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 the model by Christensen and Burley [CB15] which is parameterized by the length of the mean free path for each color channel (R_{scatt}). Direct light (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 run-time and fed into the network. Specifically, we use the Phong reflection model and shadow maps. Finally, to support motion blur, per-pixel object motion F is encoded as a 2D polar coordinate in each pixel, assuming 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 π (motion blur is time-symmetric for time-symmetric shutter functions), the second component holds the distance in that direction.

In summary, each pixel contains a high-dimensional features vector, 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 come at almost no additional cost but improve the performance of 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. They are produced from virtual scenes using rendering. More specifically, we use path tracing for AO, DO and IBL and sample multiple lens positions or points in time for depth-of-field and motion blur, respectively. For anti-aliasing, reference images are computed with 8× super-sampling relative to the label images. We use 64 samples per pixel (spp) to compute the AO training and validation data and 256 samples for the remaining effects. While this means that some Monte Carlo noise remains, compute time is better invested into producing more individual images as demonstrated in Fig. 15. The test sets however are rendered at higher sample counts guaranteeing for noiseless images.

All per-object attributes which are allowed to vary at run-time (e.g., material parameters) are sampled randomly for each training sample. For effects including depth-of-field and sub-surface scattering we found it beneficial to texture objects by randomly as-

signed textures from a large representative texture pool [CMK*14] to increase the information content with respect to the underlying blurring operations. Automatic per-object box mapping is used to assign UV coordinates.

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.

In practice, some effects like AO and DO do not compute final appearance in terms of RGB radiance, but rather a quantity which is later multiplied with albedo. We found 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, the networks for effects 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

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. 4 shows an example of one such level. Overall, up to 6 levels with corresponding resolutions ranging from 512×512 px to 16×16 px are used. Further, we refer to the layers of a particular level and branch (i.e., left or right) as a *step*. Each step is comprised of a convolution and a subsequent activation layer. The convolutions (blue in Fig. 4) have a fixed extent in the spatial domain, which is the same for all convolutions but may vary for different effects to compute. Furthermore, we use convolution groups with 2^n groups on level n. This means that both input and output channels of a convolution layer are grouped into 2^n same-sized blocks where outputs from the m-th block of output channels may only use values from the m-th block of input channels. The consecutive activation layers (orange in Fig. 4) consist of *leaky ReLUs* as described by Maas et al. [MHN13], 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. 4) and by bilinear up-sampling (green in Fig. 4) 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 (i.e., level 0) 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 for the frequent case of 5 levels. The coarsest level consists of only one step, i.e., one convolution and one activation layer, as depicted in Fig. 4. 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. 4). This allows to retain fine spatial details. A typical network has about 130,000 learnable parameters i.e., weights and bias terms (Tbl. 1). We call the CNN resulting from training on a specific input and specific labels a *Deep Shader*.

Training Caffe [JSD*14], an open-source neural network implementation, is used to implement and train our networks. To produce the input 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 facilitate learning of networks of varying complexity, without the need of hyper-parameter optimization, particularly of learning rates, we use an adaptive learning rate method (ADADELTA [Zei12]) with a *momentum* of 0.9 which selects the learning rate autonomously.

We use a loss function based on the structural similarity (*SSIM*) index [ZGFK17] which compares two image patches in a perceptually motivated way, and which we found to work best for our task (Sec. 6.3). The loss between the output of the network and the ground truth is determined by tiling the two images into 8×8 px patches and combining the SSIM values computed between corresponding patches for each channel. SSIM ranges from -1 to 1, higher values indicating higher similarity. Structural dissimilarity (*DSSIM*) is defined as $(1 - SSIM)/2$, and used as the final loss.

Testing The test error is computed as the average loss over our test sets (Sec. 4.1). The resulting SSIM values are listed in Tbl. 1.

Implementation While Caffe is useful for training the network, it is inconvenient for use inside an interactive application. Instead of integrating Caffe into a rendering framework we implement the forward pass of the CNN using OpenGL shaders operating on array textures. OpenGL also enables hardware-supported up- and down-sampling as well as to drop actual concatenation layers by simply accessing two layered inputs instead of one when performing convolutions. In our application, the Deep Shader output can be interactively explored as seen in the supplemental video.

5. Results

This section analyzes learned Deep Shaders for different shading effects. Tbl. 1 provides an overview of their input attributes, structural properties and resulting SSIM achieved on test sets, together with the time needed to execute the network using our implementation on an NVIDIA GeForce GTX 1070 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. 5.

Some shading effects like IBL or DO depend on an input environment map which is accessed globally. As our CNN structure, however, only performs local operations we cannot expect it to handle arbitrary environment maps e.g., given as an additional input

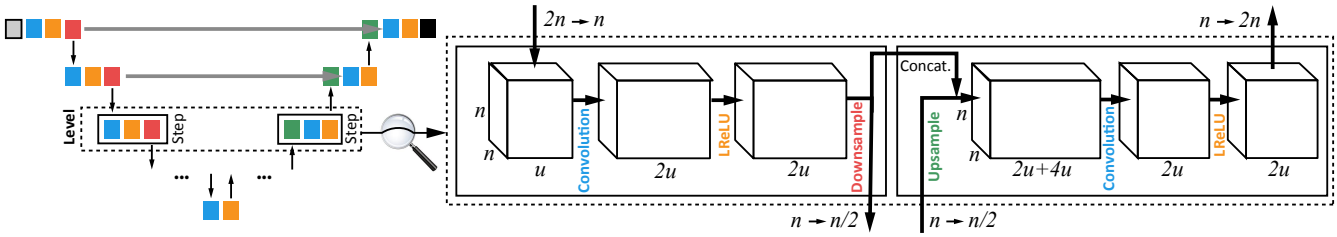

Figure 4: Left: *The big picture with one branch going down and another going up again in a U-shape.* Right:. *One level of our network. Boxes represent in- and outputs of the layers, the arrows correspond to the operations performed by the respective layers. The spatial resolution is denoted by multiples of n, the number of channels by multiples of u. The convolution groups are not emphasized for simplicity.*

slice. Yet, it is possible to train networks for such shading effects with training images computed using a specific environment map. In this case, the network will implicitly "bake" the lighting information into the weights it learns. We still consider this useful in the same way as static environment maps are used in practice. Generalizing over different environment maps by using the latter as additional network input in a differently structured CNN remains future work.

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* 768×512 *px inputs. 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
AO	N_s, P_s	✗	✓	8	6	3	71 K	.805	9 ms
GI	N_s, P_s, L_{diff}	✓	✓	16	5	3	134 K	.798	28 ms
DoF	$D_{focal} \cdot B, D_s, L$	✓	✓	16	5	3	133 K	.959	27 ms
MB	F, L, D_s	✓	✓	16	5	3	133 K	.937	26 ms
SSS	P_s, R_{scatt}, L	✓	✓	16	5	3	133 K	.905	27 ms
AA	D_s, L	✓	✓	8	1	5	1.2 K	.982	1.8 ms
Multi	(see AO, DoF)	✓	✗	16	5	3	135 K	.933	26 ms
IBL	N_w, C_w, R	✓	✗	300	1	1	3.9 K	.973	21 ms
DO	N_w, N_s, P_s	✗	✗	16	5	3	135 K	.589	26 ms
RS	N_s	✓	✗	16	5	5	370 K	.622	80 ms

Ambient Occlusion Ambient occlusion, a prototypical screen space effect, 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 using ray-tracing with a constant effect range defined in world space units. In an actual application, the AO term is multiplied with the ambient lighting term before adding it to the image.

The CNN faithfully reproduces darkening in areas with nearby geometry (Fig. 5), the most noticeable difference to the reference being blurrier fine details. To evaluate how well our learned shader performs in comparison to optimized screen space AO techniques, in Fig. 7, we show a same-time comparison to Horizon-based Ambient Occlusion (HBAO) [BSD08] which is an efficient technique used in games. As our method does not have any parameters - apart from the network structure itself - that can be tweaked, we adjust HBAO to same computation time by choosing the same effect radius, and using 24 sampling steps into 16 sampling directions. On the test set, we achieve higher SSIM than HBAO which we consider remarkable given that our method has been learned by a machine. Furthermore, our method does not exhibit the high frequency banding artifacts which are typical for HBAO (Fig. 7, insets on top row) and creates less "cut off" indirect shadows where the screen space information is insufficient (Fig. 7, bottom insets) as the CNN was trained on unbiased data. We made AO the subject of further in-depth analysis of alternative network designs described in Sec. 6 and seen in Fig. 13, a) and b).

Diffuse Indirect Light A common challenge in rasterization-based real-time rendering is indirect lighting. To simplify the problem, the set of relevant light paths is often reduced to a single "indirect bounce", diffuse reflection [TL04] and restricted to a certain radius of influence. 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. The position of the light source is sampled uniformly at random per image. 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 applying the color of nearby lit objects (Fig. 5).

In Fig. 8 (top) we perform a same-time comparison in the same way as for HBAO above. The screen space competitor is SSGI by Ritschel et al. [RGS09] which achieves comparable speed when using a 12×12 px sampling pattern but loses out on achieved SSIM on the test set. This is not surprising as SSIM, as well as human observers, are sensitive to the MC noise it produces while Deep Shading generates smooth outputs.

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

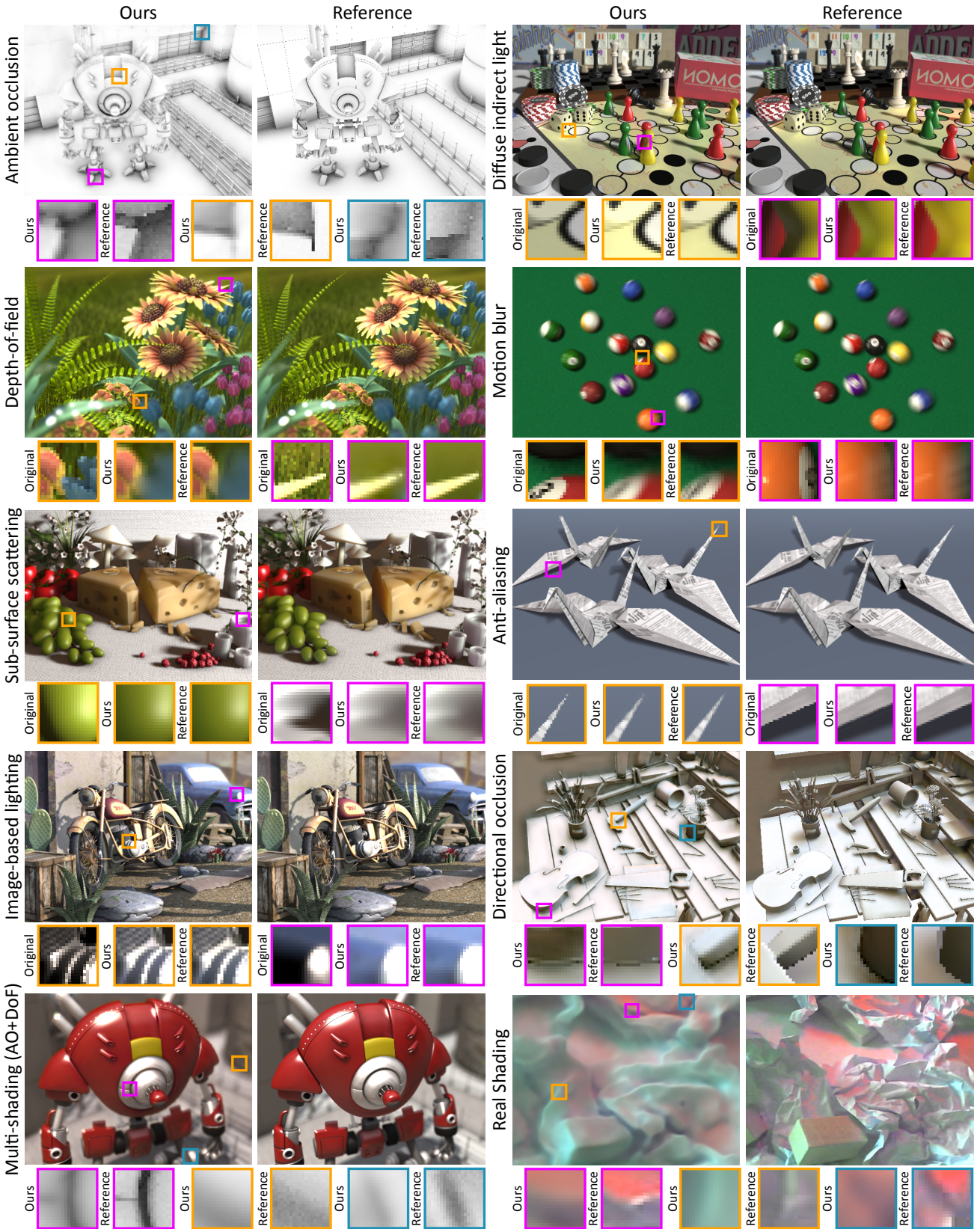

Figure 5: *Results of different Deep Shaders as discussed in Sec. 5. 'Original" shows the scenes without the respective effects. The supplemental material contains the images in higher resolution with corresponding input attributes.*

Figure 6: *Frames from animated sequences. See the supplemental for the corresponding video sequences and more effects.*

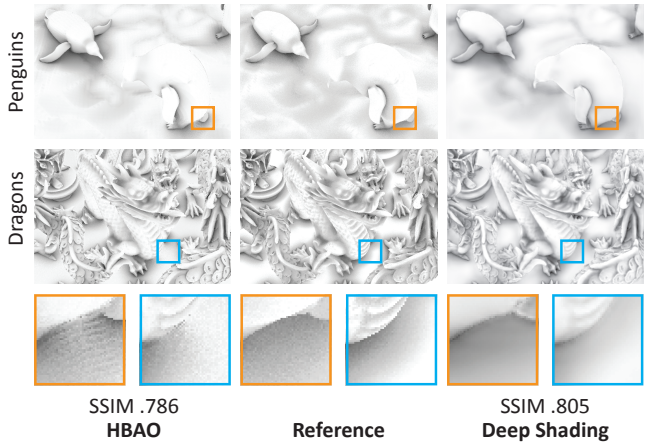

Figure 7: *In a same-time comparison, Deep Shading for AO is on-par with state-of-the-art methods like HBAO, both numerically and visually. The given SSIM values are w.r.t. the full AO test set.*

adaptive blurring of the sharp pinhole-camera image. We learn our own depth-of-field blur from sample data which we 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 well as on the circle of confusion. Both parameters are sampled randomly during data generation and then multiplied ($D_s \cdot B$) to form a single blurriness attribute which is fed to the network and permits direct manipulation of the DoF shallowness at run-time. To allow for depth order-dependent behavior of the shader, we also provide plain image depth. The Deep Shader again is trained independently for each channel, assuming a non-dispersive lens. The trained network blurs things in increasing distance from the focal plane by increasing extents. In Fig. 5, the sunflowers appear sharper than e.g., grass in the background or the leaves in front.

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 trajectories 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 [MHBO12]. For training, we randomly move objects inside

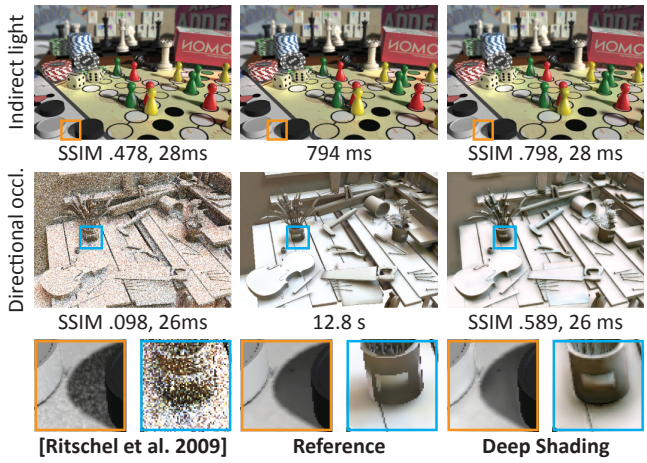

Figure 8: *Same-time comparisons for GI and DO, comparing with the method of Ritschel et al. [RGS09]. The SSIM values are w.r.t. the full test sets.*

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 a convincing way that manages to convey a sense of movement and comes close to the reference image (Fig. 5).

Sub-surface Scattering Simulating the scattering of light inside an object is crucial for achieving realistic appearance for translucent materials like wax and skin. A popular approximation to this is screen space sub-surface scattering (SSSS) [JSG09] which essentially applies a spatially-varying blurring kernel to the different color channels of the image. We produce training data at every pixel by iterating over all other pixels and applying Pixar's scattering profile [CB15] depending on the distance between the 3D position at the two pixels. After training the Deep Shader independently for all RGB channels on randomly textured training images with random parameters to the blurring profile we achieve images which transport the same sense of translucency as the reference method.

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) [Lot11] 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. 5) replaces jagged edges by smooth ones. While it cannot be expected to reach the same performance as the 8× 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 artifacts. At the

same time, the network learns to not over-blur interior texture areas that are properly sampled, but only blurs along depth discontinuities.

Image-based Lighting In image-based lighting a scene is shaded by sampling directions in an environment map to determine incoming radiance, assuming the latter is unblocked. The network is trained to render IBL based on diffuse and specular colors as well as gloss strengths using the Phong shading model which also depends on the surface normal and camera direction. As a special case, for IBL we also apply grouped convolution using three groups on the final convolution layer, effectively using one group per color channel. In an application, the IBL is typically added to shading from a small number of main light sources. As can be seen from the vehicles in Fig. 5, the network handles different material colors and levels of glossiness. The two main limitations are a slight color shift compared to the reference, as seen e.g., in the shadowed areas of the bikes' tires, and an upper bound on the level of glossiness. The latter is not surprising as the extreme here is a perfect mirror which would need a complete encoding of the illumination used in training, which has a resolution of several megapixels, into a few hundred network kernels.

Directional Occlusion Directional occlusion [RGS09] is a generalization of AO where sample directions are associated with radiance samples from an environment map and light from unblocked directions is accumulated. DO is applied by using it directly as ambient lighting term. As for AO, ray-tracing is used to resolve occluded directions within a fixed world-space radius. While the related AO works well, DO is more challenging for Deep Shading. The increased difficulty comes from indirect shadows now having different colors and appearing only for certain occlusion directions. As can be seen in Fig. 5, 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, and are harder to encode in a network. Despite these shortcomings, Deep Shading performs better than the original screen space DO (SSDO) [RGS09] given the same time-budget as seen in the bottom row of Fig. 8. Even for the 10×10 samples per pixel used, SSDO has problems handling high-frequencies in the environment map.

Multi Shading Finally, we learn a Deep Shader that combines several shading effects at once and computes a scene shaded with ambient occlusion to produce soft shadows and additional shallow depth-of-field. The network uses the union of attributes of the AO and DoF networks simultaneously. Note, that this single Deep Shader realizes both effects together in a single network. Unlike the AO network, the network output is not multiplied with the remaining shading but already applied by the CNN itself as the DoF component cannot be decoupled. An image generated using the network (Fig. 5) exhibits both effects present in the training data. Shallow depth-of-field is added and corners are darkened, as can be seen particularly well when applying the network to a fully white radiance input (cf. insets).

Real Shading As an addition, we demonstrate that Deep Shaders can not only be trained from renderings but also from actual photographs. In a prototypical experiment, we captured 12,000 samples of RGB and depth images of a scene of neutrally colored objects in a characteristically lit environment (Fig. 9, left) using a standard RGBD camera (Creative Senz3D) by fixing the position of the camera while moving and rearranging different objects in front of it. The main light sources are distant enough to create an almost directional direct lighting environment, where incident lighting only depends on the normal in camera space, which is at the same time interacting with visibility changes due to occlusions by nearby geometry and global illumination effects in general. Using the known camera intrinsics, we derive camera positions from the captured depth values which are in turn used to determine camera space normals. After registering normal and RGB data, we train a network to map the former to the latter.

Images generated by the network for real and rendered depth data (Fig. 9, middle) reproduce the lighting environment with red or bluish tints depending on the objects' normals and including AO-like darkening in corners and creases, even for geometry very different to the training data. The main limitation is due to the precision of our depth camera which cannot capture fine details in the geometry. Consequently, geometry details below a certain scale are ignored by the network which is apparent when comparing its output on training depth images to ground truth (Fig. 9).

In the two rightmost columns of Fig. 9, we compare our method to the probably closest CNN-based application, the style transfer approach by Gatys et al. [GEB15] which differs in its objective in that its input consists of an RGB image without auxiliary geometric information. While style transfer tries to change hues and structures to match the example, it becomes evident that a reproduction of light transport aspects, like shadows or bounces, really needs additional attributes such as surface normals.

We believe that more detailed shading could be learned using improved depth sensors and that an extension to proper world normals, e.g., by mounting the camera on a gonioreflectometer, or even to other varying attributes like albedo, is only a matter of acquisition.

Animations For applying the AO and GI Deep Shaders to dynamic scenes, we found it beneficial to increase temporal coherence by warping CNN outputs from a small number of previous frames to the current view and blending them with the current result [SYM*12]. Please see the supplemental video for scenes with moving cameras and moving or deforming objects. Fig. 6 shows several sample frames for fully dynamic scenes.

6. Analysis

In the first part of this section, we address some shortcomings in the form of typical artifacts produced by our method and also discuss how the network reacts when applying it to new resolutions and attributes rendered with a different field-of-view value. The remainder of the section explores some of the countless alternative ways to apply CNNs, and machine learning in general, to the problem of screen space shading. We cover different choices of actual network structure (Sec. 6.2), loss function (Sec. 6.3) and training data

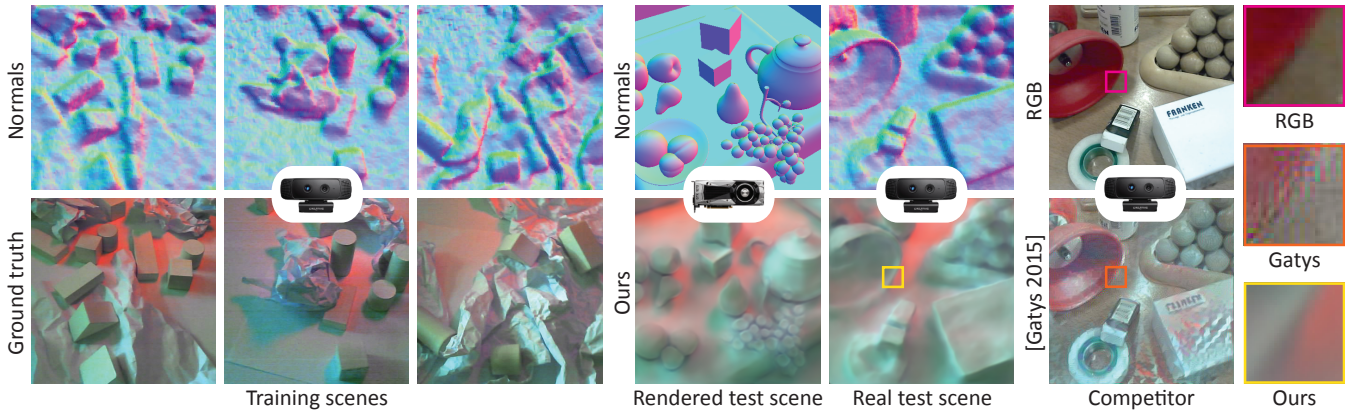

Figure 9: *Deep Shading learnt from RGBD video that captures screen space normals* (top) *and appearance* (bottom). *Deep Shading can learn the correlation including directional light, occlusion and bounces and transfer it to novel synthetic or captured normal images* (4th and 5th column). *This performs better than established deep-learning based style transfer [GEB15] from RGB to RGB images* (last column).

anatomy (Sec. 6.4) as well two techniques competing with deep CNNs, namely multi-layer perceptrons (MLPs) and random forest (RFs) (Sec. 6.5).

6.1. Visual Analysis

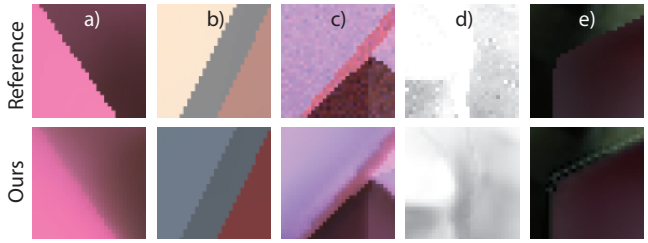

Figure 10: *Typical artifacts of our approach:* a): *Blur.* b): *Color shift.* c): *Ringing.* d): *Background darkening.* e): *Attribute discontinuities.*

Typical Artifacts Light transport can become highly complex and the mapping from screen space attributes to shading is inherently ambiguous due to partial information, hence we cannot expect a CNN to act correctly for every given input. Even what looks plausible in a static image may start to look painterly or surrealistic when seen in motion: patterns resembling correct shading emerge but being inconsistent with the laws of optics and with each other. We show exemplary artifacts in Fig. 10. Capturing high frequencies is a key challenge for Deep Shaders (Fig. 10, a). If the network does not have enough capacity or was not trained enough the results might over-blur with respect to the reference. We consider this a graceful degradation compared to typical artifacts of man-made shaders such as ringing or Monte Carlo (MC) noise (Fig. 7) which are highly unstable over time and unnatural with respect to natural image statistics. Sometimes, networks trained on RGB tend to produce color shifts (Fig. 10, b) which can typically be weakened by increasing the number of features. CNN-learned filters may also introduce high frequencies resembling ringing due to false-positive neuron activations resulting from overfitting (Fig. 10, c). Sometimes effects propagate into the wrong direction in world space, e.g., geometry may cast occlusions on things behind it (Fig. 10, d). At attribute discontinuities, the SSIM loss lacking an inter-channel prior sometimes fails to prevent color ringing (Fig. 10, e).

Range of Values While many attributes used as an input to Deep Shaders are limited to a certain range e.g., normals, others are theoretically unbounded and a CNN might react unexpectedly when confronted with values far outside the training range. We tackle this by, first, choosing particularly large ranges of possible values during training and second, if necessary, scaling inputs on which light transport depends linearly (e.g., the unblurred input to DoF) in a linear way before feeding them to the network and undoing this transformation afterwards.

Effect Radius Typically, screen space shading is faded out based on a distance term and only accounts for a limited spatial neighborhood. As we train in one resolution but later apply the same trained network also to different resolutions, 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 scale the effect radius accordingly, dividing it by N. While the effect radius is not an input to the network but fixed in the training data, it can still be adjusted at test time by scaling the attributes determining the spatial scale of the effect, e.g., of the camera space positions in the case of AO, DO or GI, or of the distance to the focal plane in the case of DoF. To conclude, effect radius and resolution can be changed at virtually no additional cost (per pixel) without re-training the network.

Internal Camera Parameters As we compute our training data using a fixed FOV (50°), it is not clear how the trained networks perform on framebuffers rendered using a different FOV. Fig. 12 investigates the influence of a FOV mismatch on image quality. To keep the image content as similar as possible while changing FOV, we performed a dolly-zoom. Judging from the minimal fluctuation of the error, the network is absolutely robust to FOV mismatches.

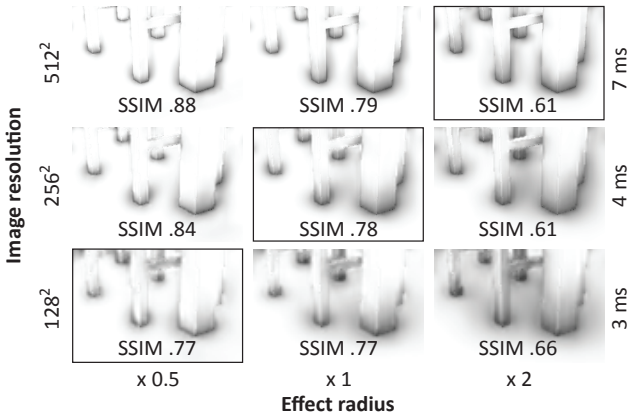

Figure 11: *Increasing (decreasing) the resolution shrinks (enlarges) effect size relative to the resolution. The radius can be increased (decreased) again with no effects on timings by scaling the input attribute determining the radius accordingly, e.g., the positions for AO. Consequently, images on the diagonal are similar. All images show outputs produced by the same network. The time for each row is identical. The SSIM is higher for smaller effect radii that are easier to reproduce.*

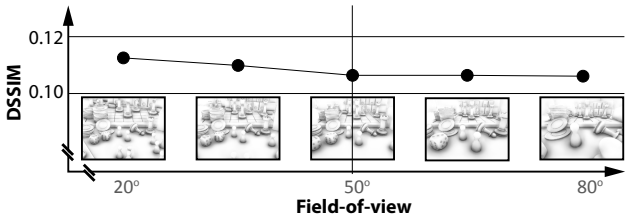

Figure 12: *Effect of FOV on image quality. The horizontal axis is FOV in degrees. The central line is the reference of 50°. The vertical axis is DSSIM error w.r.t. the ray-traced reference. Note that the vertical axis spans only a small difference (.106 to .112), indicating FOV has no large impact on visual quality.*

6.2. Network Structure

To better understand how the structural parameters of our CNN architecture control its expressiveness and computational demand, we investigate two modes of variation: varying spatial extent of the kernels as well as the number of kernels on the first level u_0, which also determines the number of kernels for the remaining levels (Sec. 4.2). We seek the smallest network with adequate learning capacity, that generalizes well on previously unseen data. The results are summarized in Fig. 13, a, b) for the example of AO.

Spatial Kernel Size Fig. 13, (a) (green and yellow lines) shows the evolution of training, validation 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 with a kernel size of 5×5, the training profile slightly lags behind that for kernel size of 3×3, but both approach a similar test loss at 100k iterations, i.e., networks have sufficient capacity to approximate the mapping, with neither beginning to overfit. We observe a similar relative timing relationship between the pairs of networks with $u_0 = 4$ and $u_0 = 16$. As, regarding running time, the one with a kernel size of 3×3 is about twice as fast as the one with 5×5 we opt for the former. Regarding memory consumption, different spatial kernel sizes have only a small impact as the memory needed to store convolution parameters is insignificant compared to the memory usage of the intermediate representations.

Initial Number of Kernels The orthogonal mode of variation is u_0, the number of kernels on the first level, also influencing the kernel counts of subsequent layers which are expressed as multiples of u_0. Again, we plot the training, validation and test errors, this time for different u_0 (Fig. 13, a, green and blue lines, yellow and purple lines). Reducing the number to $u_0 = 4$ clearly leads to a loss of expressiveness as can be seen from both the training and test loss. Further, nets with $u_0 = 16$ perform only slightly better than those with $u_0 = 8$ (Fig. 13, b,) while losing out in compute time by more than a factor of 6. This is in part due to increased memory consumption, in particular for the intermediate representations.

Structural Choices for Other Effects We perform an analogous analysis for 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 over-fit / underfit characteristics exhibited by the the train-test error curves. Tbl. 1 indicates the final choices of network structure. Additionally, the corresponding train-test error curves are shown in Fig. 13, (c), with their test loss-vs.-speed characteristics captured in Fig. 13, (d).

The number of iterations shown in Fig. 13, (a) and (c), though sufficient to make decisions about the structural parameters, still leave the network with scope to learn more (indicated by negative slopes of the train-test curves). We therefore resumed training for the respective optimal choices of $u_0 = 8$ for about 400k additional iterations until the losses settled completely.

It is worth noting that, while the training errors we measured are smaller than the respective validation errors as expected, in some cases we measured smaller test than training errors (e.g., Fig. 13, a). This is possible because we use disjoint sets of scenes for training and validation on one hand and testing on the other hand, i.e., the scenes used for testing might be "easier" to tackle for the network. For example, they might contain less cases where the deferred shading inputs to the network are insufficient to derive ground truth results because of lacking information about occluded geometry.

6.3. Choice of Loss Function

The choice of loss function in the optimization has a significant impact on how Deep Shading will be perceived by a human observer. We trained the same network structure using the common L1 and L2 losses as well as the perceptual SSIM metric and also using combinations of the three. Fig. 14 shows a visual comparison of results produced by the respective nets. We found L1 and L2 to be prone to producing halos instead of fading effects out smoothly as can be seen in the first two columns. The combination of L2 with SSIM also exhibits these kind of artifacts to a lesser extent. SSIM

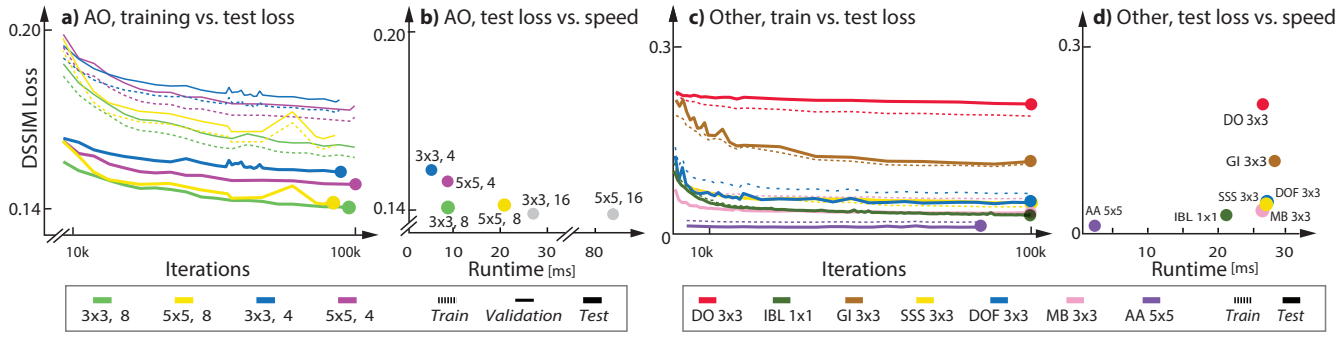

Figure 13: *Analysis of different network structures. We here compare different design choices for different effects in terms of compute time and DSSIM loss. The vertical axes on all plots corresponds to DSSIM loss (less is better). The horizontal axes of the line plots range over the number of training iterations. The scatter plots have computation time of the Deep Shader as the horizontal axis.* a) *Train, test and validation 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.* c) *Loss as a function of iterations for the chosen designs for other effects (curves).* d) *Comparison of compute time and final loss for the other effects, as a means of placing their relative complexity.*

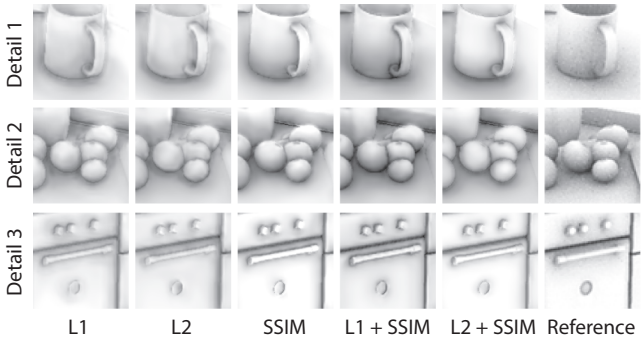

Figure 14: *Outputs produced by the same network trained with different loss functions for the case of AO.*

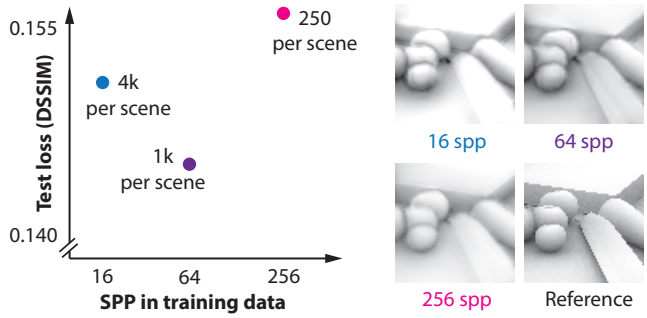

Figure 15: Left: *Data points correspond to the same time budget to produce training data but using different numbers of samples per pixel. The resulting number of individual views per scene (before data augmentation) is given for each point.* Right: *AO produced by the corresponding trained networks.*

and SSIM + L1 both produce visually pleasing results with pure SSIM being more faithful to the amount of contrast found in the reference images.

6.4. Training Data Trade-offs

Ideally, a training set consists of a vast collection of images with no imperfections from Monte Carlo noise and showing a large number of different scenes. Yet, in practice, the time budget to produce training data is typically limited and the question how to spend this time best arises. In this section, we investigate different trade-offs.

Amount of Noise vs. Image Set Size The time spent to generate a training set is roughly linear in both, the number of MC samples taken per pixel and the number of individual images rendered (before data augmentation), e.g., we can render twice as many images if we only use half as many samples per pixel. To investigate to which extent noise in the training data affects the quality of the trained network and to find out which trade-off between the number of samples and individual views should be taken, we performed a same-time comparison using AO as an example. As can be seen from Fig. 15, a larger number of views per scene is typically more desirable than noiseless images. Only for very low sample counts leading to excessive noise (around 16 spp in the case of AO), the variance in the individual images begins to hinder proper training of a network.

Scene Diversity Another factor that has influence on the quality of the trained Deep Shaders is the diversity of scenes in the training set, e.g., a CNN that has only seen round objects during training will fail to correctly re-produce its effect for square objects. In our training sets, we use 1000 views from each of 10 different scenes as our starting points (cf. Sec. 6.4). To see how well CNNs perform for less diverse data we produced DO training sets of the same total size but for a lower number of different scenes. DO was chosen as we observed it to be particularly sensitive to the scene diversity. The resulting DSSIM values for (the same) test set are plotted in Fig. 16, left. While the error for five scenes compared to a single

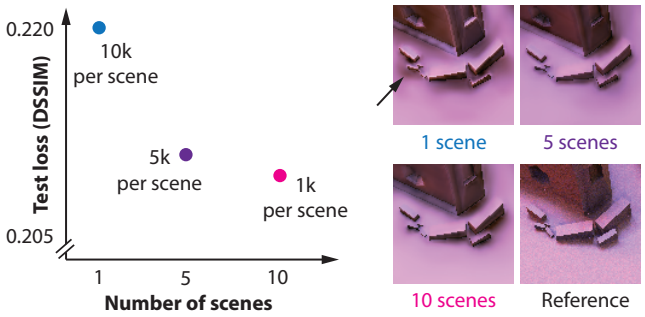

Figure 16: Left: *Data points correspond to the same time budget to produce training data but with different trade-offs regarding the scene count.* Right: *Patches from a test scene.*

one is 5% smaller, increasing the number further to 10 scenes leads to only a smaller advantage of about another 1% which indicates that our scene set is of acceptable diversity. In the case of DO, the difference in the loss visually translates to a more correct placement of darkening. A network trained with only one scene tends to create "phantom occlusions" in free spaces (Fig. 16, right).

6.5. Comparison With Other Regression Techniques

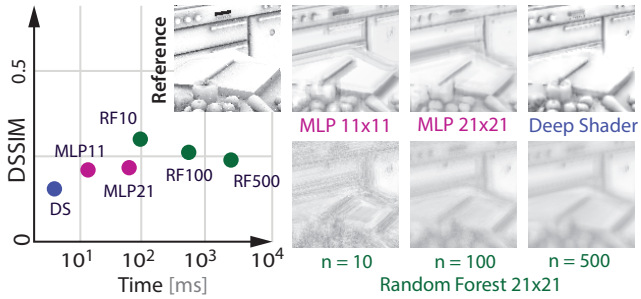

Figure 17: *AO computed using random forests, shallow MLPs and Deep Shading. The vertical axis is image error (DSSIM) on a linear scale. The horizontal axis is compute time for* 256×256 *px images on a logarithmic scale.* n *indicates the number of trees.*

Besides deep CNNs, competing approaches such as shallow multilayer perceptrons (MLPs) and random forests (RFs) [CS13] could supposedly be used for our objective, having found use in other image synthesis tasks such as estimation of filter parameters [KBS15] or relighting [RWG*13, RDL*15]. To investigate these alternatives, we perform a comparison, training our Deep Shader, MLPs, and RFs to produce AO for 256×256 px deferred shading buffers. As for MLPs and RFs direct regression on the full buffers would be prohibitively expensive, we train the former on patches of 11×11 and 21×21 px and the latter on patches of 21×21 px, to predict AO for the central pixel. To compute AO in high resolution later on, they are swept across the whole input buffers. The MLPs consist of 2 hidden layers with 50 nodes each while the RFs are split across four cores and have a minimum of five samples per leaf.

As SSIM can only be computed on patches but MLPs and RFs have to be restricted to single pixels in training due to their computational demand, we resort to training both methods as well as our CNN using $\mathscr{L}_2$ loss for this comparison. We still evaluate all approaches using SSIM. For MLPs, we measure their execution time using the same OpenGL implementation as used for our CNNs while we employ scikit-learn [PVG*11] on a regular workstation for the RFs. Fig. 17, shows the relative speed and quality of MLPs and RFs compared to Deep Shading. Pixel-wise predictions with RFs clearly lose out on both visual quality and run time. RF run times increase linearly with the number of trees, more of which are necessary to construct a better ensemble. Even with more readily parallelizable variants of RFs [BZM07], there would have to be a run time improvement of more than two orders of magnitude to be comparable to a Deep Shader. For the MLPs, we actually observe a degradation of image quality with increasing patch size: as the number of parameters increases quadratically with the diameter of the filter, the resulting MLP becomes prone to overfitting. Only far more training data and training iterations could mitigate this problem.

In conclusion, deep CNNs have an edge over competing approaches for our setting as they allow for large receptive fields through stacked (smaller) convolutions and downsampling, without an exponential increase in parameter count, while leveraging the expressiveness of deep representations [Has86].

7. Conclusion

We have leveraged deep learning to turn attributes of virtual scenes into appearance, showing that CNNs can model any screen space effect as well as combinations of them at competitive quality and speed. This is a first proof that full image synthesis can be learned from data without human intervention and programming effort.

Our limitations are as for common screen space techniques, namely missing shading from objects not contained in the image due to occlusion, clipping or culling. But we also inherit its benefits such as handling of large, dynamic scenes in an output-sensitive manner. In future refinements, Deep Shaders could even learn to fill-in more missing information than they already do, e.g., for AO (Fig. 7) where the network seems to complete geometry configurations in a natural way. Different scene representations, e.g., surfels or patches, could help achieving this. Some shading effects are due to complex relations between screen space attributes. Not all configurations are resolved correctly by networks with limited capacity, such as ours which run at interactive rates. We however observe that typical artifacts are less salient than from human-designed shaders which suffer from ringing or MC noise (Fig. 7). Instead, CNNs reproduce patterns encountered during training which are inherently natural as they have been computed using traditional light transport methods and appear visually plausible. A perceptual study could verify this. Temporal coherence can be addressed in the same way as for classic screen space shading by temporal integration, resulting in mostly coherent behavior as seen in the supplemental video.

We have demonstrated that Deep Shading can achieve better approximations than common methods in the case of AO given the same time budget (Fig. 7). We hope that even more diverse training data, advances in learning methods, and new types of representations or losses will allow surpassing human-programmed shaders for more effects in a not-so-distant future.

References

[BSD08] BAVOIL L., SAINZ M., DIMITROV R.: Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 Talks* (2008). 6

[BZM07] BOSCH A., ZISSERMAN A., MUNOZ X.: Image classification using random forests and ferns. In *Proc. ICCV* (2007), pp. 1–8. 13

[CB15] CHRISTENSEN P. H., BURLEY B.: *Approximate Reflectance Profiles for Efficient Subsurface Scattering*. Tech. rep., 2015. 4, 8

[CMK*14] CIMPOI M., MAJI S., KOKKINOS I., MOHAMED S., , VEDALDI A.: Describing textures in the wild. In *CVPR* (2014). 5

[CS13] CRIMINISI A., SHOTTON J.: *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media, 2013. 13

[Dac11] DACHSBACHER C.: Analyzing visibility configurations. *IEEE Trans. Vis. and Comp. Graph. 17*, 4 (2011), 475–86. 2

[DTSB15] DOSOVITSKIY A., TOBIAS SPRINGENBERG J., BROX T.: Learning to generate chairs with convolutional neural networks. In *Proc. CVPR* (2015), pp. 1538–1546. 3

[EPF14] EIGEN D., PUHRSCH C., FERGUS R.: Depth map prediction from a single image using a multi-scale deep network. In *Proc. NIPS* (2014), pp. 2366–74. 2

[ERS13] ELEK O., RITSCHEL T., SEIDEL H.-P.: Real-time screen-space scattering in homogeneous environments. *IEEE Computer Graph. and App.*, 3 (2013), 53–65. 2

[FFL11] FARBMAN Z., FATTAL R., LISCHINSKI D.: Convolution pyramids. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 6 (2011), 175:1–175:8. 2

[GDDM14] GIRSHICK R., DONAHUE J., DARRELL T., MALIK J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR* (2014), pp. 580–7. 2

[GEB15] GATYS L. A., ECKER A. S., BETHGE M.: A neural algorithm of artistic style. *arXiv 1508.06576* (2015). 2, 9, 10

[HAGM15] HARIHARAN B., ARBELÁEZ P., GIRSHICK R., MALIK J.: Hypercolumns for object segmentation and fine-grained localization. In *Proc. CVPR* (2015). 3

[Has86] HASTAD J.: Almost optimal lower bounds for small depth circuits. In *ACM Theory of computing* (1986), ACM, pp. 6–20. 13

[Her03] HERTZMANN A.: Machine learning for computer graphics: A manifesto and tutorial. In *Proc. Pacific Graphics* (2003). 2

[JDA*11] JOHNSON M. K., DALE K., AVIDAN S., PFISTER H., FREEMAN W. T., MATUSIK W.: CG2Real: Improving the realism of computer generated images using a large collection of photographs. *IEE Trans. Vis. and Comp. Graph. 17*, 9 (2011), 1273–85. 2

[JSD*14] JIA Y., SHELHAMER E., DONAHUE J., KARAYEV S., LONG J., GIRSHICK R., GUADARRAMA S., DARRELL T.: Caffe: Convolutional architecture for fast feature embedding. In *Proc. ACM Multimedia* (2014), pp. 675–8. 5

[JSG09] JIMENEZ J., SUNDSTEDT V., GUTIERREZ D.: Screen-space perceptual rendering of human skin. *ACM Trans. Applied Perception 6*, 4 (2009), 23. 2, 8

[Kaj86] KAJIYA J. T.: The rendering equation. In *ACM SIGGRAPH* (1986), vol. 20, pp. 143–50. 2

[KBS15] KALANTARI N. K., BAKO S., SEN P.: A machine learning approach for filtering Monte Carlo noise. *ACM Trans. Graph. (Proc. SIGGRAPH)* (2015). 2, 13

[KSH12] KRIZHEVSKY A., SUTSKEVER I., HINTON G. E.: Imagenet classification with deep convolutional neural networks. In *Proc. NIPS* (2012), pp. 1097–105. 2

[KWKT15] KULKARNI T. D., WHITNEY W., KOHLI P., TENENBAUM J. B.: Deep convolutional inverse graphics network. In *Proc. NIPS* (2015). 3

[Lot11] LOTTES T.: FXAA. Nvidia White Paper, 2011. 2, 8

[LSD15] LONG J., SHELHAMER E., DARRELL T.: Fully convolutional networks for semantic segmentation. In *Proc. CVPR* (2015). 3

[MHBO12] MCGUIRE M., HENNESSY P., BUKOWSKI M., OSMAN B.: A reconstruction filter for plausible motion blur. In *Proc. ACM i3D* (2012), pp. 135–42. 2, 8

[MHN13] MAAS A. L., HANNUN A. Y., NG A. Y.: Rectifier nonlinearities improve neural network acoustic models. *Proc. ICML 30* (2013). 5

[Mit07] MITTRING M.: Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 Courses* (2007), pp. 97–121. 2

[NAM*16] NALBACH O., ARABADZHIYSKA E., MEHTA D., SEIDEL H., RITSCHEL T.: Deep shading: Convolutional neural networks for screen-space shading. *arXiv 1603.06078* (2016). 1

[NKF09] NOWROUZEZAHRAI D., KALOGERAKIS E., FIUME E.: Shadowing dynamic scenes with arbitrary BRDFs. In *Comp. Graph. Forum* (2009), vol. 28, pp. 249–58. 2

[NMY15] NARIHIRA T., MAIRE M., YU S. X.: Direct intrinsics: Learning albedo-shading decomposition by convolutional regression. In *Proc. CVPR* (2015), pp. 2992–3. 2

[OLG*07] OWENS J. D., LUEBKE D., GOVINDARAJU N., HARRIS M., KRÜGER J., LEFOHN A. E., PURCELL T. J.: A survey of general-purpose computation on graphics hardware. In *Comp. Graph. Forum* (2007), vol. 26, pp. 80–113. 2

[Pho75] PHONG B. T.: Illumination for computer generated pictures. *Communications of the ACM 18*, 6 (1975), 311–317. 4

[PVG*11] PEDREGOSA F., VAROQUAUX G., GRAMFORT A., MICHEL V., THIRION B., GRISEL O., BLONDEL M., PRETTENHOFER P., WEISS R., DUBOURG V., VANDERPLAS J., PASSOS A., COURNAPEAU D., BRUCHER M., PERROT M., DUCHESNAY E.: Scikit-learn: Machine learning in Python. *arXiv 1201.0490* (2011). 13

[RDL*15] REN P., DONG Y., LIN S., TONG X., GUO B.: Image based relighting using neural networks. *ACM Trans. Graph. (TOG) 34*, 4 (2015), 111. 2, 13

[RFB15] RONNEBERGER O., FISCHER P., BROX T.: U-Net: Convolutional networks for biomedical image segmentation. In *Proc. MICAI*. 2015, pp. 234–41. 3

[RGS09] RITSCHEL T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *Proc. ACM i3D* (2009), pp. 75–82. 2, 6, 8, 9

[Rok93] ROKITA P.: Fast generation of depth of field effects in computer graphics. *Computers & Graphics 17*, 5 (1993), 593–95. 2

[RWG*13] REN P., WANG J., GONG M., LIN S., TONG X., GUO B.: Global illumination with radiance regression functions. *ACM Trans. Graph. (Proc. SIGGRAPH) 32*, 4 (2013), 130. 2, 13

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-d shapes. In *ACM SIGGRAPH Computer Graphics* (1990), vol. 24, pp. 197–206. 2

[SYM*12] SCHERZER D., YANG L., MATTAUSCH O., NEHAB D., SANDER P. V., WIMMER M., EISEMANN E.: Temporal coherence methods in real-time rendering. *Comp. Graph. Forum 31*, 8 (2012), 2378–2408. 9

[TL04] TABELLION E., LAMORLETTE A.: An approximate global illumination system for computer generated films. *ACM Trans. Graph. (Proc., SIGGRAPH) 23*, 3 (2004), 469–76. 6

[WFG15] WANG X., FOUHEY D. F., GUPTA A.: Designing deep networks for surface normal estimation. *Proc. CVPR* (2015). 2

[Zei12] ZEILER M. D.: ADADELTA: an adaptive learning rate method. *CoRR abs/1212.5701* (2012). 5

[ZGFK17] ZHAO H., GALLO O., FROSIO I., KAUTZ J.: Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging 3*, 1 (2017), 47–57. 3, 5