Report on Deep-learning the Latent Space of Light Transport

ALEXANDER EPPLE, Technical University of Munich

Light transport is a complex and difficult problem, as it involves the unsolvable rendering equation. While approximations exist, namely in the form of raytracing, they are slow to compute and not yet practical for real time rendering [Akenine-Mller et al. 2018]. This report covers a novel approach, as introduced by [Hermosilla et al. 2019], which tries to solve this problem by modelling the transport of light as a deep neural network. To add to the existing work, we provide a discussion and ideas for future work.

Additional Key Words and Phrases: Deep learning; Convolutional neural networks; Point clouds; Rendering; Shading

1 INTRODUCTION

In the field of computer graphics, rendering is considered to be the process of generating 2D images from virtual cameras, light sources and 3D objects [Akenine-Mller et al. 2018] and simulates the transport of light. While artificial intelligence has been used for rendering, for example in *Deep Shading* [Nalbach et al. 2017], little research has been done to directly map a 3D scene to the final, shaded image. The authors of the original work thus propose an approach that directly learns the latent space of light transport from 3D point clouds.

The proposed network is split into two steps and features end-to-end learning from the point cloud to the final, shaded image. The network is then trained to output ambient occlusion (AO), global illumination (GI) and subsurface scattering (SSS) effects, which are subsequently evaluated. In this report, we provide additional evaluation of this novel approach in the form of a discussion and ideas for future work.

2 TRAINING DATA GENERATION

The first step in training a novel network, such as this one, is generating training data. The authors aim to teach the network three shading effects, GI, AO and SSS. Hence, separate training sets are created. To generate the training data, 1000 models from two datasets are randomly chosen and uniformly but randomly sampled at 20.000 surface points. The models are on a ground plane and scaled to the same size for consistency.

The shading effects are computed using raytracing for each model and each surface point. While direct lighting is ignored, as the effects are all due to higher order bounces, indirect light is generated from 30 environment maps. The training data consists of different inputs and outputs depending on the effect. For AO, the inputs are positions and normals, with the output being the AO value as a scalar. GI needs more information and has additional inputs, namely diffuse albedo and direct illumination irradiance, whereas the output is indirect irradiance. Finally, SSS has the same inputs as GI and additionally the absorption coefficient and index of refracion. The paper does not state the output specifically, but it is most likely also the scattered irradiance. The choice for irradiance instead of radiant exitance is also explained, as irradiance allows for texture modulation.

The training data is then split into training, validation and test sets, consisting of 20.000, 1.000 and 2.500 point clouds respectively. Additionally, they generate a dataset of animated models to test generalization and whether the model has temporal stability.

This report is a part of the lecture, Master-Seminar - Deep Learning in Computer Graphics, Informatics 15, Technical University of Munich.

The original work is introduced by [Hermosilla et al. 2019].

3 NETWORK ARCHITECTURE

The network is split into two components, the first step operates on the 3D point cloud alone, while the second step transforms the 3D information to the 2D image.

3.1 Monte Carlo Convolutions

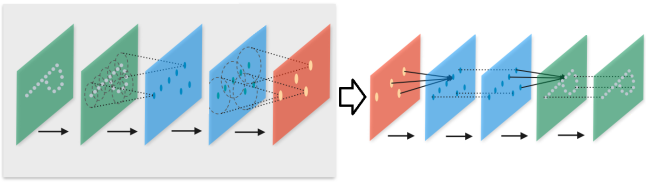

Fig. 1. An example of a 2D point cloud sampling hierarchy, adapted from [Qi et al. 2017].

To understand the proposed networks' architecture, it is critical to first look at its core building blocks, which in this case are Monte Carlo convolutions (MCC). As the network operates on point clouds, which are often irregular and unstructured, convolutions and feature abstraction needs to be invariant to sampling. Hermosilla *et al.* [Hermosilla et al. 2018]. Qi *et al.* note, that compared to CNNs, which have data on regular grids with uniform constant density, deciding local neighbourhood is more challenging. They also state, that it nevertheless is beneficial and important to extract local features and group them to produce higher level features, as this allows for better generalizability [Qi et al. 2017].

Monte Carlo (MC) methods use random samples to approximate the value of an integral [Kalos and Whitlock 2009]. As convolutions are integrals, convolutions of point clouds can be solved using MC integration. This allows for an efficient, scalable and invariant way of up- and downsampling point clouds.

$$(f * g)(x) \approx \frac{1}{\|N(x)\|} \sum_{j \in N(x)} \frac{f(y_j) g(\frac{x - y_j}{r})}{p(y_j|x)} \tag{1}$$

Equation 1 shows the MCC function as described in [Hermosilla et al. 2018]: For a point x, the neighbourhood $N(x)$ within the receptive field of radius r is used to compute the estimate. The probability density function $p(y_j|x)$ needs to be estimated as well and roughly correlates to how dense or sparse samples are in the receptive field.

The learnable kernel g maps the offsets $x - y_j$ to scalar weights of a multi-layer perceptron (MLP).

Additionally, parallel Poisson disk sampling is used to create sampling hierarchies, as they serve scalability and allow bounding the sample count. These hierarchies can be compared to increasing or reducing the resolution of an image.

Figure 1 shows how a sampling hierarchy for point clouds can look like and how it could be constructed.

3.2 3D Step

The first step consists of 10 MCCs that takes the scene's 3D point cloud, consisting of n samples, as input. Each sample is labeled with the required attributes for the shading effect, for AO each sample would therefore feature its position and normal. The network is built on the encoder-decoder principle, which allows transport of local features to the global level. This architecture was chosen because global shading effects also result from the culmination of many localized interactions. This step is marked as *3D Step* in figure 4.

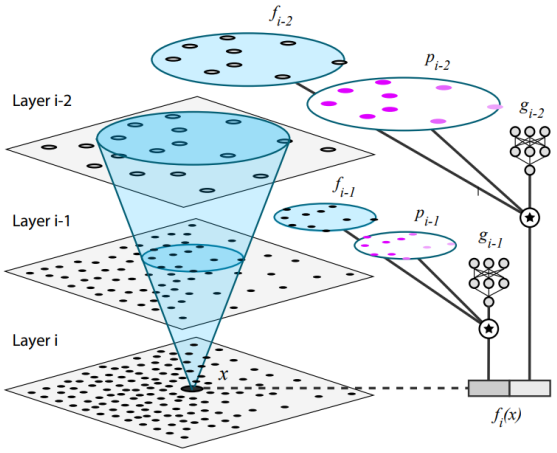

Fig. 2. An example of applying a MCC to a point in a 2D Poisson disk sampling hierarchy, from [Hermosilla et al. 2018]

Each convolution takes a slice of a point cloud hierarchy as input. Each sample of which has its own feature vector. This hierarchy is created using parallel Poisson disk sampling as described in 3.1 and ensures both a good distribution of samples and having a local and global version of the point cloud available. An example for this can be seen in figure 2. Here the feature vectors from each level are concatenated for the sample point x. The encoder doubles the features on each level, from an initial 8 to 64 on the deepest level. It also has an additional within-level convolution on each level, allowing for non-linear feature adjustments. The convolutions between levels make it possible to transfer learned, local features to the global view. This is also evident, as the radius R of the receptive field also doubles from level to level, resulting in a larger sphere of influence for each point.

The global features are then transferred back to the most shallow level, which leaves each 3D point with a latent space encoding of its impact on the per-pixel shading effect. The encoding roughly contains the relevant information for other points, depending on shading effect.

3.3 2D Step

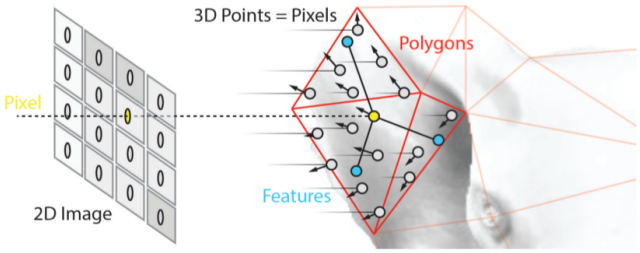

Fig. 3. Shading a single pixel, taken from [Hermosilla et al. 2019].

The second step consists of one MCC to transfer learned features between the whole point cloud and a subset of it, as well as a fully connected (FC) layer that outputs the shaded image. This step is labeled *2D Step* in the overview image 4. The feature transfer differs for training and testing. During training it is applied to a subset of the original sample points, while it performed on pixel / point pairs as seen by a camera during testing. There is no difference between points seen by a camera and random samples, but using a random subset of the original point cloud allows for efficient training.

Training is done by taking the $\mathcal{L}_2$ loss of the network output and the real shading value, as computed by raytracing during training data generation. This also makes end-to-end training possible. The pixel / point pairs during testing are labeled with the necessary attributes for shading, such as positions and normals for AO. The pairs are the closest surface points of the scene point cloud, as seen by a camera for each pixel of its output image. This approximation is necessary, as the amount of pixels is usually far greater than the size of the point cloud, which would prohibit scalabilty.

The single MCC is transferring the features to the points used for shading, which is done by the FC layer in a single convolution. This last convolution computes a final color using the features and input attributes, for example a monochormatic gray value for AO. The points used for this convolution are the pixel point and a certain amount of additional points within a set Possion disk radius around this point. The additional points, which are selected from the original scene point cloud, determine the quality of the result and are controlled by the radius of the receptive field. This performance / quality trade-off makes it possible to have constant performance, as every pixel computes the integral from approximately the same amount of points. This final step is also displayed in figure 3.

4 EVALUATION

In the original work, evaluation is done both qualitatively and quantitatively by comparing their method to a screen space approach and to two lesser versions of their network, featuring either the 3D or 2D step. Outperforming those versions means that both the internal 3D features, as well as the learned 2D sampling, serves a purpose. All methods are compared to a path traced reference for

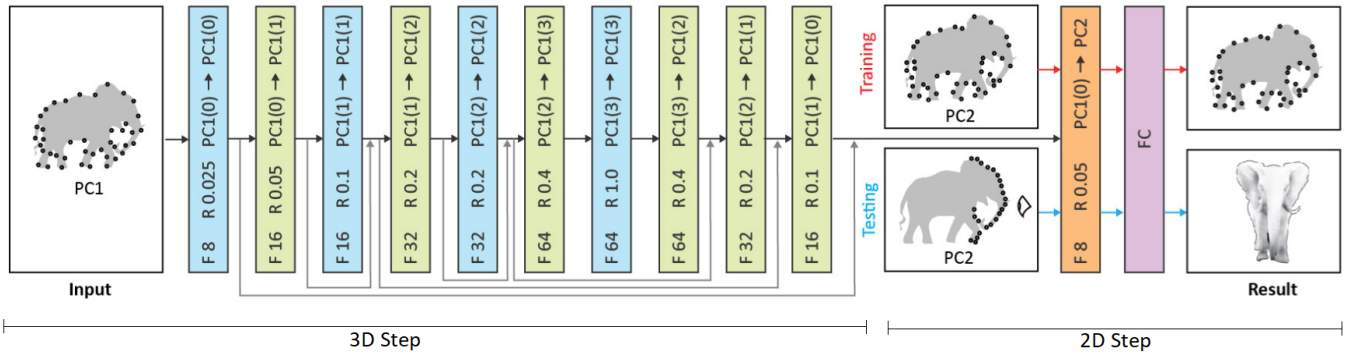

Fig. 4. An overview of the network architecture and its layers, adapted from [Hermosilla et al. 2019]

comparison and three metrics are employed, one on the 3D point cloud and two on the 2D image.

The results of the quantitative evaluations are not fully conclusive, as the full network only heavily outperforms the other approaches in the case of SSS. While AO results are better, the difference is not as drastic. When it comes to GI, all methods perform roughly the same. The authors do note, however, that their approach has the best temporal coherence, meaning the shading of animated objects changes realistically and gradually.

An overview of the results for AO can be seen in figure 5. It can be observed, that both screen space and 2D-only methods mostly resolve local, 3D-only methods mostly global features, whereas the combined method outputs both.

5 DISCUSSION

With the approach layed out and evaluated, we will now provide a short discussion highlighting both possible advantages and limitations of the method. As their method seems to be mostly aimed at real time graphics, rather than an alternative to offline rendering methods such as raytracing, we will focus on this aspect. Since a comparable deep learning approach does not exist, the discussion is limited to comparing it to traditional methods.

5.1 Advantages

One notable feature of this approach is the fact that it operates fully in 3D and regresses shading by taking the entire scene into account. This is similar to raytracing and can capture details that screen space methods, as often used in real time graphics, have no access to.

Another advantage lies within the way shading is computed, it can be fully decoupled from standard rendering tasks such as direct light. This makes it possible to fill empty GPU compute slots in the rendering pipeline, similar to other compute tasks, which may be advantageous compared to state-of-the-art screen space methods. Since every aspect of their method is scalable, quality and performance can easily be adjusted to meet the needs, which is a benefit as well.

Finally, their method fully works on unseen data as well as dynamic scenes. This means a well trained model can potentially replace the need for offline computations, such as lightmap baking. Many popular GI methods, for example, rely on precalculated data, which not only takes up GPU memory but also has to be kept up-to-date and requires static scenes. Thus, the proposed method could be a good quality / limitations trade-off.

5.2 Limitations

While the approach of Hermosilla *et al.* has many potential benefits, there are also some non-negligible hurdles to overcome in order to make it practical for real time rendering.

Firstly, their method fully operates on point clouds with uniform materials that do not support specular reflections. This heavily limits its use in interactive applications, which mostly use polygon meshes instead of point clouds and require textured and reflective surfaces. The authors, too, acknowledge these shortcomings.

Another issue lies within the integratability of the network into existing frameworks. Where methods such as *Deep Shading* [Nalbach et al. 2017] operate on deferred shading buffers, which are easily accessible in most deferred renderers, the discussed approach requires an uncommon input structure in the form of a point cloud containing the entire scene. This requires one to first uniformly sample the entire scene and then, ideally, feed the point cloud into an acceleration structure, such as the voxel grid the authors use. This means, that integrating the model into existing frameworks may be very difficult and the required pre-processing of the scene could potentially make it slow and unpractical.

Lastly, the performance of the network when regressing shading is also problematic. While the results are better than screen space methods, the time requirements are not. All network-based shading effects perform much worse than their screen space counterparts, taking 150% longer on average. GI alone takes 107.6ms, which is too slow for most real time applications. Any effect that takes longer than roughly 42ms can not be run at the minimum 24 frames per second required for convincing interactivity. With recent

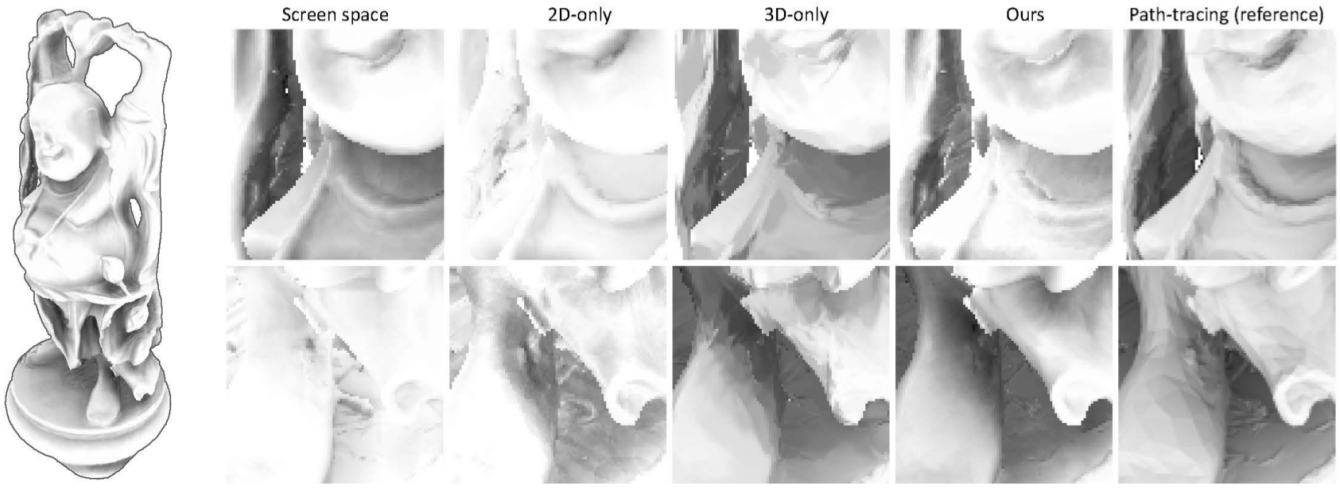

Fig. 5. Examples for the AO results output by the various methods used for the evaluation, as seen in [Hermosilla et al. 2019].

advances in real time raytracing [NVIDIA 2021b], it is also questionable whether the proposed method can outperform equivalent traditional methods, both in quality as well as performance.

6 RECENT WORK

Before providing some ideas for future work, a couple of related works are presented for comparison. The aforementioned *Deep Shading* network [Nalbach et al. 2017] also targets real time rendering, but uses CNNs on deferred rendering buffers to regress shading. This, while being faster and easier to integrate, is limited by being essentially a screen effect.

A more recent work [Sanzenbacher et al. 2020] performs rendering entirely based on neural networks, instead of a selection of shading effects. This work also has not only a comparable 3D & 2D step, but works on the basis of point clouds. They use PointNet instead of MCCs, however, which performs worse [Hermosilla et al. 2018].

Lastly, Vicini *et al.* teach a neural network SSS sampling and achieve impressively accurate results [Vicini et al. 2019]. Their method is limited to SSS and can also be seen an alternative to path tracing, rather than being aimed at real time rendering.

7 CONCLUSION & FUTURE WORK

In this report, we presented a method for learning the latent space of light transport, as introduced by [Hermosilla et al. 2019]. We showed how the most basic building block of the architecture, the Monte Carlo convolution, enables learning in 3D space and how structured 2D information can be regressed from it. Finally, we provided an extended discussion, detailing several benefits and limitations of the proposed method.

To round out the report, we also want to introduce some ideas for future work and possible extensions of the work. Firstly, comparing the model to state-of-the-art, hardware accelerated real time raytracing could provide valuable insight into the practicality of the approach in modern, high-fidelity interactive applications. While screen space methods are definitely still common, it no longer reflects the current high-end methods. Secondly, it would be interesting to see the model used in a production renderer. While the very limited rendering pipeline is sufficient for testing the model, testing how well the model can generalize is not nearly as feasible.

Finally, implementing shading using dedicated hardware could be very beneficial, similar to how Tensor Cores can be used for supersampling [NVIDIA 2021a]. Accelerating the discussed shading effects with sparingly used hardware could both speed up regression drastically and free up resources previously used to compute equivalent effects in screen space. Using all capabilities of modern GPUs effectively is certainly the best way to increase rendering quality and could make the method practical, as well as interesting for real time applications.

REFERENCES

Tomas Akenine-Mller, Eric Haines, and Naty Hoffman. 2018. *Real-Time Rendering, Fourth Edition* (4th ed.). A. K. Peters, Ltd., USA.

Pedro Hermosilla, Sebastian Maisch, Tobias Ritschel, and Timo Ropinski. 2019. Deep-learning the Latent Space of Light Transport. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 207–217.

Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. 2018. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–12.

Malvin H Kalos and Paula A Whitlock. 2009. *Monte carlo methods*. John Wiley & Sons.

Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, H-P Seidel, and Tobias Ritschel. 2017. Deep shading: convolutional neural networks for screen space shading. In *Computer graphics forum*, Vol. 36. Wiley Online Library, 65–78.

NVIDIA. 2021a. NVIDIA deep learning super-sampling (DLSS). (2021). https://developer.nvidia.com/dlss (accessed 31.10.2021).

NVIDIA. 2021b. NVIDIA RTX real time raytracing overview. (2021). https://developer.nvidia.com/rtx/raytracing (accessed 31.10.2021).

Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413* (2017).

Paul Sanzenbacher, Lars Mescheder, and Andreas Geiger. 2020. Learning Neural Light Transport. *arXiv preprint arXiv:2006.03427* (2020).

Delio Vicini, Vladlen Koltun, and Wenzel Jakob. 2019. A learned shape-adaptive subsurface scattering model. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–15.