# Neural Shadow Mapping - Supplemental

SAYANTAN DATTA, McGill University, Canada and Meta Reality Labs, USA
DEREK NOWROUZEZAHRAI, McGill University, Canada
CHRISTOPH SCHIED AND ZHAO DONG, Meta Reality Labs, USA

**ACM Reference Format:**
Sayantan Datta, Derek Nowrouzezahrai, and Christoph Schied and Zhao Dong. 2022. Neural Shadow Mapping - Supplemental. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings (SIGGRAPH '22 Conference Proceedings), August 7–11, 2022, Vancouver, BC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3528233.3530700

## 1 FEATURE ENGINEERING

*1.0.1 Overfitting and underfitting.* Neural networks are known for their ability to fit a dataset, even when there is no underlying relationship between the features and the target - a phenomenon known as overfitting. An extreme example is mapping our ray-traced targets to white noise textures as features. In this case, the neural network acts as a hash-map, mapping input to the output. This might work for a very small dataset but will obviously fail for any larger dataset. In practice, some features provide crucial information and some do not. Thus, to isolate noise from information, we need a diverse dataset. In our case, we can create diversity by collecting data across various emitter and camera positions, and scenes. In fact, as we generate more training data, we run into the regime of underfitting, where the network lacks enough capacity to learn the details in the dataset.

*1.0.2 Feature selection.* As described in the main paper, we use *sensitivity* as our metric to select and prune our features from the set $U = \{d, \mathbf{n}, z, \mathbf{n_e}, z_f, c_e, c_c\} + \{z - z_f, z/z_f, c_c/d, \mathbf{n} \cdot \mathbf{n_e}\}$. However, pruning based on sensitivity alone does not lead to a unique combination of buffers. There are sever combinations with similar sensitivity. We use validation error as tie-breaking rule in such scenario. For example as shown in table 1, buffer combinations $\{z/z_f, z - z_f, c_e, c_c/d\}$ and $\{z/z_f, z - z_f, \mathbf{n} \cdot \mathbf{n}, d\}$ have similar sensitivity but one has lower validation error.


*SIGGRAPH '22 Conference Proceedings, August 07–11, 2022, Vancouver, BC, Canada*

ACM ISBN 978-1-4503-9337-9/22/08…$15.00
https://doi.org/10.1145/3528233.3530700

Table 1. Validation error for various combination of buffers. At each row, we add 2 new features and remove features with sensitivity below 1.5%. +/− indicates set addition and subtraction.

| Before pruning | | | After pruning | | |
|---|---|---|---|---|---|
| Fea. Count | Feature Description | Err $10^{-3}$ | Fea. Count | Feature Description | Err $10^{-3}$ |
| 2 | $\{z/z_f, c_e\}$ | 6.86 | No change | | |
| 4 | $+\{z - z_f, c_c/d\}$ | 6.65 | No change | | |
| 6 | $+\{\mathbf{n} \cdot \mathbf{n_e}, d\}$ | 6.61 | 4 | $-\{\mathbf{n} \cdot \mathbf{n_e}, c_c/d\}$ | 6.69 |

## 2 TEMPORAL LOSS

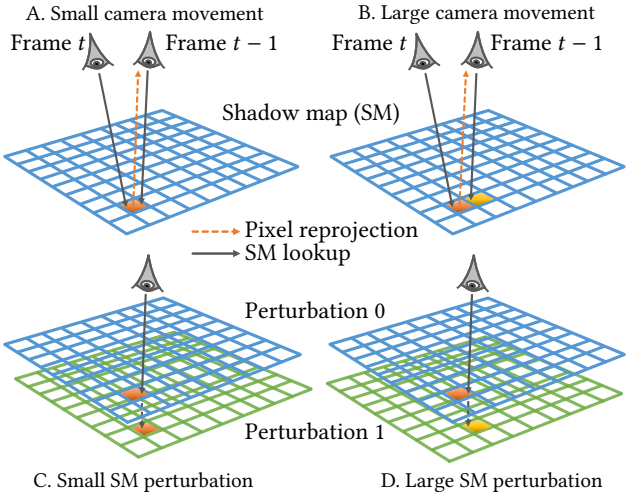


Fig. 1. Comparing the effect of motion-vector and perturbation loss on shadow map texture lookup. A and C indicate that a small camera movement in time is equivalent to a small perturbation of SM in space. Similarly, B and D indicate that a large camera movement is same as large perturbation of SM in space.

In this section, we compare the similarities an dissimilarities between our perturbation loss with motion vector [7] based loss function. A motion-vector finds a pixel's location in the previous frame by projecting the pixel's world-space position on the previous frame. A motion-vector based loss thus computes the motion-vector compensated temporal pixel difference in the network output as error which is then backpropagated through the network for learning. Note that a motion-vector based loss uses historical data **only** during training.

We argue that our perturbation loss has similar end effect as motion vector based loss while offering more control over temporal stability and simplicity in data collection. In case of motion vector loss, due to discrete nature of pixels, the reprojected pixel in the previous frame may correspond to a slightly different world-space location. Assuming the emitter is fixed, a small difference in world space value of a pixel across time may result in different shadow-map texture lookup, as shown in figure 1 (b). This is equivalent to perturbing the emitter (our approach) while keeping the camera position fixed as shown in (d). However, for motion-vector loss, if

Table 2. Layer-wise compute, learnable parameters, and temporary storage read/write access for a 5-layer network processing 1024x2048 resolution inputs.

| # | runtime(ms)/ resolution | Compute (GFlops) | # parameters ($\times 10^3$) | Temp. storage IO (MPixels) |
|---|---|---|---|---|
| 0 | 11.0ms (1Kx2K) | 7.15 | 3.4 | 278.9 |
| 1 | 3.67ms (512x1K) | 8.05 | 15.4 | 117.4 |
| 2 | 1.63ms (256x512) | 8.05 | 61.4 | 58.7 |
| 3 | 0.79ms (128x256) | 8.05 | 245.8 | 29.3 |
| 4 | 0.17ms (64x128) | 2.68 | 327.68 | 6.29 |

Table 3. Measured layer-wise compute time for a 5-layer network processing 1024x2048 resolution input.

| layer/ Resolution | Encoder Op/Time(ms) | Decoder Op/Time(ms) | Total (ms) |
|---|---|---|---|
| 0 (1Kx2K) | Conv2d + DnSamp (4.27 + 0.72) | UpSamp + Skip + Conv2d (0.24 + 0.37 + 5.37) | 11.0 |
| 1 (512x1K) | Conv2d + DnSamp (1.38 + 0.63) | UpSamp + Skip + Conv2d (0.12 + 0.19 + 1.36) | 3.67 |
| 2 (256x512) | Conv2d + DnSamp (0.58 + 0.33) | UpSamp + Skip + Conv2d (0.07 + 0.10 + 0.55) | 1.63 |
| 3 (128x256) | Conv2d + DnSamp (0.29 + 0.10) | UpSamp + Skip + Conv2d (0.04 + 0.05 + 0.31) | 0.79 |
| 4 (64x128) | Conv2d (0.17) | | 0.17 |

the framerate is too high, the world-space difference in pixel may be small. The pixel and its reprojection may share the same shadow map texel, as shown in (a). Same is true if the shadow map texels are large. In both cases, the temporal error is zero and the network does not learn. Conversely, if the framerate is too low, we may not find a valid reprojection and the error must be forced to zero. As such, a careful balance of the framerate is required for motion-vector losses. However, in our case, we can directly adjust the emitter perturbation such that the differences are large enough for non-zero backpropagation as shown in figure (d). This is achieved by by setting the perturbation magnitude proportional to the distance of emitter from the scene and size of the emitter. Additionally, motion-vector losses rely on long sequence of key-framed images which adds additional complexity to data collection pipeline. Our approach do not rely on temporal information, as such we can sample the scene with arbitrary camera, emitter and object trajectories with any desired framerate.

## 3 PERFORMANCE OPTIMIZATION

We discuss an interesting aspect of UNet architecture that we did not discuss in the main document.

From table 2, we see that the runtime performance of our network is proportional to the temporary storage, not the number of compute operations. Notice how the compute flops are nearly constant for the first 4 layers, yet the runtime drops as we go down the layers. This indicates the performance is bounded by memory bandwidth. Required memory bandwidth is proportional to the size of temporary buffers and how well the buffers are cached. Caching is most effective when the size of the temporary buffers are small. In fact due to higher cache misses in the layer 0, we see a more than linear growth in runtime compared to layer 1. We verified this using a profiler. A detailed layerwise breakdow of the network runtime (without optimization) is provided in table 2.

Further performance improvements may be possible through pruning [3] of the network weights. Performance of our network largely depends on the size of temporary buffers; not how the buffers are connected. As such, simply pruning the weights may not improve performance unless we also reduce the size of temporary storage. Another avenue for exploration is lowering the precision of temporary storage to 8 bits as the network output is always between 0 and 1. We leave these optimizations for further exploration as future work.

## 4 NETWORK DEPTH OPTIMIZATION

In this section, we first derive the mathematical formula for penumbra width using our simplified model as described in the main paper. We estimate the parameters $\theta$, and $\theta_\delta$ as follows:

$$z_m = PM = \frac{z_{max} + z_{min}}{2} \tag{1}$$

$$r_s = MD = \frac{z_{max} - z_{min}}{2} \tag{2}$$

From $\triangle AMP$,

$$AM = \sqrt{z_m^2 + r_e^2} \tag{3}$$

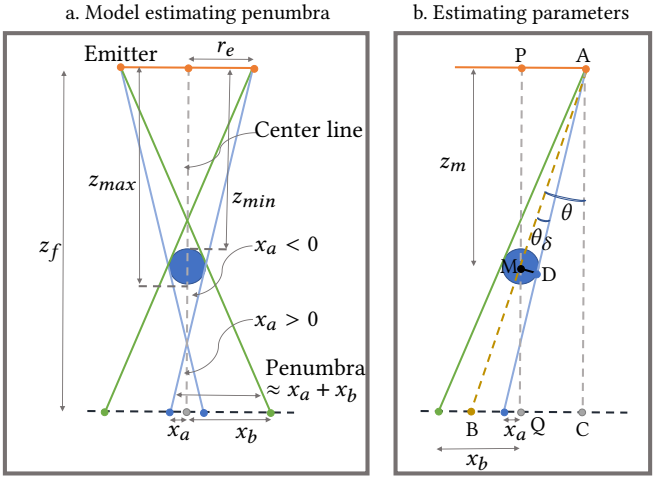


Fig. 2. A simplified model to estimate the penumbra size at a given pixel. We assume our occluder is spherical in shape, forming a convex bounding-sphere around the occlusion geometry as shown in figure (a) on the left. Figure (b) shows a simplified diagram to estimate the parameters $\theta, \theta_\delta$.

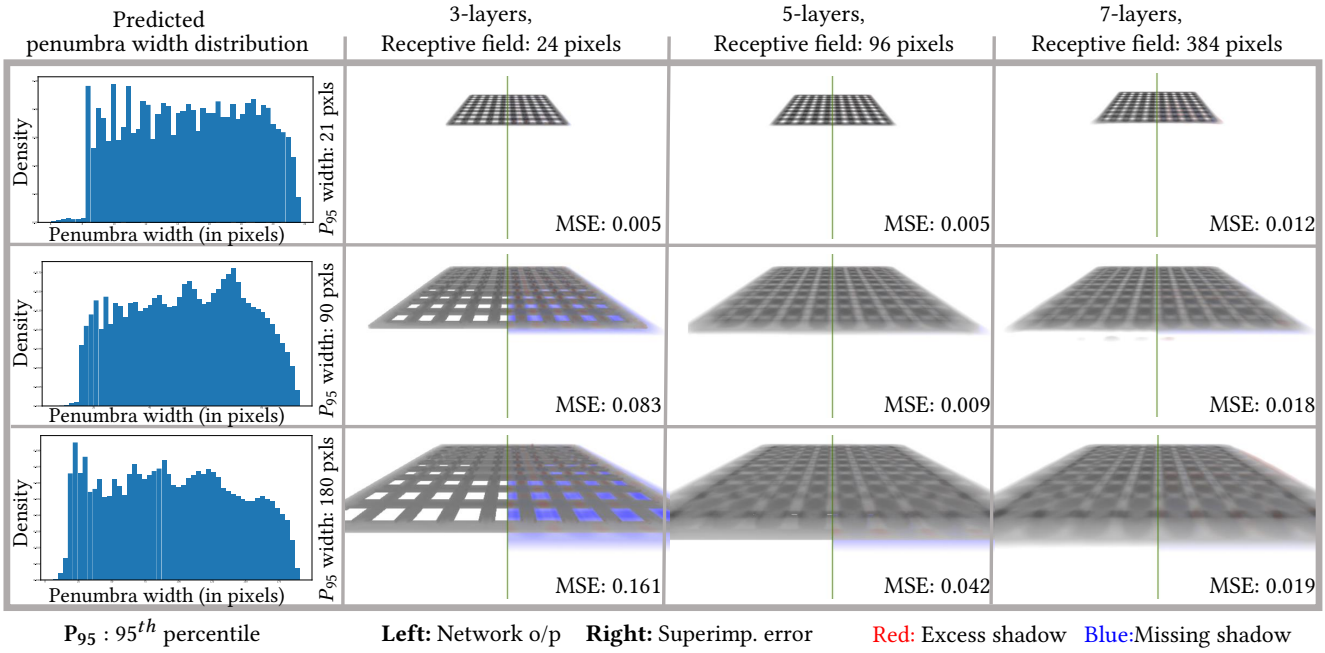


Fig. 3. Empirical validation of our penumbra width prediction model. We predict the penumbra width on the vertical axis and vary the number of layers on the horizontal axis. The diagonal represents the optimal number of layers and elements below the diagonal show high error as the receptive field is not large enough to accommodate the penumbra width.

From $\triangle AMD$,

$$\theta_\delta = sin^{-1}\frac{MD}{AM} = sin^{-1}\frac{r_s}{\sqrt{z_m^2 + r_e^2}} \tag{4}$$

From $\triangle AMP$ and $\triangle BMQ$,

$$BQ = \frac{AP \cdot MQ}{MP} = \frac{r_e(z_f - z_m)}{z_m} \tag{5}$$

From $\triangle ABC$,

$$\theta = tan^{-1}\frac{BQ + QC}{AC} = tan^{-1}\frac{BQ + r_e}{z_f} \tag{6}$$

Therefore,

$$x_a = z_f\, tan(\theta - \theta_\delta) - r_e \tag{7}$$

$$x_b = z_f\, tan(\theta + \theta_\delta) - r_e. \tag{8}$$

## 4.1 Empirical verification

In figure 3, we move a mesh object between a ground plane and an emitter producing penumbra of varying size. For each row in the figure, we predict the distribution of penumbra size and find the maximum (95th percentile) width from the distribution. Across the columns, we vary the receptive field of the network by adjusting the number of layers. Collecting the errors across all combinations, we notice that the elements below the diagonal have high error as the receptive field of the network is not large enough to accommodate the penumbra size. Note that in figure 3, networks do not have the same compute flops per pixel as shown in table 4.

Simply increasing the number of layers in a network also increases the compute cost. As such, any reduction in error might be attributed to the increased compute requirement. We verify that the reduced error is indeed due to increased receptive field of the network and not necessarily due to the increased computation. As such, we constrain the network flops across the 3, 5, and 7 layer network as shown in table 5. From figure 4, we see that the error reduces

Table 4. Table showing the compute cost and the number of learnable parameters for 3,5,and 7 layer network in figure 3.

| Network | Flops per pixel | # learnable parameters |
|---|---|---|
| 3-layer | 8528 | 39.25K |
| 5-layer | 16208 | 653.7K |
| 7-layer | 21968 | 9.501M |

Table 5. Table showing the compute cost and the number of learnable parameters for 3,5,and 7 layer network shown in figure 4.

| Network | Flops per pixel | # learnable parameters |
|---|---|---|
| 3-layer | 14928 | 110.9K |
| 5-layer | 16208 | 653.7K |
| 7-layer | 15328 | 1.626M |

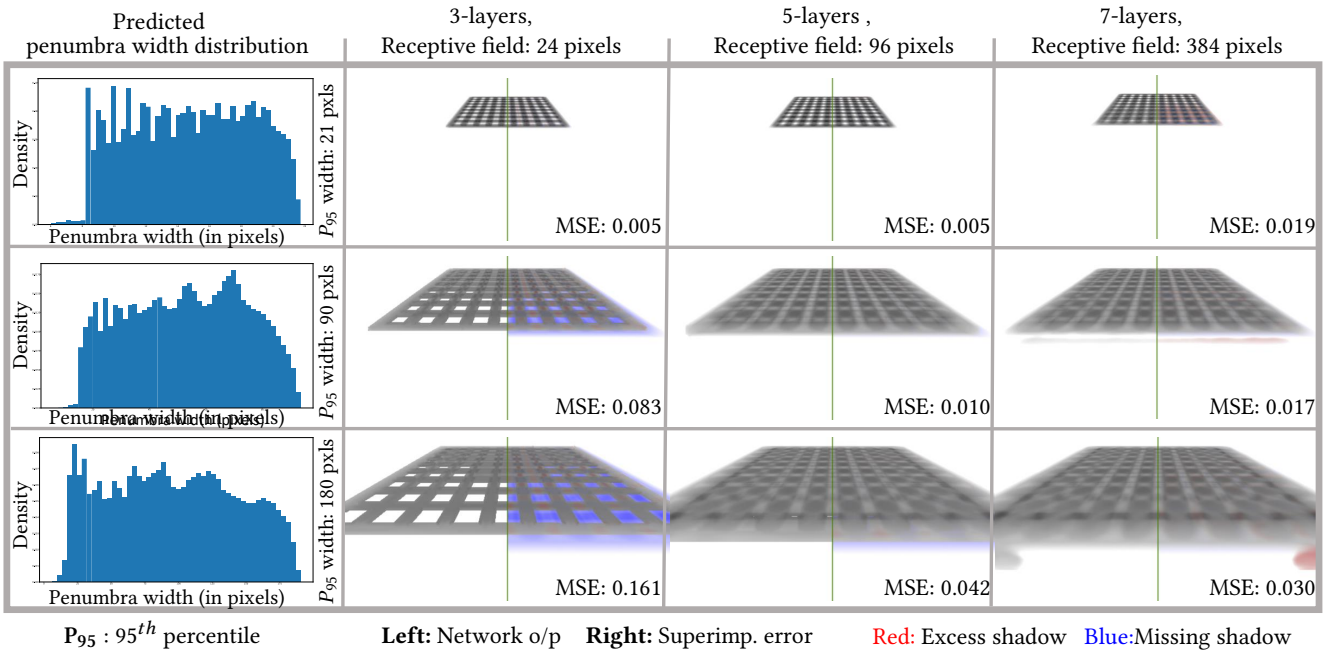


Fig. 4. Empirical validation of our penumbra width prediction model. We predict the penumbra width on the vertical axis and vary the number of layers on the horizontal axis while keeping the compute flops constant across networks. The flops constrained networks show similar behavior as the unconstrained networks in figure 3.

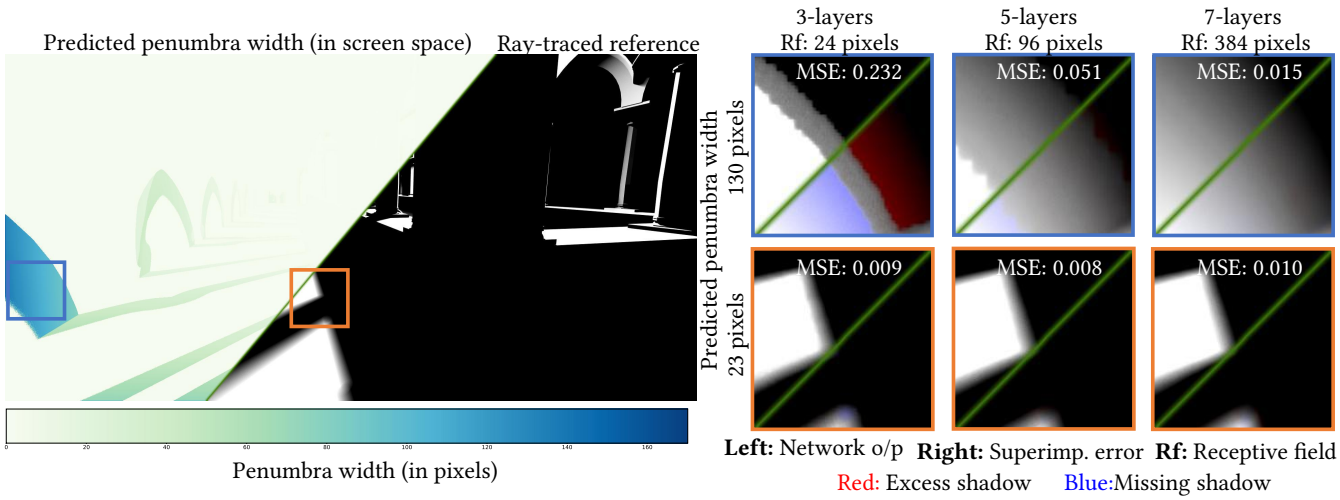


Fig. 5. Figure showing the accuracy of our penumbra width prediction model on Sponza. Region with high penumbra width (blue cutout), as predicted by our model is reproduced accurately only by the 7-layer network.

along the diagonal despite using networks that are computationally equal (almost).

In figure 5, we take a single frame from the Sponza scene and analyze the validity of our penumbra width prediction across different regions in the frame. The increasing accuracy of our network output with the number of layers in a region with large penumbra (yellow cutout) is a strong validation in support of our model.

## 5 DATA GENERATION, TRAINING AND INFERENCE

Orchestrating scene authoring, data collection, training, inference, and generating the final results is one of the key challenges to

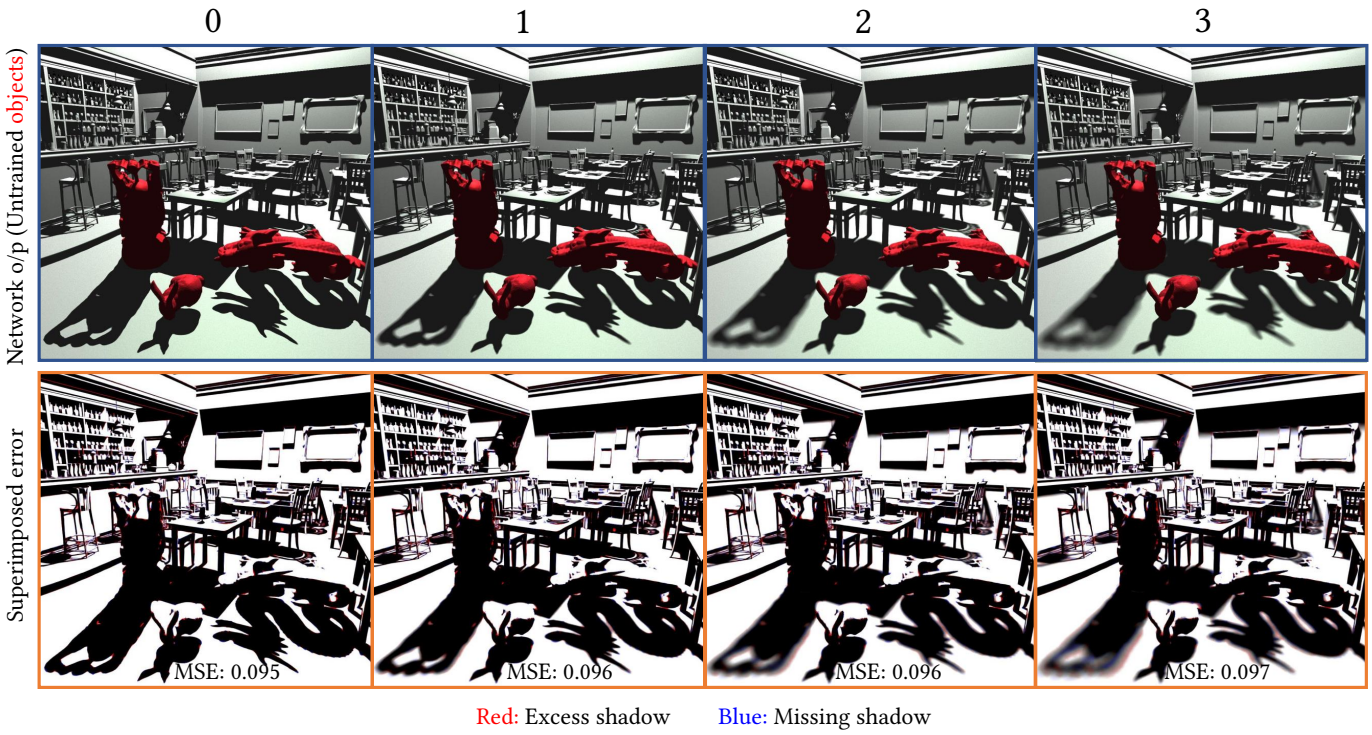


Fig. 6. Demonstration of the effectiveness of our technique on untrained objects marked in red in the first row. Second row shows the comparison w.r.t reference with error superimposed over raw network output. Going from left to right, hard-shadow is indexed with 0 while 3 indicates an emitter of diameter 50cm. Shading is applied post-process in the first row.

this project. We author the scenes using Blender [2], ensuring all keyframed camera trajectories do not go outside the emitter frustum. We keyframe several camera and emitter trajectories covering different scenarios and bake them into the scenes. We then upload the scenes to a Linux cluster to generate our training data. A modified version Falcor [4] based on Vulkan (originally DX12) is used along with Python scripts to collect, process and organize the data for training. We use Pytorch [6] scripts for training which scans through the data in a random order every epoch. While training, we save $N(= 3)$ best models based on a test error. The test consist of a small number (10-15) of handpicked images. We run the test with a probability of 0.01 at each training iteration. Thus the test is run roughly every 100 training iterations. Once the training is complete, we select the best model out of $N$ by comparing the error on full training dataset. We also collect several statistics during the training for analysis. During inference, we run our model through new trajectories which may have some overlap with training dataset but are not same. We test the performance of our network on a local machine (AMD 5600X, Nvidia 2080Ti) using Falcor and a Cudnn based solution.

The training data is generated using concurrent shadow mapping and ray-tracing passes. The shadow mapping pass outputs three perturbations of the buffers, obtained by jittering the emitter and camera positions. The ray-tracing pass uses 1500 rays per pixel distributed across 8 or more accumulation passes enabling 8x or more MSAA. The ray-tracing pass simultaneously outputs shadows with 4 different levels of softness. To summarize, each frame consist of 3 perturbations of our input buffers and 4 antialiased ray-traced images with increasing softness.

A visual inspection of the generated data is crucial. We compare the unprocessed rasterized shadows with the ray-traced shadows. A match between rasterized and ray-traced output is desired and the two should have minimal and consistent bias, if any. For example, there may be a difference in the position of shadows between rasterization and ray-tracing due to the offset used for preventing self-intersection. We find rescaling the scenes to the same dimension useful for minimizing bias between rasterization and ray-tracing and generating consistent data. Bias also depend on how we generate the rays - emitter to scene or vice versa. The two may be different depending on how backface culling is configured. We prefer emitter to scene ray-tracing as the setup is closer to shadow-mapping.

## 6 NETWORK ARCHITECTURE ABLATION STUDY

We discuss the limitations of various other network architecture we implemented. The first variation is inspired by Exponential Shadow Maps [1], where we learn the parameter $\alpha$ in the depth test approximation $e^{\alpha(z-z_f)}$. To remove non-linearity in the final network layer, we perform the Adam-SGD optimization in log-space and exponentiate the output during inference. However, injecting unprocessed input $z$ directly to the output causes severe shadow-aliasing and

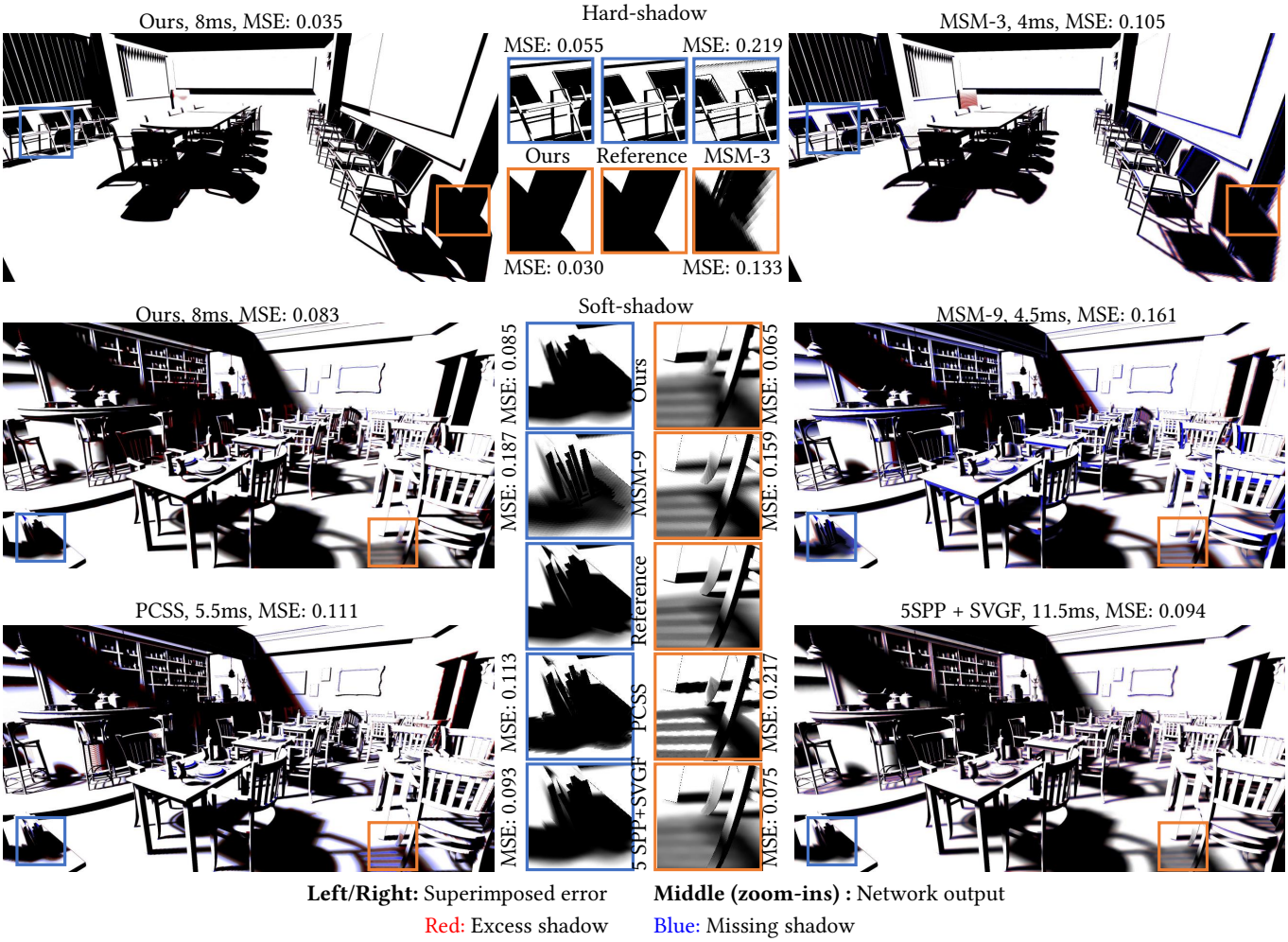


Fig. 7. Our network's ability to generalize to unseen trajectories (within the same scene) compared to output from competing techniques (MSM, PCSS, Raytracing+Denoising) for hard and soft shadows. MSM-3 and MSM-9 are Moment Shadow Map variants using 3×3 and 9×9 prefiltering kernels. Conference model ©Anat Grynberg and Greg Ward.

our loss function (VGG-19) is ineffective in correcting the aliasing in log-space. In another variation, we tried denoising the rasterized (from emitter) depth - $z$ using a separate network with ray-traced depths as targets. We split the network in two halves - first half for denosing the depth and second half for processing the output of the first half into final shadows. We trained the two network end to end. However, the architecture failed to generate good quality final output compared to our vanilla network. We also tested other variants of the same idea but were equally ineffective.

## 7 RESULTS AND COMPARISONS

Figure 7 compares our technique with other competing techniques on trajectories that were not present in the training. Figure 8 shows the application of our technique across untrained camera and emitter trajectories. For each scene, we train a separate network using a variety of emitter and camera configurations across a range of softness. As such, a single network can generate both hard and soft shadows where the softness is controlled using a scalar input. Figure 6 shows that our network generalizes across variety of shapes that were not present in the training set.

## 8 LIMITATIONS AND FUTURE WORK

One of the current limitations of our technique is that it does not naturally extend beyond a single light source. Resolving this is an exciting avenue for future work. Another limitation is it does not generalize well across a *mixture of scenes* with widely different emitter depth distributions. This is primarily due to the (purposefully) compact size of our networks and their limited capacity to fit such multi-modal data. To avoid this, one could potentially train multiple networks across different depth variations and swap between them networks (i.e., based on emitter distance) at runtime. Artifacts also arise when the penumbra sizes exceed the receptive field of the neural network and training tile size; our conservative penumbra

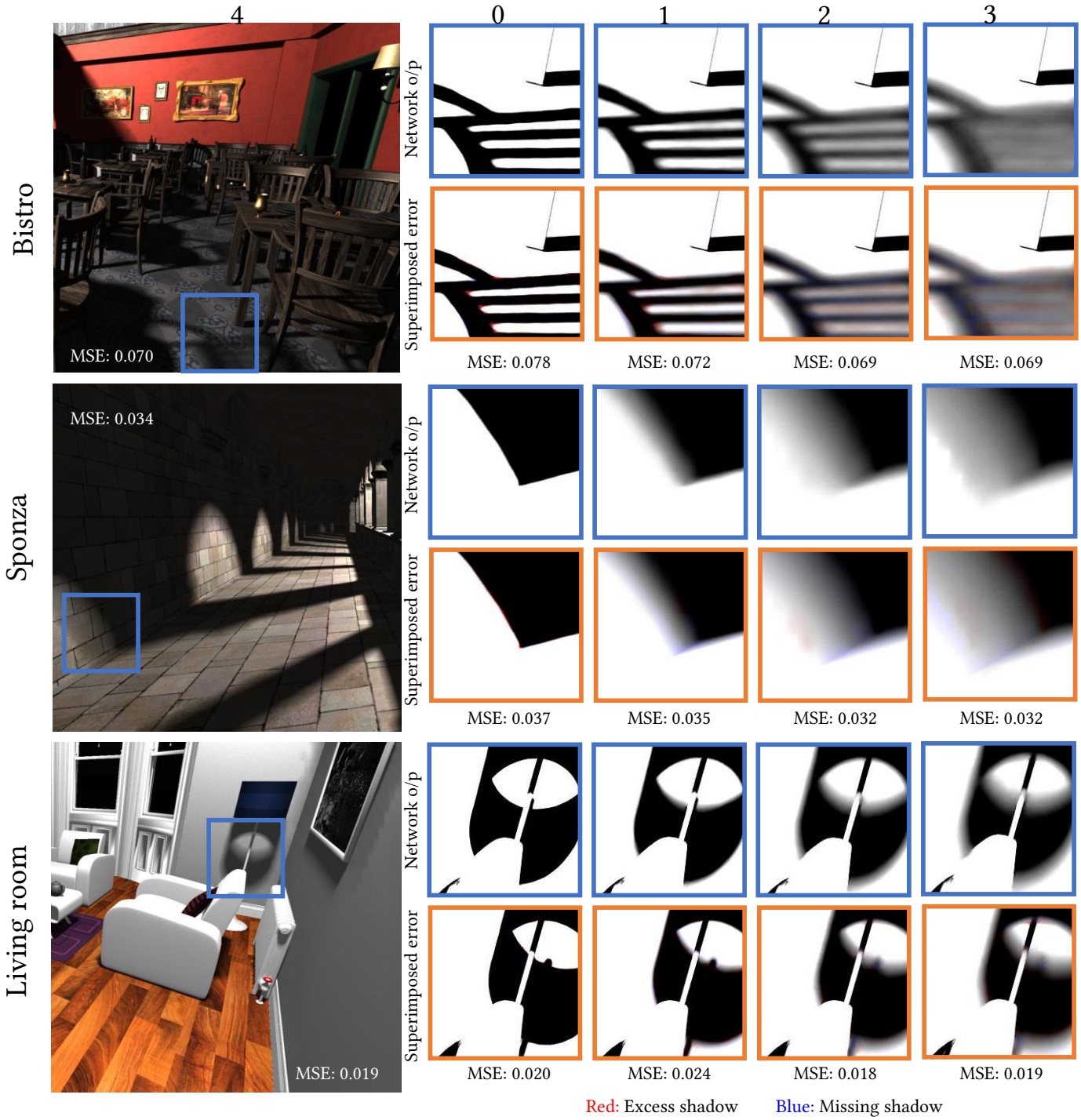


Fig. 8. Figure shows the output of our network (shaded post-process) with varying penumbra sizes in the cutouts (unshaded). Each scene is trained independently and tested on a validation set. The second row in each scene shows the error (superimposed on network oputput) w.r.t. reference. Hard-shadow is indexed 0 while 4 indicates an emitter of diameter 50cm.

size estimate affords us the opportunity, however, to tailor our architecture's receptive field accordingly. Finally, our perturbation loss can sometimes overblur fine geometric details, such as from

thin features like wires, however the gains in temporal stability are significant; fine tuning here can help to mitigate such overblurring.

An exciting avenue for future work involves extending our technique to a finite number of light sources. We can draw inspiration from graph-coloring approaches [5] here, where non-overlapping emitter frustums are grouped together in layers and each layer is filtered independently. A neural approach could use a network that takes as input buffers from $n$ sources and outputs the result. During training, we can disable (i.e., zero input) a fraction of the light sources $p$ ($< n$) to permit the network to more effectively learn the response from each source, i.e., without confounding the effects *between* sources.

Finally, runtime optimization of our network is another avenue for exploration. The performance of a UNet depends primarily on memory bandwidth, as opposed to compute operations. A customized pruning technique focused on improving the compute density by reducing the number of temporary buffers is an interesting direction of research. Also the network output is non-HDR, which can be exploited to reduce the size of buffers.

## REFERENCES


[1] Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. 2008. Exponential Shadow Maps. In *Proceedings of Graphics Interface 2008* (Windsor, Ontario, Canada) *(GI '08)*. Canadian Information Processing Society, CAN, 155–161.

[2] Blender Online Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www.blender.org

[3] Jonathan Frankle and Michael Carbin. 2018. The Lottery Ticket Hypothesis: Training Pruned Neural Networks. *CoRR* abs/1803.03635 (2018). arXiv:1803.03635 http://arxiv.org/abs/1803.03635

[4] Simon Kallweit, Petrik Clarberg, Craig Kolb, Kai-Hwa Yao, Theresa Foley, Lifan Wu, Lucy Chen, Tomas Akenine-Moller, Chris Wyman, Cyril Crassin, and Nir Benty. 2021. The Falcor Rendering Framework. https://github.com/NVIDIAGameWorks/Falcor https://github.com/NVIDIAGameWorks/Falcor.

[5] Viktor Heisenberg Marton Tamas. 2018. *Practical Screen Space Soft Shadow, GPU Pro 360 Guide to Shadows* (1st ed.). A. K. Peters, Ltd., USA.

[6] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[7] Lei Xiao, Salah Nouri, Matt Chapman, Alexander Fix, Douglas Lanman, and Anton Kaplanyan. 2020. Neural Supersampling for Real-Time Rendering. *ACM Trans. Graph.* 39, 4, Article 142 (July 2020), 12 pages. https://doi.org/10.1145/3386569.3392376