Efficient Graphics Representation with Differentiable Indirection - Supplemental

Sayantan Datta
McGill Univeristy
Canada
Meta Reality Labs
USA

Carl Marshall
Zhao Dong
Zhengqin Li
Meta Reality Labs
USA

Derek Nowrouzezahrai
McGill University
Canada

ACM Reference Format:
Sayantan Datta, Carl Marshall, Zhao Dong, Zhengqin Li, and Derek Nowrouzezahrai. 2023. Efficient Graphics Representation with Differentiable Indirection - Supplemental. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 IMPLEMENTATION DETAILS

This section provides additional implementation details for each task.

1.1 Isotropic GGX approximation

For isotropic GGX training, we use a cascade of two 2D arrays. The primary array is initialized with an undistorted uv-rmap while the cascaded array with a constant 0.5. No non-linearity is added to the output of the cascaded array. For isotropic GGX, we set $\rho = 2$. As described in the main document, ρ is the ratio of he ratio of the length of one side of the primary array to the cascaded array. Thus, the primary array is 16x16 and the cascaded array is 8x8 resolution. We obtain training samples from appropriate distributions for the two inputs; h_z sampled from cosine-hemisphere distribution and α_h from an exponential-like distribution with more samples biased towards lower roughness values. Output of the network is compared with the reference analytic output and the network is trained in a local coordinate frame with normal pointing at (0,0,1).

1.2 Disney BRDF approximation

For training, we sample the view and emitter directions from uniform hemisphere sampling, the metallic-roughness from exponential-like distribution and rest of the artist control parameters and albedo from a uniform random distribution. We learn in a local coordinate frame with normal pointing at (0,0,1).

1.2.1 Flops calculation. The reference Disney BRDF uses 92 additions, 150 multiplications, 17 divisions, 5 square-roots, and 1 logarithm. Our implementation requires 11 additions, 26 multiplications, and 2 divisions. We also require 5 lookups as detailed below in table 1. We assume additions and multiplications require 1 flop each, divisions 2 flops, square root and logarithms 4 flops.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Brief description of various lookups associated with Disney BRDF approximation.

Dimension/ Interpolation	O/P Channels	Associated Output
16x16 / Bilinear	3	Anisotropic GGX
16x16 / Bilinear	1	Smith Masking
16x16 / Bilinear	4	Disney Diffuse, Clear coat masking
16x16 / Bilinear	4	Metallic, Clear coat, Sheen
16 / Linear	2	Clear coat gloss

We note that our implementation is not unique but one of several possible implementations using *differentiable indirection*. However, our factorization yields good results close to reference with performance advantage even on unoptimized hardware. Note the total array size is 6KB (half-precision) and 12KB (full-precision) and easily stored in specialized on-chip memory.

1.3 Compact Image Representation

Setup: For the image representation task, we use a 2D array with 4 channels as the primary and a 3D array with k-channels as the cascaded array, where k is the number channels in a texture or image. We initialize the primary array with uv-ramp repeated twice for 4 channels. We initialize the cascaded array with grey, white, light-blue (0.5,0.5,1), grey for albedo, ao, normal, and roughness channels respectively. Other channels are set zero. Let us assume the length of one side of the primary array is N_p and the cascaded array is N_c. The ratio of the two sides indicated by $\rho = N_p/N_c$. Some pseudo-optimal values of ρ are provided in table 2. The values are obtained by varying ρ for a given resolution and compression and selecting the one with best output quality. Our use cases have $\rho \in [40, 128]$. A safe default value at 4K texture resolution is $\rho = 96$ and at 1K or 2K is $\rho = 64$ for 12x compression ratios.

A general trend is optimal value of ρ is proportional to the redundancies in a texture. Usually, natural images and high resolution images have more redundant pixels and requires a higher ρ. The

Table 2: Empirically obtained optimal resolution ratios (N_p/N_c) for varying compression of 4K RGB-textures.

	Compression Ratio			
	3x	6x	12x	24x
RGB	128	128	80	72

Table 3: Network configuration details for texure sampler.

Table Name	Shape/Resolution	Input	o/p channels
Primary-0	$N_{p0} \times N_{p0}$	uv-coords	3
Primary-1	N_{p1}	Pixel-footprint	1
Cascaded	$N_{lod} \times N_c^3$	Primary-1 Primary-0	k

optimal ρ also varies inversely with the required compression ratios. Note that using the defaults values under most circumstances produce results within 5% from optimal.

To set up the resolution of the primary and cascaded array, we use three information - uncompressed size of the texture in bytes denoted by B, expected compression denoted by e, and ρ. The size of our representation in bytes is given by:

$$\begin{aligned} B_{comp} &= 4N_p^2 + kN_c^4 \\ &= 4\rho^2 N_c^2 + kN_c^4, \text{ using } \rho = \frac{N_p}{N_c}. \end{aligned} \tag{1}$$

Note that $e = B/B_{comp}$; thus using equation 1, we can solve for N_c. However, we add some additional constraints. We require N_p, and N_c are integer multiples of 8, and 4 respectively to avoid memory alignment issues. Thus instead of directly solving the variable, we do a linear search in N_c that minimizes the difference $e - B/B_{comp}$ while also satisfying all constraints.

Training and inference: We train the network using uv-color pairs and use stratified random sampling to generate the training uv coordinates. Each strata corresponds to a texel in the base texture. We obtain the target color values using a bi-linear sampler. We tested with VGG-19 [1] and SSIM losses which require 2D patches of texels to perform convolutions. However, the convolutional losses did not significantly impact quality but slows the training. We use MAE as loss and ADAM optimizer with 0.001 learning rate. For inference, we quantize both both primary and the cascaded array to 8-bit. An interesting effect is that the cascaded array is often $\leq$ 1MB for 4K textures, which may fit in a lower tier cache.

1.4 Neural Texture sampling

Setup: Our texture sampler takes two input - a uv coordinate and an estimate of the pixel footprint. In real-time systems, the latter is computed using shader derivatives, often as part of hardware texture sampler. For inference purposes, we compute the pixel footprint as the magnitude of the cross product between *ddx*, *ddy* (in *Direct3D* terminology) of the uv coordinates. Our network consist of two primary arrays- one corresponding to the uv-input and the other corresponding to the pixel footprint input. A single cascaded array is used. The dimensionality of the arrays are provided in the table 3.

For the cascaded array, N_c is the sides attached to *Primary-0* and N_{lod} is the side attached to *Primary-1*. We first estimate N_{lod} - setting an initial value $\approx log_2(N_{base})$, where N_{base} is the base texture resolution. We fine tune N_{lod} as hyper parameter search. The pseudo-optimal values are N_{lod} = 8, 12, 12 for 1K, 2k, 4k textures respectively. We set $N_{p1} = 4N_{lod}$. The total bytes required for our representation is thus

$$\begin{aligned} B_{comp} &= 3N_{p0}^2 + N_{p1} + kN_c^3 N_{lod} \\ &= 3\rho^2 N_c^2 + N_{p1} + kN_c^3 N_{lod}, \text{ using } \rho = \frac{N_{p0}}{N_c}. \end{aligned} \tag{2}$$

Rest of the values - N_{p0}, and N_c are estimated similar to section 1.3 with pseudo-optimal values values of ρ provided in table 4.

Training: To generate the target data for training, we use a trilinear texture sampler that mimics a trilinear texture sampler in modern real-time systems such as *OpenGL* or *D3D*. Essentially, we generate the mip-pyramid and (tri-)interpolate between the levels with pixel footprint. We generate the training the uv-coordinates similar to section 1.3. We randomly sample pixel footprint values $\in [0, 1)$, where 0 corresponds to most detailed and 1 to least detailed level-of-detail in the mip-chain. We ensure half of the random samples belong to LOD-0 by generating uniform random samples and raising it to the power n, where $n = -log(N_{base})/log(p)$. In our case, $p = 0.5$ - corresponding to 50% samples in LOD-0. A theoretical proof is provided in the next section. We pass the uv and pixel footprint values to our network and the texture sampler and compare their output to train our network.

Sampling LODs for training data generation: Our goal is to generate more training samples from more detailed and less samples from less detailed LODs. One way to do so is to sample the pixel footprint values from an exponential distribution, however, exponential distribution tends to put too few samples for less detailed LODs. The samples however, are better distributed by simply raising the samples collected from a uniform distribution to the power of n. In this section, we calculate the optimal value of n. We generate the pixel footprint samples (x_f) as

$$x_f = u^n, \; u \sim U(0, 1), \tag{3}$$

where U indicates a uniform distribution. Let us assume we generate fraction $p(\neq \rho)$ of the total samples from pixel footprint value t or below. The *cumulative distribution function* of U is given as

$$\text{CDF}_U(u) = P(U \leq u) = u. \tag{4}$$

We calculate the *cumulative distribution function* of X_f as

Table 4: Empirically obtained optimal resolution ratios (N_{p0}/N_c) for varying compression of 4K RGB and material texture for combined compression and sampling.

	Compression ratio			
	3x	6x	12x	24x
RGB (3-channels)	80	64	64	48
RGB, Normal, AO (7-channels)	128	96	96	72

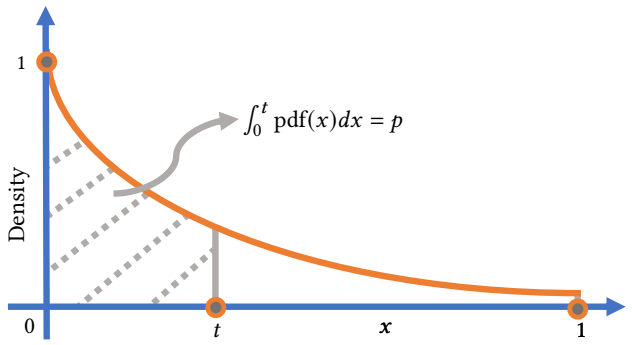

Figure 1: Figure illustrating the fraction of samples p below a threshold t as being equivalent to evaluating the *CDF* at t.

$$\begin{aligned} \text{CDF}_{X_f}(x_f) &= P(X_f \leq x_f), \text{ using eq. 3} \\ &= P(U^n \leq x_f) \\ &= P(U \leq x_f^{\frac{1}{n}}), \text{ using eq. 4} \\ &= x_f^{\frac{1}{n}}. \end{aligned} \tag{5}$$

From figure 1, note that fraction of samples p below a threshold t as being equivalent to evaluating the *CDF* at t. Therefore,

$$CDF_{X_f}(t) = t^{\frac{1}{n}} = p \text{ or } n = log_p(t). \tag{6}$$

In our application, $t = 1/N_{base}$, and $p = 0.5$.

Optional – Exponential sampling: We perform a similar analysis as last section for selecting the correct parameter (λ_e) for sampling the footprints from an exponential distribution given by $exp(-\lambda_e x)$.

$$x_f = -\frac{1}{\lambda_e} ln(u) \tag{7}$$

$$\text{CDF}_{X_f}(x_f) = 1 - e^{-\lambda \cdot x_f} \tag{8}$$

$$\lambda_e = -\frac{ln(1-p)}{t} \tag{9}$$

Even with appropriate parameters, the low resolution LODs receive too few samples, hence not used for our application.

1.5 Optimized shading pipeline

This section aims at improving the quality of rendering by making our texture sampler BRDF aware. We account for our approximate Disney BRDF when training the texture sampler. As such, we optimize with two losses. First loss compares the output of our neural texture sampler with the output of an uncompressed tri-linear filter i.e. the same loss used in the last section 1.4.

The second loss compares the output of neural texture sampler through our neural disney BRDF with the corresponding target reference. To generate the reference, we collect 16 uniformly distributed samples across the pixel footprint, and aggregate the samples post evaluation through the reference Disney Brdf; process referred as *appearance filtering* in the literature. However, to minimize aliasing at large pixel footprints without increasing sample count, we also pre-filter each individual sample with tri-linear filtering. We assume axis-aligned pixel footprint. For backpropagation, we use our pre-trained neural Disney BRDF as a differentiable fixed function layer - i.e. we freeze the contents of the pre-trained decoder arrays in our neural BRDF while allowing gradients to back-propagate through the decoder arrays.

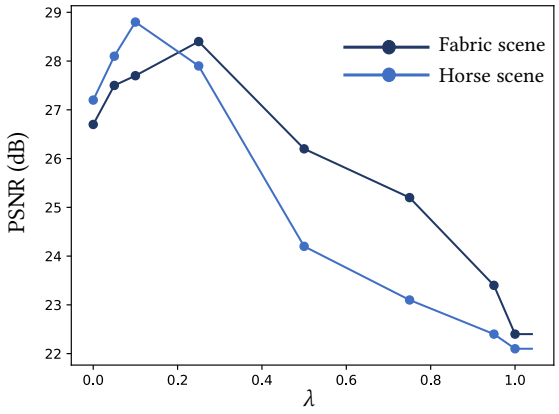

Figure 2: Plot illustrating variation in quality with increasing λ for two scenes. λ indicates the fraction of shading loss used for training as discussed in 1.5

We blend in the two losses using a hyper-parameter $\lambda \in [0, 1]$, where $\lambda = 0$ indicates purely the first loss while 1 indicates pure second loss. We notice some interesting details.

Optimizing our neural sampler directly using the shaded samples (i.e. $\lambda = 1$) results in training instability. Our test with a simple diffuse BRDF also indicates similar issues. We believe there are two main issues. First, the optimization is underconstrained i.e. different combinations of albedo, normal, AO may yield the same shaded result. Second, the unbounded non-linearities in the Disney BRDF may also cause the training to diverge. We solve the first issue by adding a regularization term, in our case we do so by setting $\lambda < 1$. Setting λ strictly less than 1 essentially uses the first loss as regularization. We fix the second issue by clipping the gradients backpropagating through the non-linear metallic component of the BRDF in ±0.1 range. A learning rate scheduler may also improve convergence in this case. A variation in quality due to increasing λ is shown in plot 2. For the *Horse scene*, and the *Fabric scene* in figure 1, 13 of the main paper shows an improvement in quality with a $\lambda = 0.1, 0.25$ respectively.

Table 5: Empirically obtained optimal resolution ratios (N_p/N_c) for varying parameter size in the SDF task.

	Parameter size					
	96MB	48MB	24MB	16MB	12MB	6MB
Piano	2.56	2.71	2.08	2.81	2.56	2

Table 6: Empirically obtained optimal resolution ratios (N_p/N_c) for varying compression ratios in NeRF task. Uncompressed density and RGB grids have a resolution of $256 \times 256 \times 256$ with 1 and 12 channels respectively.

	Compression Ratio					
	5x	10x	25x	50x	75x	100x
Density Grid (1 channel)	3.57	3.8	2.69	1.97	2.82	2.51
RGB Grid (12-channels)	5.05	3.42	4.14	4.53	3.58	5.74

1.6 Signed Distance Fields

In the SDF case, we use an Adam optimizer paired with a learning rate scheduler. Also, as described in the main paper, our training samples mostly consist of near surface samples and use an MAE loss. We quantize the distances to ±1 and learn a truncated SDF representation. For faster convergence, we also initialize the cascaded 3D-grid with an approximate SDF representation. At each voxel of the cascaded array, we use a Kd-Tree to lookup the nearest sample in the training set and set the voxel's content as the corresponding truncated distance. We notice, pre-initializing the cascaded array minimizes artifacts. The primary 3D-array is initialized as uniform ramp as usual. We set up the array resolutions similar to section 1.3 and use $\rho \in [1.5, 2.5]$ as shown in table 5. During inference we quantize both the primary and cascaded arrays to 8-bit.

1.7 Radiance field compression

Similar to section 1.6, we initialize the cascaded 3D-array with the a scaled version of the target grid extracted from the *Direct Voxel* technique. Such initialization is optional but improves training convergence. However, unlike section 1.6, we only quantize the primary array to 8-bit during inference. ρ values are provided in table 6. We use standard MAE loss and ADAM optimizer.

REFERENCES

[1] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.