Efficient Graphics Representation with Differentiable Indirection

Sayantan Datta
McGill Univeristy
Canada
Meta Reality Labs
USA

Carl Marshall
Zhao Dong
Zhengqin Li
Meta Reality Labs
USA

Derek Nowrouzezahrai
McGill University
Canada

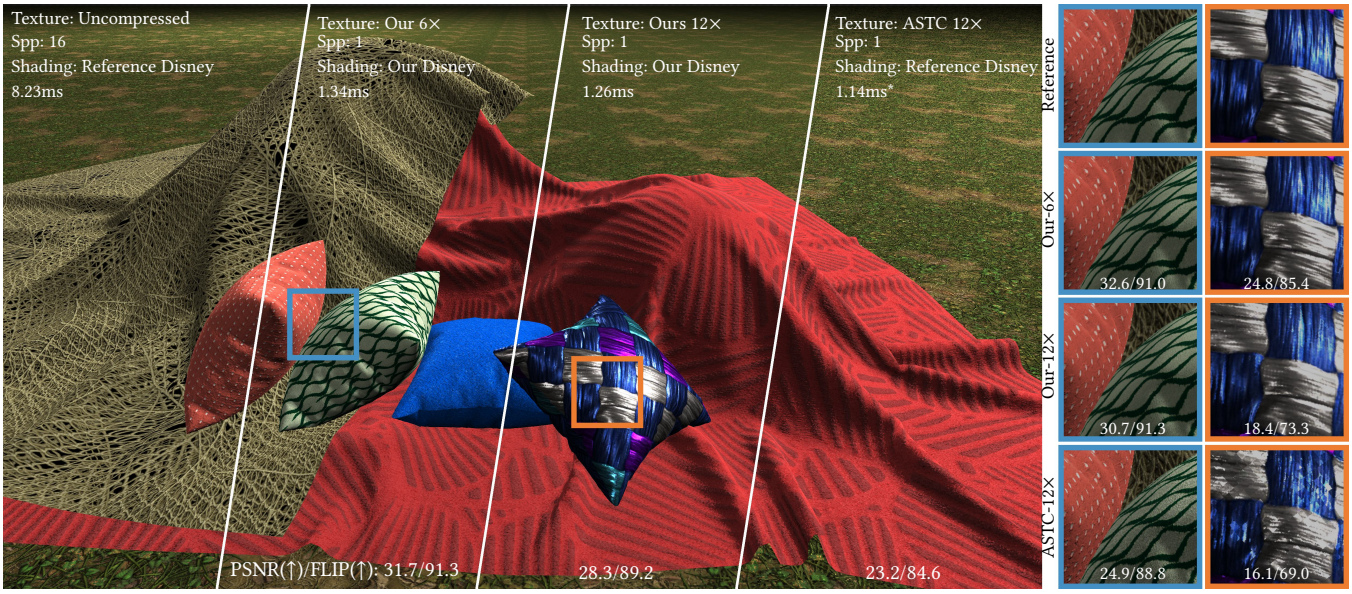

Figure 1: Figure shows the use of *differentiable indirection* for texture compression/sampling and parametric shading. Our technique relies on few linearly interpolated memory lookups and applies to a wide range of tasks in the graphics pipeline including signed distance and radiance field compression.

ABSTRACT

We introduce *differentiable indirection* – a novel learned primitive that employs differentiable multi-scale lookup tables as an effective substitute for traditional compute and data operations across the graphics pipeline. We demonstrate its flexibility on a number of graphics tasks, i.e., geometric and image representation, texture mapping, shading, and radiance field representation. In all cases, *differentiable indirection* seamlessly integrates into existing architectures, trains rapidly, and yields both versatile and efficient results.

CCS CONCEPTS

• **Computing methodologies** → *Reflectance modeling*; **Image compression**; *Shape representations.*

KEYWORDS

Differentiable LUT, Memory Indirection, Multi-modal Representation, Efficient Neural Alternative.

ACM Reference Format:
Sayantan Datta, Carl Marshall, Zhao Dong, Zhengqin Li, and Derek Nowrouzezahrai. 2023. Efficient Graphics Representation with Differentiable Indirection. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/a.bb/ccc.dddd

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

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/a.bb/ccc.dddd

1 INTRODUCTION

Neural primitives are the fundamental building block of neural networks and used for a variety of purposes in graphics applications, such as appearance capture [Zhang et al. 2022], shading [Schied and Kaplanyan 2022], radiance caching [Müller et al. 2021], view-synthesis [Mildenhall et al. 2020], and shadows [Datta et al. 2022]. Having *efficient* neural primitives is vital, due to their impact on latency, power, and training speed. Achieving high runtime performance with neural primitives is essential to the adoption of neural networks in real-time and low-power applications, such as AR/VR.

We introduce a simple primitive with excellent runtime characteristics, featuring *low compute FLOPs*, *minimal memory reads per query*, and *a compact parameter size*. Many networks rely primarily

on multilayer perceptrons (MLP) due to their appeal as universal function approximators; however, MLP layers are often the most computationally expensive component of a network and scale quadratically (both in FLOPs and bytes transferred) with quality due to large matrix operations [Schmidhuber 2015]. Conversely, combining memory grids with fixed function non-linearities such as Spherical Harmonics (SH) [Fridovich-Keil et al. 2022] or ReLUs [Karnewar et al. 2022] reduces compute and memory transfer but incurs a large parameter cost. Our novel primitive – *differentiable indirection* – strikes a balance across these criteria and useful for a variety of data compression and compute representation tasks. It is compatible with any differentiable logic, such as MLPs or fixed function approaches, but significantly reduces or even eliminates reliance on MLPs. Notably, all of our examples are MLP-free, thereby eliminating the need for specialized hardware [Nvidia 2019] acceleration in real-time applications. *Differentiable indirection* draws its expressive power solely from memory indirections and linear interpolation. This approach aligns well with the emerging computing paradigm of *compute in memory* [Lin et al. 2022; Wang et al. 2021], which departs from traditional von Neumann model that MLPs are modelled on. We apply *differentiable indirection* to various tasks in the (neural) graphics pipeline, showcasing its potential as an efficient and flexible primitive for improving runtime efficiency.

2 RELATED WORK

Neural primitives serve as the fundamental building blocks for modern neural techniques. We provide an overview of existing neural primitives and explore their applications in graphics.

MLP architectures provide a compact implicit representation that seamlessly scales up to higher dimensional inputs, such as signed distance field [Park et al. 2019], neural radiance field [Mildenhall et al. 2020], and neural BRDF [Bi et al. 2020; Boss et al. 2021a,b; Zhang et al. 2021]. They trade parameter size for compute, memory bandwidth, and a relatively longer training time as each training example affects all network weights. Even small MLPs (2 layer deep, 64 unit wide) are computationally and memory-intensive, requiring thousands of FLOPs and bytes transferred per query. In contrast, our differentiable indirection relies solely on memory indirections and linear interpolations, resulting in significantly reduced computational demands.

Grid-based representations explicitly store trainable parameters on a regular grid [Chabra et al. 2020; Karnewar et al. 2022] or a tree [Sara Fridovich-Keil and Alex Yu et al. 2022; Takikawa et al. 2021] and then retrieve them at run-time using the input coordinate as key. The stored features are processed further using a non-linearity, such as ReLU [Karnewar et al. 2022] or Spherical Harmonics (SH) [Fridovich-Keil et al. 2022]. While suitable for fast, localized updates, such explicit representations tend to have large parameter sizes and has difficulty scaling up to higher dimensional inputs.

Recent works combine MLP and grid representations to balance between memory and computational cost, achieving complex neural shading [Kuznetsov et al. 2021; Zeltner et al. 2023], neural material texture compression [Vaidyanathan et al. 2023], and efficient & high-quality neural radiance field rendering [Chen et al. 2022; Müller et al. 2022; Sun et al. 2022; Takikawa et al. 2022a]. Particularly, instant-NGP combines *multi-resolution hash encoding* [Müller et al. 2022] with pyramid of latent features, demonstrating to be effective for a wide-range of reconstruction and compression applications. Similarly, we also show the effectiveness of our primitive in broader context of neural rendering. Moreover, our technique is effective as a standalone unit without an MLP. This provides unparalleled efficiency advantages, making it particularly suitable for low-power applications like neural shading on mobile devices.

3 OVERVIEW

Differentiable indirection (*DIn*) is a flexible and powerful tool for solving compute/data problems in both the modern and neural graphics pipelines. *DIn* is similar to a pointer indirection - we query a memory location that contains a pointer to a secondary location containing the final output. However, we also make pointer indirection differentiable, hence *Differentiable indirection*. Our algorithm learns the pointer values stored as an array using gradient descent. *DIn* is a flexible and simple-to-integrate representation, as demonstrated in our applications to geometric and image representation, texture mapping, shading, and radiance field compression.

Our technique in its simplest form requires two arrays – a **primary** array and a **cascaded** array. The article uses the same terminology throughout. Lookup into the primary array returns a pointer into the cascaded array. The corresponding location in the cascaded array contains the output. Figure 2 shows a visual representation of differentiable indirection where the primary and the cascaded arrays are highlighted in orange and blue respectively.

Differentiable Arrays: We introduce fully differentiable arrays - the main building block of our technique. A key requirement of our technique is that the arrays are not only differentiable w.r.t. to the content of each cell but also w.r.t. to its indices or the uv-coordinates. The latter allows the gradients to backpropagate though the cascaded array to learn the pointer values stored in the primary array. The differentiability is achieved by linearly interpolating the array cells. For an input coordinate $\mathbf{x} \in [0,1)^d$, the coordinate is scaled by the array resolution (N) and rounding down $\lfloor \mathbf{x} \cdot \mathbf{N} \rfloor$ and up $\lceil \mathbf{x} \cdot \mathbf{N} \rceil$ - forming a d-dimensional voxel encapsulating $\mathbf{x}$. The vertices of the voxel is interpolated according to the distance of the query point from the corners to produce the output (o) given by:

$$\mathbf{o} = \sum_{i=0}^{2^d-1} \alpha_i \mathcal{F}(\mathbf{c}[i]), \tag{1}$$

where α_i, and $\mathbf{c}[i]$, are the interpolation weights, and the array cell content at the voxel-vertex i respectively. We pass the cell contents through an additional non-linearity $\mathcal{F}$ with some specific characteristics as discussed in the next section. The gradients w.r.t. to input coordinate $\mathbf{x}$ and cell content $\mathbf{c}[i]$ is given by

$$\frac{d\mathbf{o}}{d\mathbf{x}} = \sum_{i=0}^{2^d-1} \frac{d\alpha_i}{d\mathbf{x}} \mathcal{F}(\mathbf{c}[i]), \text{ and } \frac{d\mathbf{o}}{d\mathbf{c}[i]} = \alpha_i \frac{d\mathcal{F}}{d\mathbf{c}[i]} \tag{2}$$

respectively. The gradients are plugged into *autodiff* framework such as PyTorch [Paszke et al. 2017], enabling backpropagation though arrays. Prior techniques - *Multi Resolution Hash Encoding*, abbreviation *MRHE* [Müller et al. 2022] and *ReLU Fields* [Karnewar et al. 2022] only compute gradient w.r.t. cell contents $\mathbf{c}[i]$.

Non-linearity: We apply a periodic non-linearity $\mathcal{F}$ to the primary array and an optional periodic/aperiodic non-linearity to

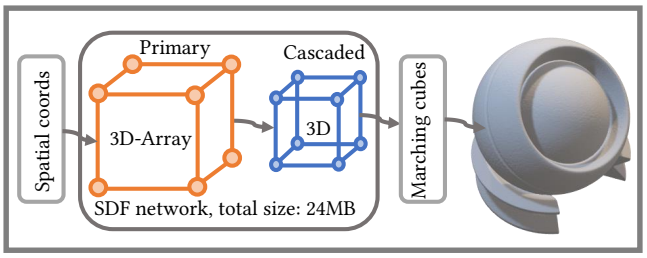

Figure 2: SDF representation using *DIn*. Primary and cascaded resolutions are $200^3 \times 3$ and $64^3 \times 1$ respectively.

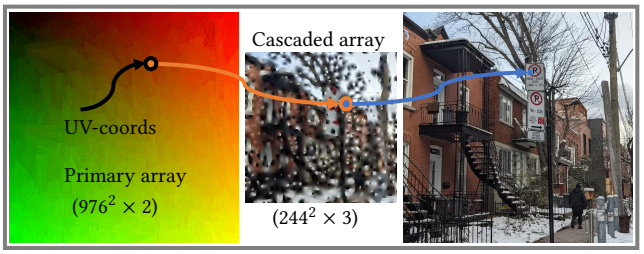

Figure 3: Visualizing learned primary (2D) and cascaded (2D) arrays, compressing 2k image by 6×. A 2D-primary and 4D-cascaded produce better results in practice.

the cascaded array. The purpose of the periodic non-linearity is to bound and continuously *wrap-around* the array content for use as an input in the next layer. We test two periodic functions - a sinusoid given by $(1+\sin(nx))/2$ and a non-negative triangle wave with a period 2 with peak output 1 at input 1. We use the triangle wave for all cases except Disney-BRDF where we use the sinusoid. Note that the non-linearity $\mathcal{F}$ is applied before interpolation, thus can be removed during inference and baked directly into the array cells for improved efficiency. The output is also bounded to $[0, 1)$ when using a periodic non-linearity, lending the opportunity to quantize the array values to 8/16-bit for inference without significant impact on quality; a property we utilize heavily for all tasks.

Initialization: *Differentiable indirection* is thus a cascade of multi-dimensional differentiable arrays. We initialize the primary array using a linear ramp resembling a standard uv map on a 2D-square or a similar analog in higher dimension. The gradient descent algorithm simultaneously updates the primary array containing the pointer and the cascaded array containing the output. Note that linear interpolation puts an implicit constraint on the values the primary array can accommodate. If we imagine the uv-map as a fabric, the gradient descent is only allowed to locally wrinkle the fabric. This effect is illustrated in figure 3. The choice of initialization for the cascaded array is application specific.

4 APPLICATIONS

We demonstrate applications of *Differentiable indirection* across various stages of the graphics pipeline – starting with geometric representations, followed by examples in deferred shading such as image and texture compression, and parametric shading. Finally, to compress implicit representation such as neural radiance field.

4.1 Compact geometric representation

Our technique is easily adapted to implicit geometry representation tasks such as a *Signed Distance Function* (SDF) representation. SDFs volumetrically encodes the zero-level set of a spatial 3D shape function. *Differentiable indirection* readily applies to such a representation, compressing the volumetric information in spatial 3D arrays. Figure 2 shows a primary array queried by 3D spatial coordinates that point to the learned cascaded array of scalar SDF distances: we train one network per SDF and use marching cubes [Lorensen and Cline 1987] for illustrative surface reconstructions.

Implementation details. We generate training samples pairs – a position and its corresponding signed distance using an SDF dataset generator [Takikawa et al. 2022b]. We preferentially sample points closer to the surface zero-level set [Takikawa et al. 2021]; one billion samples from near the surface and 20 million uniformly distributed over the volumetric domain. We use 100 million near-surface samples to compute test time error statistics. While *MRHE* proposes MAPE as the loss function, this results in many "floater" artifacts near the surface; we believe this is due to MAPE over-emphasizing on-surface sample importance at a cost of distorting the distance field slightly off-surface. We solve this problem by quantizing signed distances to ±1 a.k.a. *Truncated SDF* and applying an MAE loss. In a grid based technique like ours, only having two discrete values allows the transition-boundary (representing a surface) to adapt more freely. Figure 2 illustrates a 3D shape generated using our training methodology. We include more results in section 6.0.2 and training details in supplemental 1.6

4.2 Compact texture and image representations

We apply *Differentiable indirection* to real-time texture and natural image (de-)compression, and filtered texture sampling.

4.2.1 Compact image representation . For this task we query the primary array with 2D uv-coordinates (figure 3). The cascaded array encodes the corresponding color value at the uv location. Figure 3 visualizes the learned arrays' contents – note the primary array favors high frequency details as a pseudo-distorted uv-map. We however recommend alternative network configurations than those in figure 3 which produce improved results, as described next.

Implementation details. We use a 2D 4-channel array as primary and a 4D 3-channels array containing RGB values as the cascaded. For *PBR* textures, we extend the number of channels in the cascaded array to include various shading parameters. During training, the network maps the queried uv-coordinates to the corresponding color/parameter value. A higher dimensional (> 2D) cascaded array is chosen based on hyper-parameter search. For a mixture of textures containing natural images and several *PBR* materials from *Adobe Substance*, we noticed a 15% improvement in PSNR moving from 2D to 3D and 3% improvement from 3D to 4D. Stratified random sampling is used to generate the training uv-coordinates where each strata corresponds to a texel in the base texture. Target color values are obtained with bi-linear sampling the reference image at the queried uv-coordinates. Details in supplemental 1.3.

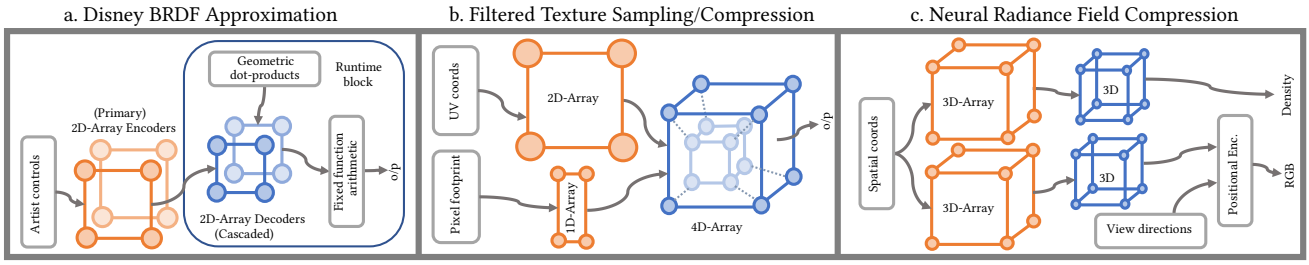

Figure 4: From left to right, parametric shading, compact texture sampling, and radiance field based on *differentiable indirection*.

4.2.2 Neural Texture Sampler . For shading applications, we also need to account for texture filtering [Williams 1983] based on the projected pixel footprint onto a surface. With minor modifications to the previous compression-only network, we can additionally treat filtered texture sampling. Figure 4(b) shows our texture sampling/filtering network configuration, requiring two inputs – a uv-coordinate, and a pixel footprint magnitude, both of which are readily available in modern interactive renderers.

Implementation details. We use two primary arrays - a 2D array with 3 channels for input uv coordinates, and a 1D array with 1 channel for pixel footprint. The output of the two primary arrays is concatenated and used as the input for the cascaded 4-D array. The network approximates a *trilinear texture sampler* similar to those available on a modern GPU but without storing an explicit mip-chain. We emulate a GPU texture sampler in code to generate our target data. We train our network on random uv-samples and pixel-footprint values with corresponding target generated with emulated trilinear filtering. More details is provided in the supplemental 1.4.

The cost of evaluating the new network increases only by a fraction compared to section 4.2.1, while providing higher quality per sample for shading tasks, especially for pixels with large footprint at grazing angles. The effect is illustrated in Figure 5.

4.3 Efficient parametric shading models

So far, we have seen the applications of our technique in data representation. This section introduces parametric shading as compute approximation task. We use our technique on two different *BRDFs* - a simple isotropic GGX and a much more complex *Disney BRDF*.

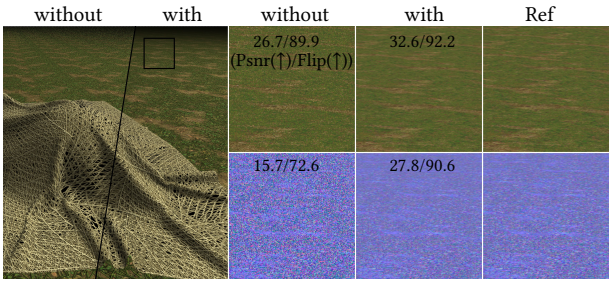

Figure 5: Comparisons of texture compression with and without pixel footprint. Without the footprint information, the output at grazing angles is noisy as shown in the cutouts.

4.3.1 Isotropic GGX approximation . Isotropic GGX is a popular *BRDF* used for specular shading, expressed analytically as $D(h_z, \alpha_h) = \alpha_h^4 \cdot (1 + (\alpha_h^4 - 1) \cdot h_z^2)^{-2} / \pi$. We approximate the analytic expression using our technique. We use the input parameters h_z, α_h as the input uv-coordinates to the primary array (figure 6) and compare the output of the cascaded array with the corresponding output from the analytic expression. The example serves as a benchmark for comparing various neural primitives in section 6.0.1.

4.3.2 Principled Disney BRDF approximation . We approximate the Disney BRDF using *differentiable indirection*, retaining all artist controls with negligible impact on final quality (figure 11), all while improving evaluation efficiency (table 1) to the reference analytic implementation. We follow the *Principled Disney BRDF implementation* reference [Li 2022] to generate our training data. Excluding glass/transmission term, the reference BRDF has 10 artist controllable parameters and seven geometric dot-products as the input. The challenge is handling the high-dimensional input while also being efficient. While theoretically possible, 17D arrays would be prohibitively expensive in practice. Other primitives using MLPs such as *MRHE* can scale up with higher dimensional inputs, they require exceedingly large MLPs to attain desirable results here, and do not improve upon the efficiency of reference evaluation.

We use a divide-and-conquer approach to partition the task into several components, leveraging the available domain knowledge in this setting: we start by factoring out the albedo ($\mathcal{A}$) from final BRDF output – i.e., as referenced in equation 19 in [Li 2022] – and rewrite the reference BRDF equation as $\text{Disney}(\mathbf{x}, \mathcal{A}) =$

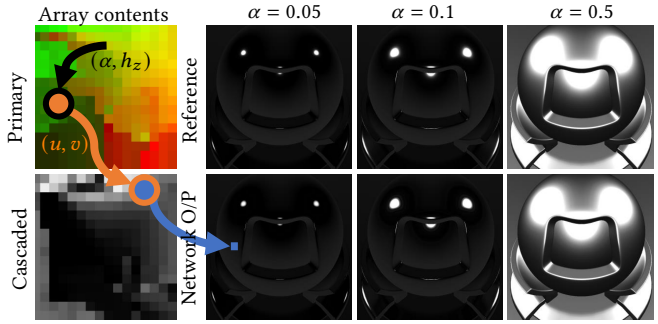

Figure 6: Visualization of learned primary and cascaded arrays for Isotropic GGX approximation and the rendering results on right with an average 40dB PSNR.

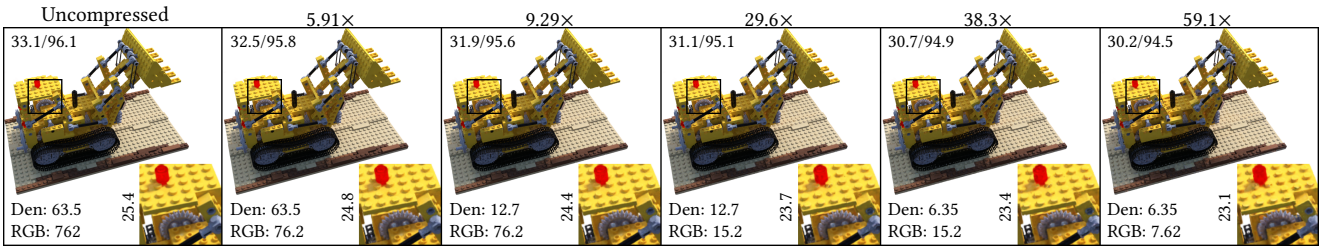

Figure 7: Radiance field compression with *DIn*. Top left shows average PSNR(↑)/FLIP(↑) w.r.t. to all images in test set for varying compression ratios. Bottom left shows the size (in MBs) of the Density and RGB fields. PSNR(↑) for the cutout is provided nearby. A typical resolution for Density, and RGB grid at 9.3× compression is $160^3 \times 3/64^3 \times 1$, and $280^3 \times 3/64^3 \times 12$ respectively.

$(\mathcal{A}/l) \cdot p(\mathbf{x}) + q(\mathbf{x})$ where $\mathbf{x} \in [0, 1)^{17}$ are the control parameters, and l is the luminance computed as a weighted sum of albedo-RGB channels according to $(0.2126, 0.7152, 0.0722)$. The p and q terms are single channel positive scalars obtained directly from our factorization. Note that the learnable quantities p, q do not learn any color information, instead we modulate the albedo with the learned parameters. This is crucial for reducing color bleeding in the final output. We further refactor p and q according to

$$\begin{aligned} p(x) &= c_d D_d + c_{m0} D_m + c_{s0} \\ q(x) &= c_c D_c + c_{m1} D_m + c_{s1}, \end{aligned} \tag{3}$$

where D_d, D_m, and D_c are the Disney-diffuse, Disney-metallic, and Disney-clearcoat distribution (equation 5, 8, and 12 in [Li 2022]). The remaining c_* terms follow naturally from the factorization, and we first approximate each D_* and c_* term independently using *differentiable indirection*; the primary aim here is the understanding of the functional space required to approximate each term accurately. Once we understand the requirements, we find similar lookups and merge them into fewer indirection pairs. Figure 4(a) illustrates the resulting network architecture for Disney BRDF approximation, which also leverages the fact that the artist control parameters can be encoded into a latent vector (separately and completely offline). As such, the primary arrays are also encoders while the cascaded arrays are runtime decoders. Such a factorization yields efficiency advantages, discussed in section 6.0.6.

4.3.3 Optimized shading pipeline . We improve the quality of our final rendering (figure 1, 13) by optimizing our learned shading pipeline end-to-end. Note that our learned texture sampler is trained independent of our learned Disney BRDF. The goal here is to further improve the quality of the rendered results by making our sampler aware of the learned BRDF. We do so by training the texture sampler with an additional regularization term that compares the final rendered output (through learned Disney BRDF) with reference rendered output. The extra complexity only affects the training pipeline while the networks and inference pipeline essentially stay the same. More details in supplemental section 1.5.

4.4 Compact radiance fields

NeRF volumetrically represents a scene as 5D spatiodirectional function whose outputs are spatial density and view-dependent emitted radiance. While the original *NeRF* [Mildenhall et al. 2020] used a deep neural network to represent the density and radiance, subsequent versions obtain better quality, and improved training using coordinate networks comprising 3D data-structure such as regular grid [Karnewar et al. 2022] along with fixed function non-linearity such as SH or ReLUs. Recent state of the art *Direct Voxel* [Sun et al. 2022] achieves high quality representation by adaptive scaling of the voxel grid resolution and fine tuning the representation at each update. However, the resulting voxel grid is enormous requiring hundreds of megabytes in parameter space. We follow up on their work and improve the compression of their voxel representation by an order of magnitude without losing significant details (figure 7).

Implementation details. We extract density and view-dependent radiance/RGB-grid pre-trained using the *Direct Voxel* technique and apply our technique to compress the volumes. We use a multi-head network (figure 4(c)) for the density and RGB fields and use spatial 3D coordinate as input for the two primary arrays. The output of the cascaded array is trained against trilinearly interpolated values from the corresponding target grids. Details in supplemental 1.7.

5 GENERAL IMPLEMENTATION DETAILS

The section provides general implementation details across tasks. We refer to the supplemental for more accurate task-specific details. To train our network, we use PYTORCH without any special optimizations and use 32-bit *full-precision* arithmetic for backpropagation and data generation. We use the *ADAM* optimizer with a learning rate of 0.001 and an MAE loss. An important hyperparameter for training is $\rho = N_p/N_c$ – the ratio of the length of one side of the primary array (N_p) to the cascaded array (N_c). We set pseudo-optimal values through hyperparameter search in each setting. In all cases, the optimal ρ is greater than 1. This often results in the cascaded array being much smaller in size (raw bytes) compared to the primary array. In a real-time environment, the access pattern to the primary array is more coherent compared to the cascaded array, but the cascaded array is also much smaller – a potentially-exploitable property in caching hardware.

Another advantage when ρ is greater than 1 is in quantization: note that the cascaded array may contain signed or unbounded values, rendering the effective use of quantization cumbersome and additionally ineffective given the smaller array size; however, for primary arrays, in all cases we quantize values to 8-bits after applying the non-linearity $\mathcal{F}$ to array cells. Quantization either

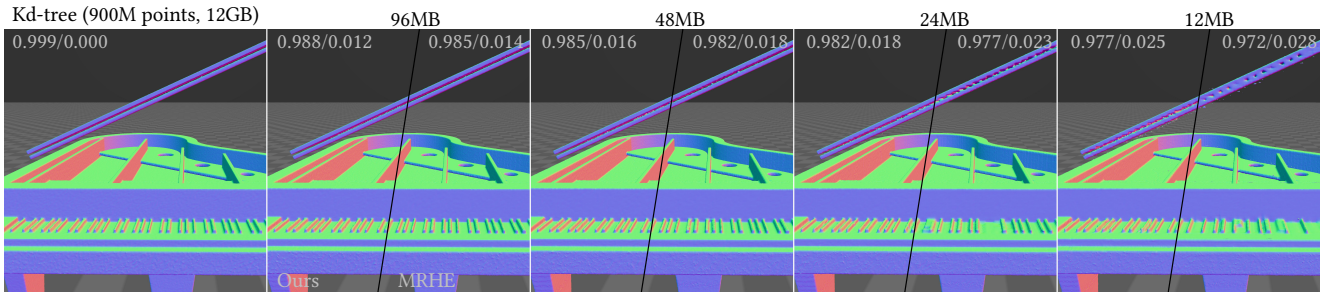

Figure 8: Comparing *differentiable indirection*(left) with *MRHE*(right) at equal parameter for SDF compression. *MRHE* requires 6 extra memory lookups (8 levels) and ≥ 512 extra FLOPs (4 layer, 16 unit MLP) to achieve similar results. Metrics IoU(↑)/MAE(↓) are measured on uniform near-surface SDF samples. A typical resolution of the primary/cascaded grid at 96MB is $316^3 \times 3/96^3 \times 1$.

lowers memory size or improves resolution for the primary array. Note that a single level grid cannot take advantage of this approach.

To set array sizes, N_p, N_c, we specify two inputs – desired total representation size (in bytes) and ρ. Some additional constraints are used – the length of the sides are a nearest multiple of eight for primary arrays and four for cascaded arrays. We use a simple 1-D search to satisfy these constraints whilst closely matching the desired size. As discussed in section 3, we initialize the primary array with an uv-ramp in 2D or equivalent in higher dimension. More generally, a d-dim array is initialized such that it returns an identity mapping between the input and output.

We implement *Multi Resolution Hash Encoding* in our PYTORCH-based framework. When comparing with *MRHE*, in all cases we use eight multi-resolution grid levels, composed of six levels of multi-dimensional arrays and two levels for the hash-table. We found that quantizing arrays and hash table to 8-bit increases resolution and quality. Each level stores a 2-channel feature vector, totaling 16 latent input channels to the MLP. The MLP is four layers deep and 16 wide. For images, we use bilinearly-interpolated 2D arrays, and for NeRF/SDF we use trilinear 3D arrays. At equal parameter count, *MRHE* always requires > 5 additional memory accesses and > 1024 additional *FLOPs* compared to our method. Figure 15 shows the training characteristics of our technique w.r.t. *MRHE*.

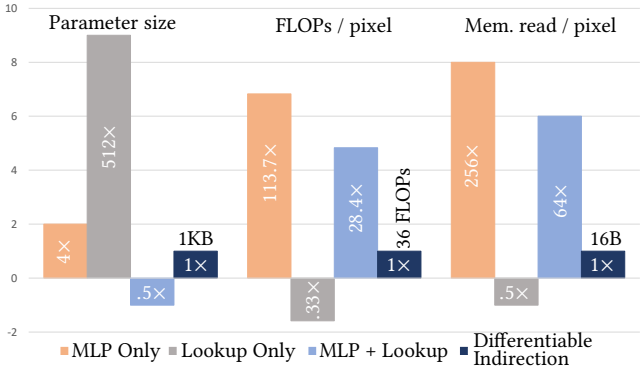

Figure 9: Comparing relative resource utilization of various architectures on a log scale at equal quality (PSNR 40dB) approximation of Isotropic GGX.

6 RESULTS AND ANALYSIS

6.0.1 Isotropic GGX approximation . We begin with a discussion of Isotropic GGX approximation as it serves as a simple testbed to compare various neural primitives and analyze their efficiency. Figure 9 compares various neural primitives at equal PSNR (> 40dB) measured across a range of roughness as shown in 6. Figure 9 also serves as an overview of the neural primitives landscape. Notice how different primitives use resources differently. An MLP with 4 hidden layers and 32 hidden units per layer not only requires thousands of FLOPs but also require a large memory read per pixel to fetch the network weights. A single level grid storing the isotropic GGX as a texture require very large parameter space but very little FLOPs and memory transfer per pixel. A combination of MLP and grid strikes a better balance across the three criterion, but our technique improves further – requiring only a modest parameter space, memory transfers, and FLOPs per pixel. The few FLOPs required in our technique are due to the linear interpolation of array cells and maps directly to hardware texture samplers in GPUs.

6.0.2 Signed Distance Fields . Figure 8 and 14(c) provides a visual and a quantitative analysis of our technique for various parameter sizes. Our technique compares favourably with *MRHE* at equal parameter count but requires fewer lookups and compute per query.

6.0.3 Neural Radiance Fields. Figure 7 shows various compression combinations for density and RGB grid, with total compression between 5-60×. Note that the RGB grid is more compressible than the density grid and we refer to plot 14(b) to choose a combination of density and RGB compression that retains highest quality. While the density field can be compressed by 10×, the RGB field allows a compression of 100x before significant impact on quality. We note the reference density field is about 10x smaller than the RGB field at same resolution due to higher number of latent channels in the RGB field. This may explain the differences in compression as there are more redundancies to be exploited in the RGB field.

6.0.4 Compact Image Representation . Our texture compression technique is resource efficient while being competitive with state of the art *block compression* [1979] techniques such as *ASTC* (figure 14). We require few bytes (≤ 12) memory read per pixel and a few linear interpolations (≤ 20) to decode a texture. Other neural techniques [Vaidyanathan et al. 2023] using per material MLP decoder require

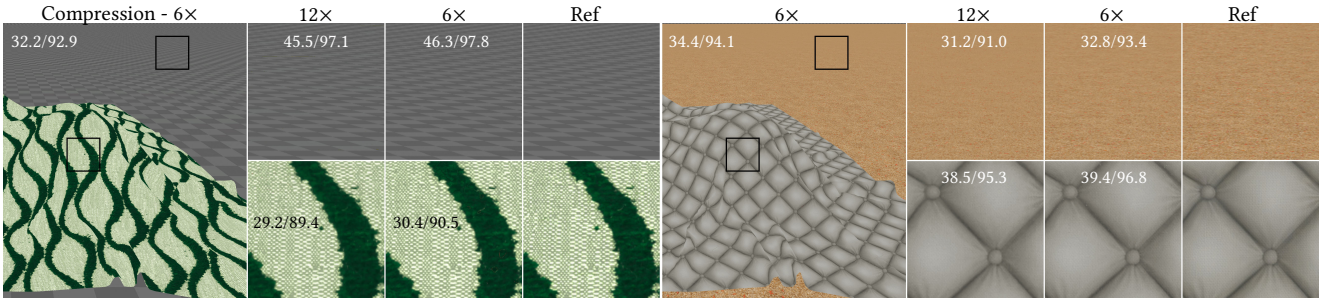

Figure 10: Compressed texture sampling (filtered) using *differentiable indirection* at varying pixel footprints. Reference is uncompressed 16spp anisotropic filtered. PSNR(↑)/FLIP(↑) w.r.t. reference (base resolution 1K) is provided.

thousands of additional FLOPs and *Kilobytes* of memory transfer per pixel to fetch the MLP weights. This is potentially problematic when several materials are present on screen, especially on low end hardware. Our technique has a constant resource utilization independent of the number of materials present on screen.

We compare our technique with *ASTC [2012], ETC2 [2005],* and *MRHE* as shown in plot 14(a). *Block compression* technique ETC2 offer fixed compression of 6x while *ASTC* has variable compression up to 24×. Ours and *MRHE*, being learned techniques, achieve unbounded variable compression. In figure14(a), we downsample the image for compression beyond their respective maximum for block compression techniques. Our technique is visually comparable in quality with proprietary *ASTC* while being more general.

6.0.5 Texture sampling. Figure 10 shows the output of our texture sampler at 6×, and 12× compression. Figure 12, compares a single evaluation of our network (1-spp) with nearest neighbor and anisotropic sampled *ASTC* for Albedo, Normal, and AO textures. Note the cost of evaluation of our network is comparable to nearest neighbor *ASTC* while retaining quality much superior to 1-spp *ASTC*. Thus by amortizing texture compression and filtering in one network we are able to extract higher quality per sample than we could with isolated compression and sampling.

6.0.6 Disney BRDF approximation . Here, the primary and cascaded arrays are split into an offline-encoder and a runtime-decoder (figure 4(a)). The encoder transforms the artist control parameters into a latent representation that is consumed by the decoder along with other geometric dot products. This lends the opportunity to make the encoder arrays much larger - 2k×2k in resolution, while the cascaded arrays (decoder) is small 16×16 in resolution so that they fit in lowest tier caches/SRAM. We refer the reader to the supplied code for exact implementation details.

A sweep across all parameters is shown in figure 11. The runtime part of the network requires four 2D-bilinear lookups and a single 1D lookup. We require an additional 41 fixed function *FLOPs* to combine the output of the lookups into final result. The reference analytic implementation requires 240+ *FLOPs* in total. Our runtime arrays are 16x16 resolution and ≤ 4 channels deep, making it easier to test on a commodity GPU. Table 1 shows the runtime performance of our technique running at 4K resolution for up to 4 point light sources. A performance advantage is obtained when using the hardware texture sampler for bilinear interpolations.

6.0.7 Shading pipeline optimization. Figure 1 and 13 shows final results of our shading pipeline using technique mentioned in section 4.3.3. We obtain an additional 8%, 5% better PSNR in the first, and second figure respectively using this approach. As shown figure 1, the runtime performance at 4K resolution is < 1.5ms for our learned texture sampling and shading on a Nvidia 3090 GPU. In absence of *ASTC* hardware, we report runtime with *BC7* as a proxy for *ASTC*.

7 CONCLUSION AND FUTURE WORK

We show *differentiable indirection* as a powerful primitive that efficiently represents *data* and *compute* across neural graphics pipeline with applications potentially beyond graphics. While our technique is *bandwidth*, *compute*, and *space* efficient, it is challenging to apply our technique to higher dimensions. In section 4.3.2, we show a recipe to overcome this using parameter space factorization exploiting the problem structure. More generally, we suspect factorization techniques such as spectral or tensor decomposition may prove useful. The effectiveness our technique is also improved by scaling the grid resolution, feature count, or by augmenting the latent representation with fixed function logic such as SH, positional encoding or other parameter-free linear/non-linear embedding. While our technique is aimed at improving runtime efficiency, the effectiveness of our technique in the context of direct reconstruction or inverse-rendering tasks is yet to be explored. Finally, our differentiable arrays can be also arranged in a variety of configuration as standard layers in neural/array networks.

ACKNOWLEDGMENTS

We thank Cheng Chang, Sushant Kondguli, Anton Michels, Warren Hunt, and Abhinav Golas for their valuable input and the reviewers

Table 1: HLSL runtime performance of approximate and reference Disney BRDF at 4K resolution on Mobile 3070Ti.

Technique	Point emitter count (time in ms)			
	1	2	3	4
Differentiable Indirection	0.654	0.710	0.798	0.923
Reference Disney	0.690	0.838	1.11	1.41
Baseline Diffuse (shader overhead)	0.642	0.648	0.656	0.668

for their constructive feedback. We also thank Moshe Caine for the horse-model [Caine 2016] with *CC-BY-4.0* license, and Adobe Substance-3D [Adobe 2023] for the *PBR* textures. This work was done when Sayantan was an intern at Meta Reality Labs Research. While at McGill University, he was also supported by a Ph.D. scholarship from the *Fonds de recherche du Québec – nature et technologies*.

REFERENCES

Adobe. 2023. Substance-3D. https://www.adobe.com/products/substance3d/3d-assets.html

Sai Bi, Zexiang Xu, Pratul Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. 2020. Neural reflectance fields for appearance acquisition. *arXiv preprint arXiv:2008.03824* (2020).

Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik Lensch. 2021a. Nerd: Neural reflectance decomposition from image collections. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 12684–12694.

Mark Boss, Varun Jampani, Raphael Braun, Ce Liu, Jonathan Barron, and Hendrik Lensch. 2021b. Neural-pil: Neural pre-integrated lighting for reflectance decomposition. *Advances in Neural Information Processing Systems* 34 (2021), 10691–10704.

Moshe Caine. 2016. 3D Bronze horse model. https://skfb.ly/Lz7L

Rohan Chabra, Jan Eric Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard A. Newcombe. 2020. Deep Local Shapes: Learning Local SDF Priors for Detailed 3D Reconstruction. *CoRR* abs/2003.10983 (2020). arXiv:2003.10983 https://arxiv.org/abs/2003.10983

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. Tensorf: Tensorial radiance fields. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*. Springer, 333–350.

Sayantan Datta, Derek Nowrouzezahrai, Christoph Schied, and Zhao Dong. 2022. Neural Shadow Mapping. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada) *(SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 8, 9 pages. https://doi.org/10.1145/3528233.3530700

E. Delp and O. Mitchell. 1979. Image Compression Using Block Truncation Coding. *IEEE Transactions on Communications* 27, 9 (1979), 1335–1342. https://doi.org/10.1109/TCOM.1979.1094560

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5501–5510.

Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. 2022. ReLU Fields: The Little Non-Linearity That Could. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada) *(SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 27, 9 pages. https://doi.org/10.1145/3528233.3530707

Alexandr Kuznetsov, Krishna Mullia, Zexiang Xu, Miloš Hašan, and Ravi Ramamoorthi. 2021. NeuMIP: Multi-Resolution Neural Materials. *Transactions on Graphics (Proceedings of SIGGRAPH)* 40, 4, Article 175 (July 2021), 13 pages.

Tzu-Mao Li. 2022. UCSD CSE 272 Assignment 1: Disney Principled BSDF. https://sayan1an.github.io/disneyLi.html

Zhiting Lin, Zhongzhen Tong, Jin Zhang, Fangming Wang, Tian Xu, Yue Zhao, Xiulong Wu, Chunyu Peng, Wenjuan Lu, Qiang Zhao, and Junning Chen. 2022. A review on SRAM-based computing in-memory: Circuits, functions, and applications. *Journal of Semiconductors* 43, 3 (mar 2022), 031401. https://doi.org/10.1088/1674-4926/43/3/031401

William E. Lorensen and Harvey E. Cline. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. Association for Computing Machinery, New York, NY, USA, 163–169. https://doi.org/10.1145/37401.37422

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *CoRR* abs/2003.08934 (2020). arXiv:2003.08934

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-Time Neural Radiance Caching for Path Tracing. *ACM Trans. Graph.* 40, 4, Article 36 (jul 2021), 16 pages. https://doi.org/10.1145/3450626.3459812

Nvidia. 2019. Nvidia cooperative matrix. https://registry.khronos.org/vulkan/specs/1.3-extensions/man/html/VK_NV_cooperative_matrix.html

J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, and T. Olson. 2012. Adaptive Scalable Texture Compression. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics* (Paris, France) *(EGGH-HPG'12)*. Eurographics Association, Goslar, DEU, 105–114.

Jeong Joon Park, Peter R. Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. *CoRR* abs/1901.05103 (2019). arXiv:1901.05103 http://arxiv.org/abs/1901.05103

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).

Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance Fields without Neural Networks. In *CVPR*.

Christoph Schied and Anton Kaplanyan. 2022. Systems and methods for graphics rendering based on machine learning. https://patents.google.com/patent/US11436793B1/en US Patent No. 11436793B1, Filed February 12, 2021, Issued September 6th., 2022.

Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (jan 2015), 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Jacob Ström and Tomas Akenine-Möller. 2005. IPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (Los Angeles, California) *(HWWS '05)*. Association for Computing Machinery, New York, NY, USA, 63–70. https://doi.org/10.1145/1071866.1071877

Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2022. Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. In *CVPR*.

Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. 2022a. Variable Bitrate Neural Fields. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada) *(SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 41, 9 pages. https://doi.org/10.1145/3528233.3530727

Towaki Takikawa, Andrew Glassner, and Morgan McGuire. 2022b. A Dataset and Explorer for 3D Signed Distance Functions. *Journal of Computer Graphics Techniques (JCGT)* 11, 2 (27 April 2022), 1–29. http://jcgt.org/published/0011/02/01/

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes. (2021).

Karthik Vaidyanathan, Marco Salvi, Bartlomiej Wronski, Tomas Akenine-Möller, Pontus Ebelin, and Aaron Lefohn. 2023. Random-Access Neural Compression of Material Textures. In *Proceedings of SIGGRAPH*.

Yin Wang, Hongwei Tang, Yufeng Xie, Xinyu Chen, Shunli Ma, Zhengzong Sun, Qingqing Sun, Lin Chen, Hao Zhu, Jing Wan, Zihan Xu, David Wei Zhang, Peng Zhou, and Wenzhong Bao. 2021. An in-memory computing architecture based on two-dimensional semiconductors for multiply-accumulate operations. *Nature Communications* 12, 1 (07 Jun 2021), 3347. https://doi.org/10.1038/s41467-021-23719-3

Lance Williams. 1983. Pyramidal Parametrics. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques* (Detroit, Michigan, USA) *(SIGGRAPH '83)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/800059.801126

Tizian Zeltner, Fabrice Rousselle, Andrea Weidlich, Petrik Clarberg, Jan Novák, Benedikt Bitterli, Alex Evans, Tomáš Davidovič, Simon Kallweit, and Aaron Lefohn. 2023. Real-Time Neural Appearance Models. arXiv:2305.02678 [cs.GR]

Kai Zhang, Fujun Luan, Zhengqi Li, and Noah Snavely. 2022. IRON: Inverse Rendering by Optimizing Neural SDFs and Materials from Photometric Images. In *IEEE Conf. Comput. Vis. Pattern Recog.*

Xiuming Zhang, Pratul P Srinivasan, Boyang Deng, Paul Debevec, William T Freeman, and Jonathan T Barron. 2021. Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–18.

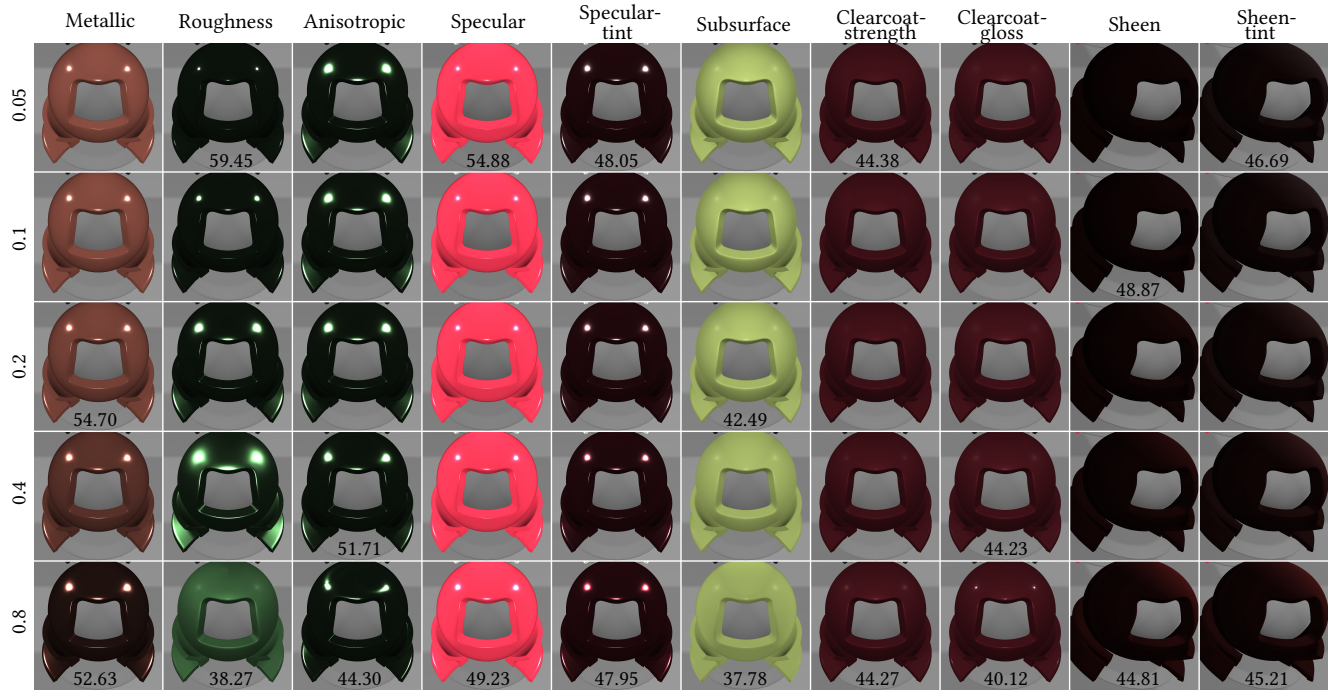

Figure 11: Figure demonstrating the retention of all artist-control parameters using our Efficient Disney BRDF approximation. The minimum and maximum PSNR when compared with analytic evaluation is provided for each row of varying parameter value along the vertical axis.

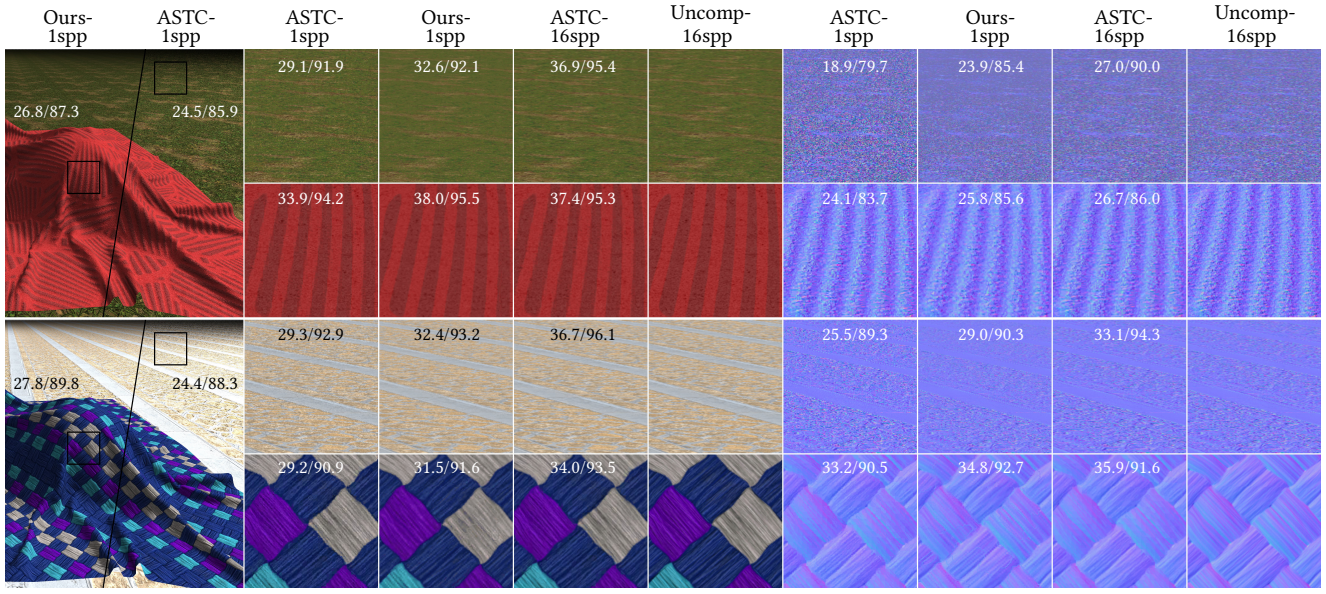

Figure 12: Figure comparing combined effect of texture compression (12x) and filtering using *differentiable indirection* with ASTC at varying pixel footprints and filtering sample count. Note that our technique is computationally comparable to ASTC-1spp and yields better filtered results compared to the same. PSNR(↑)/FLIP(↑) w.r.t. uncompressed 16spp reference is provided. Uncompressed textures are have base resolution of 1K.

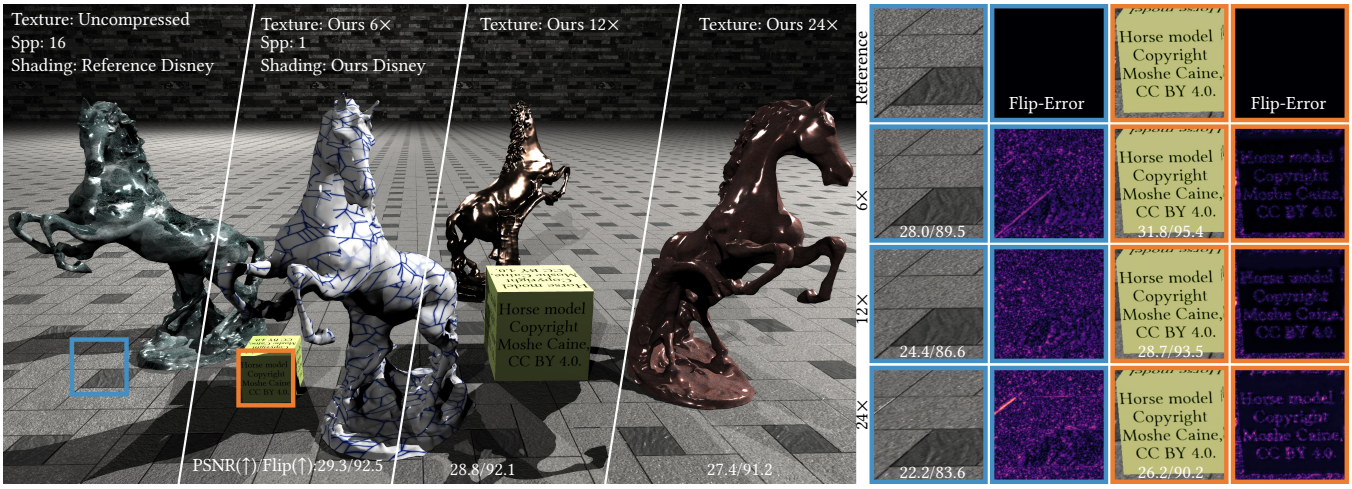

Figure 13: Figure shows the use of our technique for full PBR shading. We jointly optimize our compressed latent texture representation using our differentiable Disney BRDF. PSNR(↑)/FLIP(↑) w.r.t. uncompressed 16spp reference is provided. Uncompressed textures are have base resolution of 1K. Horse model © Moshe Caine, CC-BY-4.0

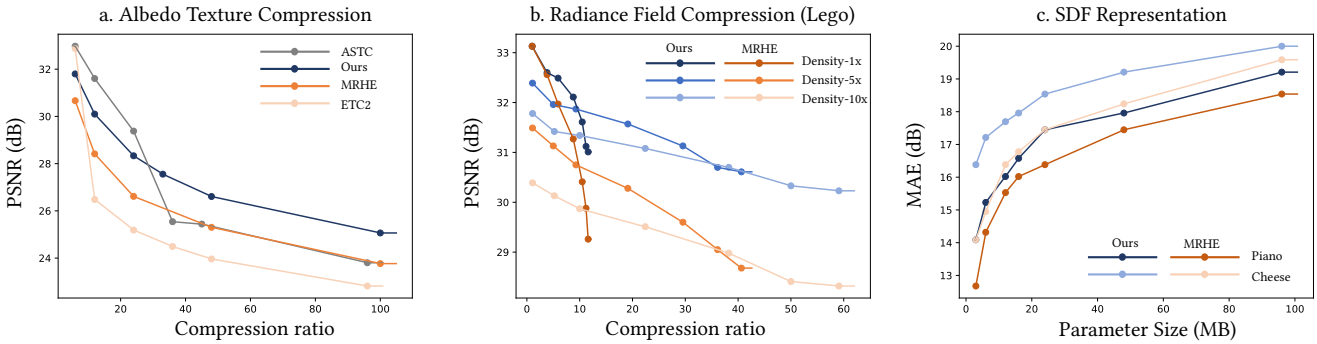

Figure 14: a. Figure compares texture compression behavior for various techniques for 1k textures. *ASTC*, and *ETC2* textures are downsampled beyond their maximum compression ratio of 24x, and 6x respectively. b. Figure shows various iso-lines corresponding to a fixed density-grid compression with varying RGB-grid compression. Total uncompressed grid size is 826MB. c. Figure shows variation of MAE (dB) with parameter size for SDF representation. MAE is computed only on perturbed near-surface samples.

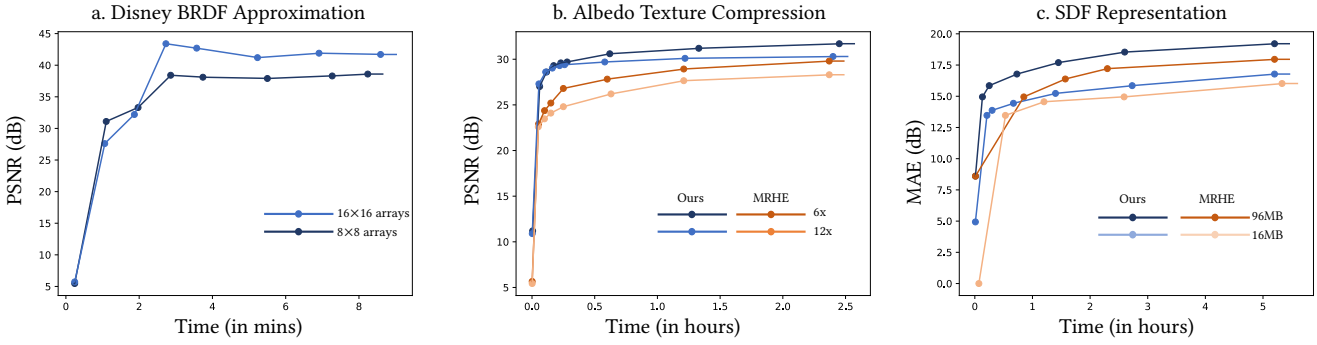

Figure 15: Figure shows the convergence characteristics of our technique and *MRHE* for various tasks in the same framework with similar training overheads.