# Learning Neural Implicit Representations with Surface Signal Parameterizations

Yanran Guan[a,*], Andrei Chubarau[b,c], Ruby Rao[c], Derek Nowrouzezahrai[b]

[a] *Carleton University*
[b] *McGill University*
[c] *Huawei Technologies Canada*

**Abstract**

Neural implicit surface representations have recently emerged as popular alternative to explicit 3D object encodings, such as polygonal meshes, tabulated points, or voxels. While significant work has improved the geometric fidelity of these representations, much less attention is given to their final appearance. Traditional explicit object representations commonly couple the 3D shape data with auxiliary surface-mapped image data, such as diffuse color textures and fine-scale geometric details in normal maps that typically require a mapping of the 3D surface onto a plane, i.e., a surface parameterization; implicit representations, on the other hand, cannot be easily textured due to lack of configurable surface parameterization. Inspired by this digital content authoring methodology, we design a neural network architecture that implicitly encodes the underlying surface parameterization suitable for appearance data. As such, our model remains compatible with existing mesh-based digital content with appearance data. Motivated by recent work that overfits compact networks to individual 3D objects, we present a new weight-encoded neural implicit representation that extends the capability of neural implicit surfaces to enable various common and important applications of texture mapping. Our method outperforms reasonable baselines and state-of-the-art alternatives.

*Keywords:* Neural implicit surfaces, Surface parameterization, Overfit digital content

## 1. Introduction

The 3D surface of an object can be encoded *implicitly* as the zero isocontour of a 3D scalar field, such as a distance field [1, 2]. Neural networks have become an alluring and powerful tool for parameterizing these fields, and these *neural implicit representations* are capable of encoding a variety of 3D shapes. One such recent approach looks to efficiently encode a *single 3D object* as the weights of a small, overfit neural network [3], unlike the original approaches that seek to learn a latent embedding of *many* 3D shapes [1]. In most cases, however, these neural implicit representations focus exclusively on the underlying scalar distance field that encodes the surface geometry, ignoring auxiliary appearance data commonly co-authored during the digital content creation process, such as diffuse colors and fine-scale geometric deviations.

On the other hand, those works that do consider the neural representation of object or scene appearance tend to implicitly define the appearance properties, such as with continuous radiance fields [5], in a manner that does not disentangle them from objects' geometry. Here, it is difficult to decouple the auxiliary appearance data from its underlying geometry, e.g., to edit it separately from the object data — as is standard in 3D digital content creation pipelines.

Figure 1: A collection of textured objects, rendered using a sphere tracer [4], where the geometry is represented by the neural implicit surface encoded using the model of Davies et al. [3] and the texture is applied through our neural surface parameterization.

We instead seek to learn neural representations that are suitable for surface geometry *and* appearance data,

*Corresponding author
*Email address:* yanran.guan@carleton.ca (Y. Guan)

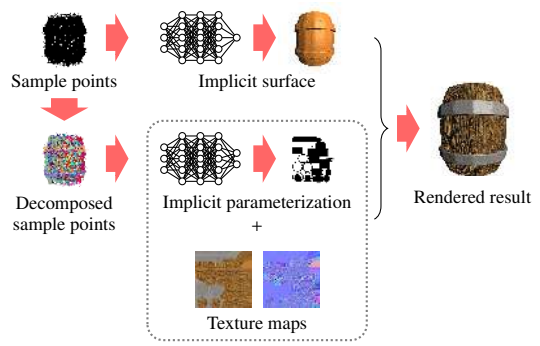



Figure 2: The pipeline of applying our neural implicit surface parameterization to the neural implicit surface to enable texture mapping. For each object, we train two neural representations, one representing the implicit surface and the other representing the implicit parameterization. Then, when rendering the object, we can derive the associated 2D UV coordinate for any 3D location on the zero isocontour defined by the neural implicit surface, thus allowing for texture mapping.

without entangling the two. Inspired by traditional explicit 3D object representations, i.e., meshes, we rely on surface parameterizations [6] to treat auxiliary appearance data in a manner that naturally disentangles it from the underlying geometry. These parameterizations — also called UV maps — map 3D points on the surface onto a 2D chart. During visualization, we can invert this mapping to project 2D image data onto the 3D surfaces: a process called *texture mapping*.

Our main idea is to treat the surface parameterization as an implicit bijective function, which maps 3D input locations onto unique 2D UV coordinates, and we parameterize this function with a compact neural network. We jointly learn an overfit neural implicit surface network with our appearance mapping network for texture mapping. Figure 1 illustrates a collection of 3D objects with their textures applied using our learned surface parameterization. The main challenge here is that the neural surface parameterization must reason about the "unwrapping" of 3D surfaces onto the 2D UV texture domain. In practice, object textures comprise a discontinuous collection of piecewise smooth charts for each mesh segment, known as a texture atlas.

The discontinuous, multimodal nature of these charts makes them challenging to learn, and we propose a simple and effective learning strategy — which relies on applying a spatial decomposition when conditioning the input of the network — to tackle this difficulty. The decomposition components are defined for the input 3D locations based on the parameterization charts, each of which corresponds to a segment of the surface. We then apply a weight-encoded neural implicit representation to the surface parameterization, overfitting a feed-forward multilayer perceptron (MLP) to individual objects and their UV maps.

Our model can be applied directly to neural implicit surfaces and enables texture mapping on these surfaces, as illustrated and explained in Figure 2. In this paper, we apply our model jointly with overfit representations of neural implicit surfaces (e.g., [3]) with only modest overhead.

## 2. Related work

We briefly review the relevant literature, including neural implicit surface and appearance, surface parameterization, and overfit neural representations.

*Neural implicit surfaces.* Neural networks can approximate scalar functions $f\colon \mathbb{R}^3 \rightarrow \mathbb{R}$, where surfaces are implicitly represented as zero isocontours of a level set. These functions include occupancy fields [7, 8], distance fields [1, 2], or space partitioning trees [9, 10]. Recent works show that neural networks implemented with periodic functions [11, 12, 5] can improve the approximation of these scalar functions with reconstructions of high frequency details. Davies et al. [3] overfit neural networks to single shapes as weight-encoded representations of 3D surfaces, which can be interpreted as an efficient and lossy compression of 3D shapes. Takikawa et al. [13] propose a hierarchical representation that can be adaptively scaled to different level of details, combining neural implicit surfaces with an octree feature volume, for rapid final visualization.

These works focus only on 3D geometric surface representations, precluding texture and appearance mapping.

*Neural implicit appearance.* Similar to implicit surfaces, the appearance of objects or scenes can also be implicitly modeled by a neural network as a continuous function, typically mapping from 3D locations to RGB colors [14, 15, 16, 17]. These mappings are limited to representing simple, unimodal (i.e., only diffuse color) textures. To support a greater diversity of appearance properties, implicit surface light field by Oechsle et al. [18] conditions their model on lighting and viewpoint information, allowing the model to represent properties such as specular reflection and shadows. Neural radiance fields (NeRFs) [5] take a continuous 5D input space (3D location and 2D view direction) and map them to volumetric RGB opacities that can be accumulated to form 2D images using a (differentiable and costly) volume sampling-based reconstruction.

These continuous, implicit volumetric appearance functions are inflexible and hard to manipulate. To address this limitation, NeuTex [19] uses a separate network — atop of a volumetric NeRF representation — in an attempt to disentangle texture content into a 2D projective space parameterized by a cube map. AUV-Net [20] learns to embed 3D surfaces into aligned 2D spaces and generates corresponding texture images, thus enabling texture transfer. While similar in spirit to our motivation, we instead directly learn for each individual object the *surface parameterization* that can be used to map *any* auxiliary appearance data onto our 3D surfaces during visualization.

*Surface parameterization.* Computing a surface parameterization of a 3D object is a long-standing problem in computational geometry and computer graphics, and we refer interested readers to the comprehensive survey by Sheffer et al. [6]. Briefly, a surface parameterization refers to a bijective mapping from 3D locations on an object's surface to 2D UV coordinates in a charted parameter space. Algorithms for computing planar parameterizations from explicit polygonal mesh surfaces include methods that compute surface-to-surface bijective mappings — such as with graph embeddings [21] — or those that minimize distortion metrics of the original mesh, e.g., angular [22, 23, 24], distance [25, 26], area [24, 27], and boundary distortions [28]. Surfaces of greater geometric complexity introduce more distortion in these parameterizations so that, in practice, mesh cutting and chart packing [29, 24] are usually necessary in order to segment the original surface into several charts suitable for independent parametric distortion minimization.

Recently, given the advances of gradient-based optimization for neural networks and the growing availability of large 3D surface datasets, parameterization has also been studied through the lens of a neural representation problem, where networks are trained to map 2D coordinates to 3D surface locations [30, 31, 32, 33]. These methods aim to recover the geometry of 3D surfaces in a parametric manner and cannot be directly applied to our auxiliary appearance texture mapping setting. Our model instead learns the *inverse mapping*, directly reasoning about the UV maps of parameterization charts in an underlying texture atlas.

*Overfit digital content.* Recent work has shown that the weights of simple feed-forward MLPs can yield efficient compressed representations for many forms of digital content, including images (by mapping pixel locations to RGB values [34, 35]) and 3D shapes (by mapping 3D locations to signed distances [3, 36]). The difficulty of encoding high frequency signals with MLPs can be mitigated using periodic encodings and activations [11, 12, 5]. The compression speed and rate-distortion performance can be improved through meta-learned initializations [35, 37].

We also rely on weight-encoded neural implicit surface parameterizations, overfitting MLPs to objects' UV maps. Our model complements existing work on weight-encoded neural representations that focus exclusively on geometry encoding [3]. We further achieve a higher compression rate by pruning our model using the lottery ticket method [38].

## 3. Methodology

The parameterization of a surface $\mathcal{S}$ is defined as a mapping from the surface points $\mathbf{p}_\mathcal{S} \in \mathcal{S}$ to UV coordinates $\mathbf{u} \in \mathbb{R}^2$, denoted as:

$$\mathrm{UV}_\mathcal{S}(\mathbf{p}_\mathcal{S}) = \mathbf{u}. \tag{1}$$

To make this function applicable jointly with an implicit surface defined by the signed distance function (SDF), we want to have both functions work on a consistent domain. Therefore, we extend the parameterization function to all query points $\mathbf{p} \in \mathbb{R}^3$ as follows:

$$\mathrm{UV}(\mathbf{p}) = \mathrm{UV}_\mathcal{S}\left(\arg\min_{\mathbf{p}_\mathcal{S}} |\mathbf{p} - \mathbf{p}_\mathcal{S}|\right). \tag{2}$$

Our objective is to train a neural network $f_\theta$ that approximates this parameterization function, such that

$$f_\theta(\mathbf{p}) \approx \mathrm{UV}(\mathbf{p}). \tag{3}$$

We describe our method for representing surface signal parameterization using neural networks. We first explain how we generate the training data and then the design of our neural implicit representation.

### 3.1. Training set

We sample our training set based on the importance sampling [39] scheme proposed by Davies et al. [3], which permits the integration of weighting functions to define the importance over the sampling domain. For each sample point, apart from the UV coordinate given by Equation 2, we generate a decomposition component label indicating the surface segment in a texture atlas to which the point belongs.

*Importance sampling.* Given the SDF describing an object's surface that maps every point coordinate $\mathbf{p}$ in the 3D sampling space to a scalar distance $d \in \mathbb{R}$, denoted as:

$$\text{SDF}(\mathbf{p}) = d, \tag{4}$$

and a neural network $g_{\theta'}$ that approximates the SDF:

$$g_{\theta'}(\mathbf{p}) \approx \text{SDF}(\mathbf{p}), \tag{5}$$

the objective of the weighting function is to assign higher weights to 3D locations that are closer to the implicitly defined surface $\text{SDF}(\cdot) = 0$, such that the signals closer to the surface can be better represented by neural networks during learning. Specifically, we use the following metric [3] to define the importance for each sample point:

$$w(\mathbf{p}) = e^{-\beta|\text{SDF}(\mathbf{p})|}, \tag{6}$$

where $\beta$ is a coefficient ranging $[0, +\infty]$. In our experiments, we set $\beta = 60$. We then apply a Monte Carlo approximation to down-sample a set $S$ of $n$ points from a set of $m$ uniformly sampled points $U$, such that the mean absolute error (MAE) between the real distances and the distances predicted by the neural approximation $g_{\theta'}$ computed over the down-sampled set is approximated to the weighted MAE over the uniformly sampled set, i.e.,

$$\begin{aligned} &\frac{1}{n}\sum_{\mathbf{p}\in S}|\text{SDF}(\mathbf{p}) - g_{\theta'}(\mathbf{p})| \\ \approx &\frac{1}{m}\sum_{\mathbf{p}\in U}|\text{SDF}(\mathbf{p}) - g_{\theta'}(\mathbf{p})|w(\mathbf{p}). \end{aligned} \tag{7}$$

Davies et al. [3] provide the full details of this sampling scheme and it applies seamlessly to our setting.

*Spatial decomposition.* Previous research on deep generative models has observed that conditioned inputs can help models learn complex structured output representations [40]. We observe that the discontinuities and the piecewise nature of parameterization charts in texture atlases complicate learning; neural networks simply tend to learn smoothed boundaries, which for texture mapping specifically results in jarring visual artifacts. To minimize this effect, we assign a unique discrete label to each region of the parameterization charts, and condition the input to our model on this decomposition signal to indicate different surface segments according to the parameterization charts.

Given a texture atlas composed of $k$ parameterization charts for $k$ different surface segments, we decompose the sampling space into $k$ components by assigning each sample point to its nearest surface segment. Figure 3 illustrates a set of points sampled via importance sampling around an object, with the component label — shown as the color corresponding to the parameterization chart — associated with each sample point.

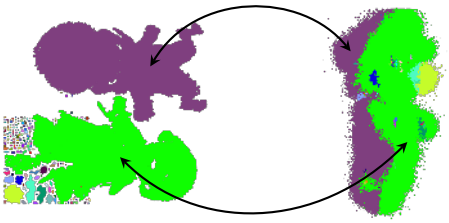


Figure 3: Decomposing the sample points according to the parameterization charts in the texture atlas, where the points in each decomposition component (right) are plotted in the color of their corresponding parameterization chart (left).

### 3.2. Designing neural implicit representations

Our neural implicit representation of the surface parameterization is a simple feed-forward MLP that takes 3D locations as input and outputs UV coordinates, composed of only fully-connected (FC) layers. We first describe the parameters of the MLP layers and then detail the overall network architecture.

*Pre-processing and layer implementation.* To improve the reconstruction quality of the MLP, we pre-process the input layer of the network with a Fourier positional encoding [5, 12] and implement the hidden layers as sinusoidal representation network (SIREN) layers [11].

Specifically, the value $p$ on each coordinate position of the input is encoded as an array of Fourier features using the following function [5]:

$$\begin{aligned} \gamma(p) = \big(&\sin(2^0\pi p), \cos(2^0\pi p), \ldots, \\ &\sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)\big), \end{aligned} \tag{8}$$

where $L$ defines the number of Fourier series terms; we use $L = 5$ in our implementation.

Our FC layers additionally differ from standard FC layers as we rely on sinusoidal representations [11]. To implement a SIREN layer, we initialize the weight matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$ using a He uniform initializer [41], before activating the layer using a sine function, as:

$$\phi(\mathbf{x}) = \sin\left(\omega_0\mathbf{W}\mathbf{x} + \mathbf{b}\right), \tag{9}$$

where $\mathbf{x}$ refers to the layer's input tensor and $\omega_0$ scales the angular frequencies. We use $\omega_0 = 1$.

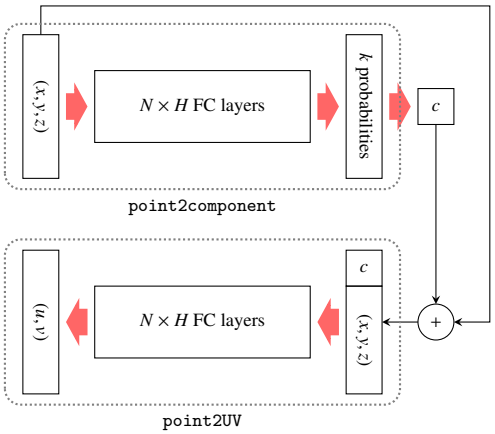



Figure 4: The network architecture of the proposed model.

*Network architecture.* We design a two-stage network architecture that handles both component predictions and UV coordinate predictions at the same time; see Figure 4 for the designed architecture of our model. We use one MLP network, termed `point2component`, to predict the component label to which the input point belongs, and another MLP network, termed `point2UV`, to predict the UV coordinate associated with the input point. Both `point2component` and `point2UV` are composed of $N$ feed-forward FC layers with each having $H$ hidden units. We set $N = 8$ and $H = 64$ in our implementation, which we found to provide a balanced trade-off between model complexity and prediction error, as per our ablation study.

The first stage, `point2component`, takes a 3D point coordinate as input and outputs a $k$-sized vector where each entry corresponds to the probability the current point belongs to a particular component. We then convert this to an integer $c$ that represents the predicted component label corresponding to the highest probability component. Then, we concatenate the component prediction $c$ with the point coordinate and use the concatenated vector as the input to `point2UV`, which outputs UV coordinate predictions. During training, `point2component` and `point2UV` are being updated in parallel separately. Once the two neural networks are trained, we use the entire concatenated architecture for predicting the surface parameterization.

Since `point2component` performs classification, its last layer uses a softmax activation and our training objective minimizes cross-entropy loss. The last layer of `point2UV` uses a sigmoid activation unit and the objective of `point2UV` is to minimize the MAE between real and predicted UV coordinates.

## 4. Results

We detail our experimental methodology before presenting various texture mapped results and a simple editing scenario. We conclude by discussing how the model is compressed. Our implementation details can be found in our code[1].

### 4.1. Experimental setup

We train our model to represent the surface parameterization of objects. To represent objects' geometry, we train for each object a model by Davies et al. [3], which we call *OverfitSDF* for brevity. OverfitSDF also uses 8 hidden layers with 64 units per layer. Here we describe the setup for training the neural implicit representations.

*Experimental data.* We use 16 3D objects from TurboSquid[2] to train our model. Every object is represented as a triangle mesh in the Wavefront `.obj` format and each has auxiliary appearance image texture file(s). We normalize object sizes to a unit bounding sphere prior to training. For each object, we generate our training data by sampling $10^6$ points in the unit sphere using the importance sampling scheme, and we split these samples into training batches of 2048 points each. For test and validation, we generate another set of $10^6$ uniformly sampled points on the object's surface.

*Optimizer and hyper-parameters.* We train all the neural networks using the Adam optimizer [42]. For `point2component` and `point2UV`, we set the learning rate to $5 \times 10^{-4}$. For OverfitSDF, the learning rate is $10^{-4}$. We train the models until convergence for 2000 epochs.

*Visualization.* We use the sphere tracing algorithm [4] to visualize our objects, where we march along camera rays by the distance to the object surface at each step location, until the current distance to the surface is smaller than a user-definable threshold $\epsilon$; our results use $\epsilon = 10^{-4}$.

*Machine configuration and timing.* We trained our neural networks on an NVIDIA GeForce RTX 2070 SUPER GPU with 8 GB of memory and CUDA version 10.1. Training our model requires an average of 1.1 h per object, roughly twice the time needed to train the geometry-only OverfitSDF model. Given $10^4$ query positions, OverfitSDF takes 0.3 s to predict the signed distances and our model takes 0.6 s to predict the UV coordinates.

[1] `https://github.com/IsaacGuan/NISP`
[2] `https://www.turbosquid.com/`

(a) OverfitColor

(b) Ours

(c) GT

Figure 5: The diffuse color of the surface sample points, with zoom-in views shown on the right of each example. Our method can preserve more high frequency details in the diffuse mapping results, compared to learning the surface color directly as a continuous function.

### *4.2. Texture mapping and editing*

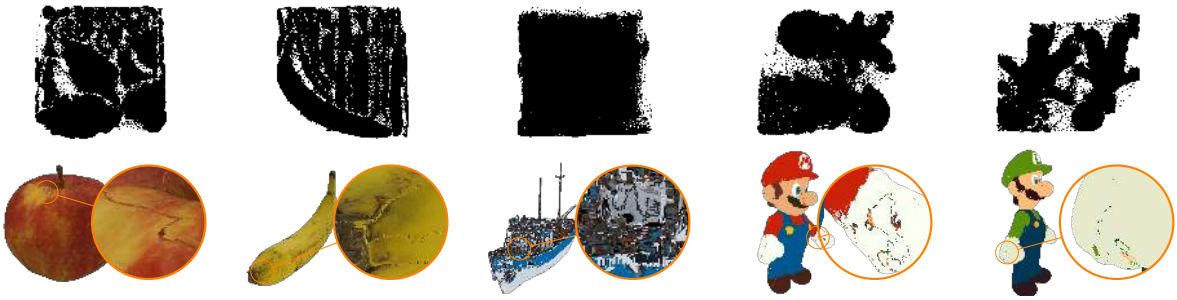
We demonstrate three proof-of-concept applications facilitated by the surface parameterization learned using our method: diffuse texture mapping, normal texture mapping, and (post-training) texture editing. We also quantitatively evaluate the distortion of our learned parameterizations and compare the texture mapping results by our method to the synthesized views by the state-of-the-art neural appearance models.

*Diffuse mapping.* To benchmark the utility and fidelity of diffuse texture mapping using our neural surface parameterization, we train an OverfitSDF-style baseline model to represent the surface color of objects as a continuous function in the 3D space — i.e., directly mapping 3D locations to RGB colors using an MLP network. We call this baseline model *OverfitColor*, and configure and train this model similarly as described before, including pre-processing the input with Fourier positional encoding. As shown in Figure 5, we compare the color on the surface sample points generated by diffuse mapping using our model (Figure 5b) to those learned by OverfitColor (Figure 5a) and to the ground truth (GT) color (Figure 5c). We clearly observe that our results more faithfully preserve the all-frequency texture detail, such as the lettering on the building's shop sign.

To illustrate the advantages of conditioning the input of neural networks with spatial decomposition when learning surface parameterization signals, we train a `point2UV` network without the conditioned inputs provided by `point2component`, i.e., mapping 3D locations directly to UV coordinates. Figure 6 compares the UV coordinates predicted by `point2UV` with and without conditioned inputs, and we observe — from the scatter charts of the UV coordinates — that, without input conditions, `point2UV` fails to provide accurate predictions in regions close to the boundary of each parameterization chart. This results in visual artifacts during, e.g., diffuse texture mapping, such as distortions on the boat and the anomalies on the characters' hands (Figure 6a). With conditioned inputs, however, these distortions are largely mitigated (Figure 6b), resulting in almost identical diffuse mapping results compared to the GT parameterization.

*Normal mapping.* The surface parameterization our model learns also allows users to easily apply normal maps to object surfaces implicitly defined by the SDF, adding more visual details to the surface that a neural implicit surface cannot capture. Figure 7 illustrates our rendered results using normal mapping on neural implicit surfaces: here, we can easily add geometric detail, such as the barrel's wood grain and the house's brick wall patterns (Figure 7b), using content creation paradigms familiar to digital artists.

*Texture editing.* As our model learns directly from the original texture space, users can easily edit original tex-



(a) `point2UV`

(b) Ours (`point2component` + `point2UV`)

(c) GT

Figure 6: The UV coordinates of the surface sample points, plotted as scatter charts, with the corresponding diffuse mapping results along with the zoom-in views shown on the bottom of each example. With conditioned input to `point2UV`, the noisy results in the UV predictions can be largely mitigated, resulting in almost identical diffuse mapping results to the GT.

ture images to change the appearance of the neural implicit surface *after* it has been trained. Figure 8 demonstrates examples of editing the diffuse maps applied to our neural implicit surfaces, where we apply a photo filter to the texture image of the watermelon and paint letters on the roof of the van.

*Distortion metric.* We quantitatively evaluate the distortion of the learned parameterizations according to the metric described by Sorkine et al. [43]. Given the affine mapping between the original triangle mesh and its corresponding parameterization, the distortion $\delta$ caused to each triangle is measured using the largest and smallest singular values $\sigma_{\max}$ and $\sigma_{\min}$ of the Jacobian matrix of this transformation, written as:

$$\delta = \max\left(\sigma_{\max}, \frac{1}{\sigma_{\min}}\right). \tag{10}$$

We feed the vertices of objects' triangle meshes into the neural networks to generate the parameterized meshes. For each parameterization, we compute the mean value $\delta_{\text{mean}}$, the maximum value $\delta_{\max}$, and the standard deviation $\delta_{\text{std}}$ of the distortions caused to all triangles in the mesh. Table 1 demonstrates this quantitative evaluation for the learned and GT parameterizations in Figure 6. We see how much the distortion is

(a) Without normal mapping

(b) With normal mapping

Figure 7: Application of normal maps to neural implicit surfaces.

(a) Original diffuse maps (b) Edited diffuse maps

Figure 8: Editing the diffuse maps applied to neural implicit surfaces, with the rendered results shown on the right side of each diffuse map.

improved using conditioned input of our model.

*Model comparison.* We compare our texture mapping results to the synthesized views by two different neural appearance models, namely, the scene representation network (SRN) [16] and NeRF [5]. Figure 9 illustrates this comparison. We further compute the mean squared error (MSE), peak signal-to-noise ratio (PSNR), and structural similarity index (SSIM) between GT and the synthesized views as well as our rendered results, as summarized in Table 2. We can see that our method outperforms these neural appearance models in preserving high frequency geometry and appearance details. Both SRN and NeRF require a higher GPU memory usage, as compared to our method. Using our experimental device (a GPU with 8 GB of memory) and to train the

Table 1: Average $\delta_{\text{mean}}$, $\delta_{\text{max}}$, and $\delta_{\text{std}}$ of the learned and GT parameterizations in Figure 6.

| | $\delta_{\text{mean}}$ | $\delta_{\text{max}}$ | $\delta_{\text{std}}$ |
|---|---|---|---|
| `point2UV` | 13.63 | $6.80 \times 10^4$ | 284.89 |
| Ours | 9.60 | $9.96 \times 10^3$ | 60.13 |
| GT | 3.58 | $6.14 \times 10^3$ | 17.84 |

(a) SRN (b) NeRF (c) Ours (d) GT

Figure 9: Comparison between the texture mapping results by our method and the synthesized views by neural appearance models.

Table 2: Average MSE, PSNR, and SSIM of the synthesized views and our rendered results in Figure 9.

| | MSE | PSNR | SSIM |
|---|---|---|---|
| SRN | 40.10 | 32.18 | 0.947 |
| NeRF | 17.95 | 35.63 | 0.976 |
| Ours | 2.95 | 43.85 | 0.996 |

models at a reasonable speed, we have to reduce the training views to a lower resolution (e.g., $128 \times 128$). We train both models until convergence for 200 000 epochs, taking around 8 h for SRN and 4 h for NeRF. However, our method incurs substantially smaller GPU memory and computational cost, with a quicker training time. With the auxiliary texture maps, we can render our texture mapping results using traditional rendering pipelines at arbitrary high resolutions (e.g., $1024 \times 1024$ or higher).

### *4.3. Model compression*

We can also interpret our model along with OverfitSDF as a compression strategy for large 3D objects represented as meshes. The average size of the `.obj` files of the 16 experimental objects is 13.4 MB, while the network weights of our model require 332 kB per object and the weights of OverfitSDF require 155 kB per object, which total to 487 kB for the entire geometry and surface parameterization representation. This

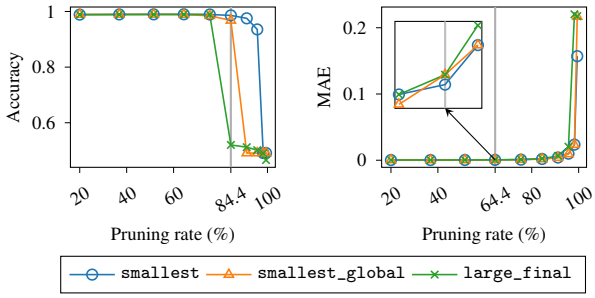
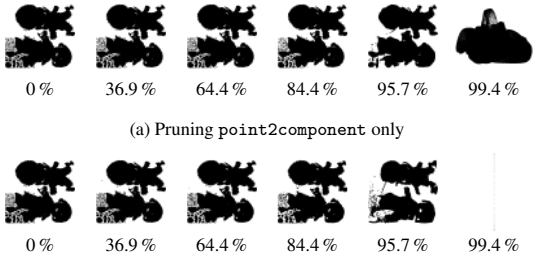



Figure 10: Test results when pruning the model using different strategies, with the $x$-axes presenting the pruning rates (in percentage) and $y$-axes being the test accuracy of `point2component` (left) and the test MAE of `point2UV` (right).




(a) Pruning `point2component` only

(b) Pruning `point2UV` only

Figure 11: The predicted UV coordinates (shown as scatter charts) of the Mario object by our model with `point2component` and `point2UV` pruned at different rates (shown in percentage below each scatter chart) using the `smallest` strategy.

already yields a compression rate of 1 : 27, but can be further compressed by applying the lottery ticket method [38], i.e., by seeking out sparse trainable subnetworks — called winning tickets — from the originally trained neural model. These (much more compact) subnetworks reach the same test accuracy as the original network.

Following the lottery ticket training methodology [38], we identify winning tickets iteratively, i.e., in each training epoch, pruning and freezing $q^{\frac{1}{t}}$ % of the weights that survive the previous epoch from the original model (trained for $t$ epochs), to obtain a final pruning rate of $q$ %. We use three strategies to decide the weights to be pruned or kept: (i) prune the smallest magnitude weights at each prunable layer; (ii) prune the smallest weights across all prunable layers; (iii) keep the weights that have the largest magnitude from a pre–trained model [44]. We refer to these three strategies as `smallest`, `smallest_global`, and `large_final`.

We demonstrate the results of these pruning strategies on the Mario object, where `point2component` and `point2UV` have 43 454 and 31 874 trainable weights, respectively. We use 10 pruning rates ranging from 20 % to 99 % and use the test accuracy and MAE as evaluation metrics for `point2component` and `point2UV`. Figure 10 summarizes our findings: for `point2component`, the test accuracy does not drop significantly until a pruning rate of 84.4 %, and for `point2UV`, the test MAE stays around $5 \times 10^{-4}$ until a pruning rate of 64.4 %, using the pruning strategy `smallest`. Figure 11 visualizes the UV predictions made by the pruned networks as scatter charts, where we see the predictions will not distort remarkably until pruning `point2component` at 84.4 % or pruning `point2component` at 64.4 %.

Table 3: Size comparison between the weights, saved in `.npz` format and measured in kilobytes, of the unpruned model and the pruned model compressed using different coding formats of sparse matrices.

| | Unpruned | COO | CSC | CSR | DIA |
|---|---|---|---|---|---|
| `point2component` | 174.4 | 54.9 | 54.5 | 55.0 | 58.5 |
| `point2UV` | 129.1 | 71.3 | 71.4 | 71.4 | 73.4 |
| Total | 303.5 | 126.2 | 125.9 | 126.4 | 131.9 |

We can save the weights of the pruned networks as sparse matrices, as pruned weights are set to zero. Table 3 compares the sizes of the weights of the unpruned and pruned models, where `point2component` is pruned at 84.4 % and `point2UV` is pruned at 64.4 % using the `smallest` strategy. The pruned model is compressed using different coding formats of sparse matrices, namely, the coordinate format (COO), the compressed sparse column format (CSC), the compressed sparse row format (CSR), and the diagonal storage format (DIA). We observe that our model can be further reduced from 303.5 kB to roughly 126 kB.

## 5. Conclusion

We proposed to learn neural representations of both an object's 3D surface *and* a surface parameterization suitable for auxiliary appearance data. Following recent works on overfitted networks, we learn complex multi-chart signal parameterizations as a weight-encoded neural representation. We rely on a novel two-stage network architecture to allow us to capture fine-scale texture domain resolution while respecting the discontinuities in texture atlases. Our model augments existing (neural) implicit surface representations with the benefits of auxiliary texture mapping and editing common to standard 3D digital content creation pipelines. We demonstrated the applicability of our model to appearance-aware neural implicit surface representations, building atop the work of Davies et al. [3].

## Acknowledgments

We thank the anonymous reviewers for their valuable comments and suggestions. This work was done during the first author's internship at Huawei Technologies Canada.

## References

[1] Park, JJ, Florence, P, Straub, J, Newcombe, R, Lovegrove, S. DeepSDF: Learning continuous signed distance functions for shape representation. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2019, p. 165–174.

[2] Chibane, J, Mir, A, Pons-Moll, G. Neural unsigned distance fields for implicit function learning. In: Adv. Neural Inform. Process. Syst. 2020, p. 21638–21652.

[3] Davies, T, Nowrouzezahrai, D, Jacobson, A. On the effectiveness of weight-encoded neural implicit 3D shapes. CoRR 2020;abs/2009.09808.

[4] Hart, JC. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. Vis Comput 1996;12(10):527–545.

[5] Mildenhall, B, Srinivasan, PP, Tancik, M, Barron, JT, Ramamoorthi, R, Ng, R. NeRF: Representing scenes as neural radiance fields for view synthesis. In: Proc. Eur. Conf. Comput. Vis. 2020, p. 405–421.

[6] Sheffer, A, Praun, E, Rose, K. Mesh parameterization methods and their applications. Found Trends Comput Graph Vis 2006;2(2):105–171.

[7] Mescheder, L, Oechsle, M, Niemeyer, M, Nowozin, S, Geiger, A. Occupancy networks: Learning 3D reconstruction in function space. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2019, p. 4455–4465.

[8] Chen, Z, Zhang, H. Learning implicit fields for generative shape modeling. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2019, p. 5932–5941.

[9] Deng, B, Genova, K, Yazdani, S, Bouaziz, S, Hinton, G, Tagliasacchi, A. CvxNet: Learnable convex decomposition. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2020, p. 31–41.

[10] Chen, Z, Tagliasacchi, A, Zhang, H. BSP-Net: Generating compact meshes via binary space partitioning. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2020, p. 42–51.

[11] Sitzmann, V, Martel, JNP, Bergman, AW, Lindell, DB, Wetzstein, G. Implicit neural representations with periodic activation functions. In: Adv. Neural Inform. Process. Syst. 2020, p. 7462–7473.

[12] Tancik, M, Srinivasan, PP, Mildenhall, B, Fridovich-Keil, S, Raghavan, N, Singhal, U, et al. Fourier features let networks learn high frequency functions in low dimensional domains. In: Adv. Neural Inform. Process. Syst. 2020, p. 7537–7547.

[13] Takikawa, T, Litalien, J, Yin, K, Kreis, K, Loop, C, Nowrouzezahrai, D, et al. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2021, p. 11353–11362.

[14] Saito, S, Huang, Z, Natsume, R, Morishima, S, Kanazawa, A, Li, H. PIFu: Pixel-aligned implicit function for high-resolution clothed human digitization. In: Proc. IEEE Int. Conf. Comput. Vis. 2019, p. 2304–2314.

[15] Oechsle, M, Mescheder, L, Niemeyer, M, Strauss, T, Geiger, A. Texture fields: Learning texture representations in function space. In: Proc. IEEE Int. Conf. Comput. Vis. 2019, p. 4530–4539.

[16] Sitzmann, V, Zollhöfer, M, Wetzstein, G. Scene representation networks: Continuous 3D-structure-aware neural scene representations. In: Adv. Neural Inform. Process. Syst. 2019, p. 1121–1132.

[17] Niemeyer, M, Mescheder, L, Oechsle, M, Geiger, A. Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2020, p. 3501–3512.

[18] Oechsle, M, Niemeyer, M, Reiser, C, Mescheder, L, Strauss, T, Geiger, A. Learning implicit surface light fields. In: Proc. Int. Conf. 3D Vis. 2020, p. 452–462.

[19] Xiang, F, Xu, Z, Hašan, M, Hold-Geoffroy, Y, Sunkavalli, K, Su, H. NeuTex: Neural texture mapping for volumetric neural rendering. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2021, p. 7115–7124.

[20] Chen, Z, Yin, K, Fidler, S. AUV-Net: Learning aligned UV maps for texture transfer and synthesis. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2022, p. 1455–1464.

[21] Tutte, WT. Convex representations of graphs. Proc London Math Soc 1960;s3-10(1):304–320.

[22] Sheffer, A, de Sturler, E. Parameterization of faceted surfaces for meshing using angle-based flattening. Eng Comput 2001;17(3):326–337.

[23] Sheffer, A, Lévy, B, Mogilnitsky, M, Bogomyakov, A. ABF++: Fast and robust angle based flattening. ACM Trans Graph 2005;24(2):311–330.

[24] Lévy, B, Petitjean, S, Ray, N, Maillot, J. Least squares conformal maps for automatic texture atlas generation. ACM Trans Graph 2002;21(3):362–371.

[25] Sander, PV, Snyder, J, Gortler, SJ, Hoppe, H. Texture mapping progressive meshes. In: Proc. SIGGRAPH. 2001, p. 409–416.

[26] Zigelman, G, Kimmel, R, Kiryati, N. Texture mapping using surface flattening via multidimensional scaling. IEEE Trans Vis Comput Graph 2002;8(2):198–207.

[27] Degener, P, Meseth, J, Klein, R. An adaptable surface parameterization method. In: Proc. Int. Meshing Roundtable. 2003, p. 201–213.

[28] Sawhney, R, Crane, K. Boundary first flattening. ACM Trans Graph 2018;37(1):5:1–5:14.

[29] Garland, M, Willmott, A, Heckbert, PS. Hierarchical face clustering on polygonal surfaces. In: Proc. Symp. Interactive 3D Graph. 2001, p. 49–58.

[30] Sinha, A, Bai, J, Ramani, K. Deep learning 3D shape surfaces using geometry images. In: Proc. Eur. Conf. Comput. Vis. 2016, p. 223–240.

[31] Groueix, T, Fisher, M, Kim, VG, Russell, BC, Aubry, M. A papier-mâché approach to learning 3D surface generation. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2018, p. 216–224.

[32] Deprelle, T, Groueix, T, Fisher, M, Kim, VG, Russell, BC, Aubry, M. Learning elementary structures for 3D shape generation and matching. In: Adv. Neural Inform. Process. Syst. 2019, p. 7435–7445.

[33] Morreale, L, Aigerman, N, Kim, V, Mitra, NJ. Neural surface maps. In: Proc. IEEE Conf. Comput. Vis. Pattern Recog. 2021, p. 4637–4646.

[34] Dupont, E, Goliński, A, Alizadeh, M, Teh, YW, Doucet, A. COIN: Compression with implicit neural representations. CoRR 2021;abs/2103.03123.

[35] Strümpler, Y, Postels, J, Yang, R, Gool, LV, Tombari, F. Implicit neural representations for image compression. In: Proc. Eur. Conf. Comput. Vis. 2022, p. 74–91.

[36] Li, Y, Liu, Y, Lu, Y, Zhang, S, Cai, S, Zhang, Y. High-fidelity 3D model compression based on key spheres. In: Proc. Data Compression Conf. 2022, p. 292–301.

[37] Dupont, E, Loya, H, Alizadeh, M, Goliński, A, Teh, YW, Doucet, A. COIN++: Neural compression across modalities. Trans Mach Learn Res 2022;2022(11).

[38] Frankle, J, Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: Proc. Int. Conf. Learn. Represent. 2019,.

[39] Kahn, H, Harris, TE. Estimation of particle transmission by random sampling. In: Monte Carlo Method; vol. 12 of *National Bureau of Standards Applied Mathematics Series*. 1951, p. 27–30.

[40] Sohn, K, Lee, H, Yan, X. Learning structured output representation using deep conditional generative models. In: Adv. Neural Inform. Process. Syst. 2015, p. 3483–3491.

[41] He, K, Zhang, X, Ren, S, Sun, J. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In: Proc. IEEE Int. Conf. Comput. Vis. 2015, p. 1026–1034.

[42] Kingma, DP, Ba, JL. Adam: A method for stochastic optimization. In: Proc. Int. Conf. Learn. Represent. 2015,.

[43] Sorkine, O, Cohen-Or, D, Goldenthal, R, Lischinski, D. Bounded-distortion piecewise mesh parameterization. In: Proc. IEEE Visualization Conf. 2002, p. 355–362.

[44] Zhou, H, Lan, J, Liu, R, Yosinski, J. Deconstructing lottery tickets: Zeros, signs, and the supermask. In: Adv. Neural Inform. Process. Syst. 2019, p. 3597–3607.