Rendering Neural Materials on Curved Surfaces

Alexandr Kuznetsov
UC San Diego
La Jolla, CA, USA
a1kuznet@eng.ucsd.edu

Xuezheng Wang
UC San Diego
La Jolla, CA, USA
xuw005@ucsd.edu

Krishna Mullia
Adobe Research
San Francisco, CA, USA
mulliala@adobe.com

Fujun Luan
Adobe Research
San Jose, CA, USA
fluan@adobe.com

Zexiang Xu
Adobe Research
San Jose, CA, USA
zexu@adobe.com

Miloš Hašan
Adobe Research
San Jose, CA, USA
mihasan@adobe.com

Ravi Ramamoorthi
UC San Diego
La Jolla, CA, USA
ravir@cs.ucsd.edu

Figure 1: Five different neural materials applied to curved bounding geometries. The materials are treated as *silhouette bidirectional texture functions* (SBTFs), are learned from synthetic microstructure data, and represented using a collection of feature textures and small fully connected neural networks. The key to our SBTF representation, compared to traditional BTFs, is considering surface curvature, and handling rays that hit the bounding geometry but miss the material microstructure. Our method accurately handles silhouette effects and matches ground truth (GT), unlike NeuMIP [Kuznetsov et al. 2021], which fundamentally cannot consider silhouette effects and simply renders the base mesh; see insets. Note that no displacement mapping nor ray marching is ever needed.

ABSTRACT

Neural material reflectance representations address some limitations of traditional analytic BRDFs with parameter textures; they can theoretically represent any material data, whether a complex synthetic microgeometry with displacements, shadows and interreflections, or real measured reflectance. However, they still approximate the material on an infinite plane, which prevents them from correctly handling silhouette and parallax effects for viewing directions close to grazing. The goal of this paper is to design a neural material representation capable of correctly handling such silhouette effects. We extend the neural network query to take surface curvature information as input, while the query output is extended to return a transparency value in addition to reflectance. We train the new neural representation on synthetic data that contains queries spanning a variety of surface curvatures. We show an ability to accurately represent complex silhouette behavior that would traditionally require more expensive and less flexible techniques, such as on-the-fly geometry displacement or ray marching.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**; **Ray tracing**; **Neural networks**; **Image processing**.

KEYWORDS

Materials, BTF, BRDF, Neural, Silhouette, Curvature

ACM Reference Format:
Alexandr Kuznetsov, Xuezheng Wang, Krishna Mullia, Fujun Luan, Zexiang Xu, Miloš Hašan, and Ravi Ramamoorthi. 2022. Rendering Neural Materials on Curved Surfaces. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings (SIGGRAPH '22 Conference Proceedings), August 7–11, 2022, Vancouver, BC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3528233.3530721

SIGGRAPH '22 Conference Proceedings, August 7–11, 2022, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9337-9/22/08.
https://doi.org/10.1145/3528233.3530721

1 INTRODUCTION

Computer graphics has been delivering a steadily increasing level of realism over the past decades, much of which owes to physically realistic materials. The typical material representation in computer graphics is a physically-based analytic microfacet BRDF, with spatially varying parameters controlled by texture maps and normal maps. This representation has been successful in practice, but has several limitations. The microfacet assumption does not always apply , and normal mapping cannot handle significant displacements, occlusions or inter-reflections, while displacement mapping adds significant cost. Traditional mipmapping hierarchies are limited in accurately representing multi-resolution appearance.

Neural material reflectance representations are a recent effort to remove these limitations by using a neural architecture instead of an analytic BRDF with traditional parameter textures [Rainer et al. 2019], [Rainer et al. 2020], [Kuznetsov et al. 2021], [Fan et al. 2021]. Neural approaches can theoretically represent any material data, whether a complex synthetic microgeometry with displacements, shadows and inter-reflections, or a real measured reflectance function. However, they still have the limitation of assuming the input material is defined on an infinite plane, which prevents them from correctly handling silhouette and parallax effects for viewing directions close to grazing.

Our goal is to represent materials with correct appearance when applied to arbitrary surface geometries. If a material microstructure is applied to a curved surface, different query rays with grazing directions may hit or miss the microstructure, leading to a complex silhouette boundary that is very important for the illusion of material complexity. This effect has been, until now, only possible with expensive techniques: actual geometric displacement mapping [Thonat et al. 2021], or mapping volumes around the surfaces that need to be rendered with ray-marching [Baatz et al. 2021].

The goal of this paper is to design a neural material representation capable of correctly handling grazing and silhouette effects. To achieve this, we add an explicit concept of surface curvature and transparency to the neural model. We extend the neural network query in NeuMIP [Kuznetsov et al. 2021] to take surface curvature information as input, and its output is extended to return a transparency value in addition to reflectance. These changes to the architecture are not by themselves sufficient; we also need to change the training strategy. Unlike previous work, which is trained on synthetic mesostructures applied to infinite planes, we need to train the new neural representation on a dataset that contains queries spanning a variety of surface curvatures. The key contributions of this paper are as follows:

- We introduce the first spatially-varying material representation handling silhouette effects using a single query, i.e. without any on-the-fly geometry displacement or ray marching, based on the concept of Silhouette BTF (Sec. 4.1).
- Building upon the NeuMIP architecture, we achieve this by explicitly considering curvature as part of the material query (Sec. 4.2), and by considering transparency as part of the query output (Sec. 4.3).
- Training on datasets designed to learn the correct behavior across surfaces with varying curvatures (Sec. 4.4) and integration into a practical renderer (Sec. 4.5).

We show an ability to accurately represent material complexity that would traditionally require more expensive and less flexible techniques. For example, Fig. 1 shows our silhouette accuracy compared to NeuMIP, which fundamentally cannot render silhouette effects and simply renders the base mesh. Our model can be integrated in practical offline and interactive rendering applications. Our code is available at https://cseweb.ucsd.edu/~viscomp/projects/CurvedNeuralMaterials/.

2 RELATED WORK

Displacement mapping and its variations. Displacement mapping is a powerful technique to add material complexity to surface geometries, producing realistic parallax, silhouette and shadowing effects. These benefits however come at a high computational cost. In typical GPU rasterization pipelines, and in the classical Reyes pipeline, displacement is implemented by generating geometry that can be immediately drawn into the framebuffer and does not need to be stored. In modern ray-tracing based renderers, such an approach is not feasible and the standard solution has been to implement displacement through tessellation of the base geometry; as expected, this is expensive both in terms of storage and computation. Several approximations to displacement mapping have been introduced. Parallax mapping [Kaneko et al. 2001; Oliveira and Policarpo 2005; Wang et al. 2005a] is a classic technique for improving bump and normal mapping by adding an approximate parallax effect, by estimating a texture space offset based on the height map and normal map. Our neural offset module is inspired by this technique. [Oliveira et al. 2000; Policarpo et al. 2005] introduced the idea of relief mapping, which use a root finding algorithm to find the ray intersection with the displaced surface. See [Szirmay-Kalos and Umenhoffer 2008] for an overview of GPU-based displacement mapping and approximation techniques.

Wang et al. [2003] proposed view-dependent displacement mapping (VDM), a method which precomputes the view-dependent distance to surface displacements. It uses curvature along the ray at the intersection point to approximate the local surface shape. It uses tabulation together with SVD compression to store the displacement values across positions, directions, and curvatures. Although VDM can render silhouettes, it is a purely geometric solution that is specialized to heightfields, and does not precompute reflectance effects like inter-reflections. Our method makes no such assumptions and can work without heightfields and ground truth displacements. (See Figure 2 in the supplemental materials.)

Wang et al. [2004] introduced generalized displacement maps (GDM), a 5D function which predicts the distance to an intersection given a 3D location and a viewing direction. Wang et al. [2005b] introduced a 4D mesostructure distance function (MDF) given a reference plane. Silhouette effects are rendered using depth peeling, which is not easily applicable in a path-tracing framework. These methods are related to our approach; however they are fundamentally geometric and heightfield-based, and do not consider the reflectance effects emerging from the detailed material geometry.

Shell mapping [Porumbescu et al. 2005] is a general way to treat any ray-traceable microgeometry (with its own acceleration structure, if required) as a material that can be mapped onto any

mesh. It builds a thin volumetric layer of tetrahedral elements around the base mesh, and applies an affine warp within each element. This method can be used for displacement mapping, or mapping any other ray-traceable primitives onto meshes. Recently, Thonat et al. [2021] introduced a tessellation-free displacement mapping technique that works with ray-tracing and is similar to shell mapping but achieves higher performance and robustness specifically for displacement. However, only storage is saved, not computation; the entire complex displaced geometry is still being explicitly intersected by rays.

Our method instead encodes displacement effects into a neural representation that can be evaluated using a single query per shading point. While displacement is a common and important effect, our method is not limited to displacement, and could handle microstructures not expressible as heightfields.

BTFs. Bidirectional texture functions (BTFs) have been introduced by Dana et al. [1999]. A BTF is a 6D function describing arbitrary reflective surface appearance; a BTF outputs a reflectance value given a spatial location, incoming and outgoing direction. Storing a discretized 6D function is very expensive; therefore, several compression methods for BTFs have been explored [Filip and Haindl 2008].

Neural reflectance. Rainer et al. [2019] introduce a neural architecture based on an autoencoder framework to compress BTF slices per texel (also termed apparent BRDFs or ABRDFs); the decoder takes incoming/outgoing directions as input in addition to the latent vector, and the autoencoder is trained per BTF. Later, they extended the work by unifying different materials into a shared latent space, so only a single autoencoder needs to be trained [Rainer et al. 2020]. [Kuznetsov et al. 2019] used Generative Adversarial Networks (GANs) to generate reflectance functions perceptually similar to synthetic or measured input data, and rendered them using partial evaluation of the generator network.

NeuMIP [Kuznetsov et al. 2021] is a recent method that we build upon. It uses a set of learned power-of-2 feature textures to represent the material at different levels, combined with a fixed per-material MLP (multi-layer perceptron, i.e. a fully connected neural network). This method can theoretically fit any material data, whether a complex synthetic microgeometry, or a measured BTF. However, it still has the limitation of only considering the input material on an infinite plane, which prevents it from correctly handling silhouette and parallax effects for viewing directions close to grazing.

We build on the neural architecture of NeuMIP with several modifications designed to remove its limitations regarding silhouette effects. However, for simplicity, we decided not to focus on the multi-resolution component of NeuMIP. More precisely, this means that we use 2D feature textures with bilinear interpolation, and do not consider pyramids of feature textures with trilinear interpolation; we consider it orthogonal and believe it would be straightforward to combine with our method.

Thin volumetric shells. Previous research explored representing materials as thin volumetric layers, wrapped around a base surface mesh. Dufort et al. [2005] approximate material mesostructure using a thin shell of semi-transparent 3D texture and use ray-marching to compute the color. Volumetric fabric models based on data scanned using micro-CT imaging were introduced by Zhao et al. [2014]; the volumetric grid was wrapped onto a curved surface using shell mapping. Recently, Baatz et al. [2021] introduced a neural material approach using volumetric layers. The material is represented as a volumetric medium with a density and a reflectance function defined at every point, encoded in a single fully-connected neural network for the entire material. This approach is capable of high quality, including correct silhouette and parallax effects. However, they do still require ray marching with multiple queries of a neural architecture (larger than ours) needed per ray. On the other hand, our method is less expensive per shading operation, as it only requires a single SBTF query per surface intersection point, and can use a smaller neural network combined with feature textures.

3 BACKGROUND

In this section, we will first present a short overview of bidirectional texture functions (BTFs) (Sec. 3.1). Next, we will summarize the concept of local surface curvature. (Sec. 3.2).

3.1 Overview of BTFs

Bidirectional texture functions (BTFs) [Dana et al. 1999] are a general representation of surface reflectance. They take as input the camera direction $\boldsymbol{\omega_o}$, light direction $\boldsymbol{\omega_i}$, and texture coordinates $\mathbf{u}$, and output a reflectance value (color). More precisely, the BTF value $\rho(\mathbf{u}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})$ can be seen as the exitant radiance from point $\mathbf{u}$ in direction $\boldsymbol{\omega_o}$, when the material is lit by distant directional light coming from direction $\boldsymbol{\omega_i}$, with unit irradiance onto the material plane. If a material is simply a homogeneous BRDF, its BTF equals the BRDF; thus a BTF is a generalization of a BRDF. It incorporates all effects from the material microstructure, including parallax, shadowing, and multiple scattering, and is generally non-reciprocal.

BTFs have traditionally been used to capture physical material samples by taking photographs under many viewing and light directions, but have also been used with synthetic data (like in our paper). Many methods have been used to compress BTFs, including several neural approaches. However, none of the previous BTF compression and representation methods have the ability to handle grazing/silhouette effects; the goal of our method is to accurately capture these effects.

3.2 Local surface curvature

The curvature of a 2D surface at point $\boldsymbol{p}$, in a direction $\mathbf{x}$ that lies within the tangent plane (i.e. is orthogonal to the surface normal $\mathbf{n}$), is defined as the reciprocal of the radius of the osculating circle, which is the circle that locally aligns with the surface at $\boldsymbol{p}$ within the plane given by direction $\mathbf{x}$ and the surface normal $\mathbf{n}$. Positive and negative curvatures represent convex and concave surfaces, respectively. The curvature at $\boldsymbol{p}$ can be fully represented by two principal directions $\mathbf{x}_1, \mathbf{x}_2$ and the associated principal curvatures along those directions k_1, k_2. Namely, the curvature $k(\mathbf{x})$ for any direction $\mathbf{x}$ in the tangent plane can be computed as

$$k(\mathbf{x}) = k_1(\mathbf{x}_1 \cdot \mathbf{x})^2 + k_2(\mathbf{x}_2 \cdot \mathbf{x})^2. \quad (1)$$

The principal directions are orthogonal (i.e. $\mathbf{x}_1 \cdot \mathbf{x}_2 = 0$), unless $k_1 = k_2$, in which case we can still choose them as orthogonal. The principal curvatures are the bounds of the range of curvatures over all other tangent directions $\mathbf{x}$ [cur 2022].

4 NEURAL SILHOUETTE BTF

In this section, we first define our new Silhouette BTF (SBTF) in Sec. 4.1, and discuss the curvature input to the SBTF (Sec. 4.2). We will then introduce our network architecture (Sec. 4.3), designed to accurately fit our SBTFs. Finally, we will discuss how to fit our model to data (Sec. 4.4) and use it during rendering (Sec. 4.5).

4.1 Silhouette BTF

In this subsection, we define our Silhouette BTF (SBTF). There are two key differences from a traditional BTF: one extra input (the curvature value along the projected ray direction as described above), and one extra output, an opacity value for the incoming ray.

In our model, we apply the material to a *bounding geometry*, that is, we assume that the represented material is fully contained under the bounding surface; see Figure 3. Therefore, rays that miss the bounding geometry will also miss the material. However, rays that hit the bounding geometry may or may not hit the material: sometimes a grazing ray passes through the geometry unaffected by the material, causing a complex silhouette effect.

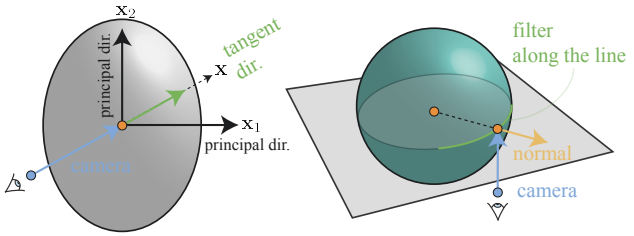
For this reason, our SBTF will also output an opacity value $\alpha \in [0, 1]$. If $\alpha = 1$, then the ray hit the material and should reflect, while if $\alpha = 0$, the material is missed, and the ray continues. Values between 0 and 1 can occur and should be treated carefully, as discussed in more detail in Sec. 4.5.

We also provide the local curvature k along the projected ray direction to the SBTF. The precise way to compute this curvature value is detailed in the next subsection.

Thus the SBTF $S(\mathbf{u}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}, k)$ can be written as two components, reflectance and transparency:

$$S(\mathbf{u}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}, k) = (\rho(\mathbf{u}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}, k), \alpha(\mathbf{u}, \boldsymbol{\omega_o}, k)) \quad (2)$$

where the reflectance value ρ is interpreted as the outgoing radiance in direction $\boldsymbol{\omega_o}$, when lit by unit directional light from direction $\boldsymbol{\omega_i}$. In a slight difference compared to the BRDF/BTF definition, we are incorporating the cosine term in the SBTF. Furthermore,



(a) Our tangent curvature (b) Filter along the line

Figure 2: (a) Our SBTF query takes as input the curvature along the projected viewing ray direction. (b) We filter the curvature by taking the minimum curvature along a surface curve in the ray direction past the shading point.

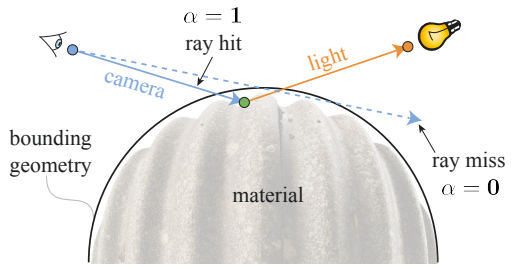


Figure 3: Illustration of our Silhouette BTF queries. The material is defined under the bounding geometry. If a ray hits the bounding geometry, it may or may not also hit the underlying material, as specified by the returned transparency value. If the material is hit, the SBTF query returns the reflectance value given the light direction.

the SBTF can be non-zero for light directions $\boldsymbol{\omega_i}$ that are facing away from the bounding geometry intersection point's normal, even for opaque surfaces. This cannot happen on an infinite plane, which always blocks light directions below the horizon, and is a consequence of considering curved surfaces in our SBTF. In other words, the reflected radiance L at point $\mathbf{u}$ in direction $\boldsymbol{\omega_o}$ can be written as a spherical integral of incoming radiance weighted by the SBTF:

$$L(\mathbf{u} \rightarrow \boldsymbol{\omega_o}) = \alpha(\mathbf{u}, \boldsymbol{\omega_o}, k) \int_{S^2} \rho(\mathbf{u}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}, k) L(\mathbf{u} \leftarrow \boldsymbol{\omega_i})\, d\boldsymbol{\omega_i}. \quad (3)$$

Our remaining tasks are: defining a neural model to fit SBTFs, generating samples of the SBTF from a given material and learning the model weights to fit them, and finally rendering using the learned model. We will cover them in the following subsections.

4.2 Curvature along the ray

To compute a curvature value along the viewing ray, our approach is to first precompute principal directions and curvatures for every vertex, using the IGL library [Jacobson et al. 2018]. For a given vertex $\boldsymbol{p}$ and a camera ray direction $\boldsymbol{\omega_o}$, we can project it into the corresponding tangent $\mathbf{x}(\boldsymbol{\omega_o})$. We then compute the curvature value $k(\mathbf{x}(\boldsymbol{\omega_o}))$ using Eq. (1); see also Fig. 2 (a).

$$k = k(\mathbf{x}(\boldsymbol{\omega_o})) = k(\text{normalize}(\boldsymbol{\omega_o} - \mathbf{n}(\boldsymbol{\omega_o} \cdot \mathbf{n}))). \quad (4)$$

We provide k to the SBTF evaluation as an additional input. For a general point $\boldsymbol{p}$ on any triangle (not necessarily a vertex), we interpolate the computation from the triangle's vertices.

Furthermore, instead of only using the value of curvature at the shading point, we filter the value by taking the minimum curvature along a short curve on the bounding geometry past the shading point, in the plane defined by the incoming ray and normal; see Fig. 2 (b). This is because we are interested in the curvature of the entire silhouette region, instead of just the shading point.

4.3 Network architecture

Our model learns the SBTF $S(\mathbf{u}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}, k)$ by fitting it to a dataset of input-output queries. Our architecture consists of three modules (see Figure 4):

- **Alpha**$(\mathbf{u}, \boldsymbol{\omega_o}, k)$: opacity prediction module,

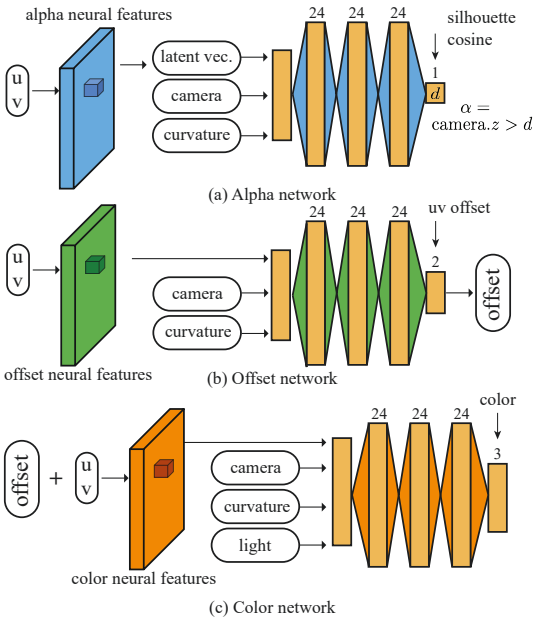


Figure 4: Our network architecture, described in Sec. 4.3. It consists of 3 parts: opacity predicting module, neural offset module and reflectance prediction module. The compact design of the networks results in fast evaluation.

- **Offset**$(\mathbf{u}, \boldsymbol{\omega_o}, k)$: neural offset module, and
- **Color**$(\mathbf{u}, \boldsymbol{\omega_o}, \boldsymbol{\omega_i}, k)$: reflectance prediction module.

Each module consists of a neural feature texture and a small fully connected network (multi-layer perceptron, MLP). Each feature texture is a 2-dimensional grid of feature vectors that can be bilinearly interpolated. For all MLPs we use the ReLU activation function, except for the last layer. In each module, we use 3 hidden layers with 24 channels each; these are fairly small neural networks, and much of their representative power comes from the information learned in the corresponding feature textures.

To evaluate $S(\mathbf{u}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}, k)$, we first evaluate the opacity prediction module (shown in blue in Fig. 4):

$$\alpha = \textbf{Alpha}(\mathbf{u}, \boldsymbol{\omega_o}, k). \tag{5}$$

If α is 0, the ray misses the material and continues to the next surface. If α is greater than 0, we must evaluate the next two modules, **Offset** and **Color**, to compute reflectance.

We optionally use a variation of the **Alpha** network that predicts an intermediate value termed *silhouette cosine* instead of the final opacity value. For heightfields and many other materials, the opacity as a function of incoming ray cosine (i.e. the dot product of ray direction and bounding surface normal) is a step function: opaque, then transparent. We simply predict the cosine value where the transition occurs; then α is zero if the camera ray cosine is lower than the predicted value and 1 if equal or greater. We find that this modification is easier to learn; see also Fig. 9).

The neural offset module **Offset** is similar to one from [Kuznetsov et al. 2021]. The one difference is that in our version, the new UV position does not have to be on a straight line given by the ray direction. Therefore, instead of predicting a single value and then calculating the UV offset, we predict the offset directly as a 2D vector (green network in Fig. 4):

$$\mathbf{u}_{\text{new}} = \mathbf{u} + \textbf{Offset}(\mathbf{u}, \boldsymbol{\omega_o}) \tag{6}$$

Finally, we use the new UV location $\mathbf{u}_{\text{new}}$ predicted by **Offset** to query the final feature texture and evaluate the last module, **Color**, which predicts the reflectance RGB value. Our full SBTF evaluation can thus be written as (orange network in Fig. 4):

$$S(\mathbf{u}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}, k) = (\textbf{Color}(\mathbf{u}_{\text{new}}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}, k), \textbf{Alpha}(\mathbf{u}, \boldsymbol{\omega_o}, k)), \tag{7}$$

where $\mathbf{u}_{\text{new}}$ is given by eq. (6) and k by eq. (4).

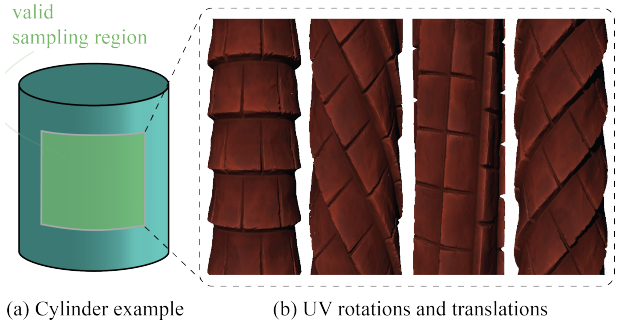


Figure 5: Example of a cylinder in training set. We apply random translation and rotation to the UV mapping. We sample from a smaller region to avoid intersecting near the boundary.

Figure 6: Importance of curvature correction when scaling an object. Top: NeuMIP, which does not support curvature. Middle: our method without curvature correction; the grooves are too shallow. Bottom: our method. The groove sizes match the scale of the object correctly.

4.4 Dataset and training

Given the importance of curvature for the SBTF evaluation, we need to train the neural model on queries of the material applied to surfaces of different curvature. One option would be to use a

database of general shapes; however, this has some disadvantages, such as various discontinuities and distortions in the UV mapping, which are hard to avoid on general shapes. Instead, because we only depend on the scalar curvature values along the viewing ray, we show that it is sufficient to train on cylindrical patches, which have zero curvature along the cylinder axis and trivially controllable curvature in the perpendicular direction.

We generate a dataset of cylindrical patches of different radius, and apply the material microstructure to them. We apply different rotations and translations to the UV mapping of the patch; see Fig. 5. The generated material is completely below the cylindrical surface, which acts as a bounding geometry. Our synthetic materials come from Adobe Substance Source [Adobe 2021], though nothing in our method is specific to Substance.

We sample viewing rays incident upon the patch from different directions perpendicular to the cylinder axis. The curvature along the rays is simply the reciprocal of the cylinder radius. Since we have applied random UV rotations and translations, this covers all combinations of texture coordinate and camera direction in the local coordinate frame for the whole neural material.

For the light direction, we randomly sample a ray in a sphere, not just the hemisphere above the horizon. This is because the light can come from below the horizon due to the curvature of the patch. For a single material, our training dataset consists of 2400 different cylindrical surface patches with 65,536 queries for each.

For the silhouette cosine version of the **Alpha** network, we also need to calculate the ground truth silhouette cosine for a given viewing ray. To do that, we use binary search to find the angle at which the incoming ray exactly grazes the material boundary.

To regularize the training process, we apply a Gaussian blur to all neural textures in the initial stages; as the training continues, we continually decrease the standard deviation of the blur, similar to [Kuznetsov et al. 2021]. Instead of using just the L1 or L2 or loss, we combine them, with L1 having one tenth of the weight of L2. We also apply an L1 loss on the neural offset vector predicted by the **Offset** network (comparing the predicted and ground truth offset vectors in texture space).

We used ground truth UV displacement and cosine term information in most of our results. However, it is possible to train our models without this information. Figure 2 in the supplemental materials demonstrates that our method can learn the appearance in a semi-unsupervised manner, only relying on color and alpha losses. We also applied our method to real-world data (see Figure 11). Since it is hard to obtain a dataset of real materials applied to cylinders (or other objects of known curvature), we instead extract a heightfield from the real data; using that, we can re-render the real-world materials applied to different cylinders.

4.5 Rendering

Our architecture can be used to evaluate an SBTF query for any single shading point; we do not batch queries at render time. Therefore, it can be easily integrated with Monte Carlo renderers. We currently use a C++ implementation of network inference on the CPU in the Mitsuba renderer [Jakob 2010], implemented as a standard BSDF material plugin. The same approach could be used to extend a GPU-based renderer using CUDA.

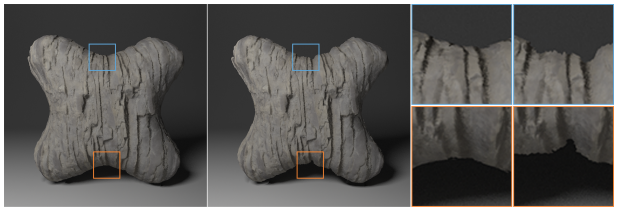


Figure 7: Effects of different curvature calculation methods. (a,c) Using the standard mean curvature as input would result in inflated silhouette boundaries. (b,d) By using curvature along the tangent direction of camera rays, we yield more faithful silhouettes of neural materials.

To ensure a perfect match between PyTorch model definitions and the C++ implementation, we developed special code, which traces all the operations done by PyTorch and then generates a C++ function and a model file automatically. This completely eliminates errors of porting models and enables fast iterations over the architecture of the model, with immediate results in the renderer.

The design of our three neural modules is amenable to efficient rendering. We can terminate early if $\alpha = 0$, i.e. only the blue network in Fig. 4 will be evaluated. Also, because the first two modules do not depend on the light direction, their evaluation can be reused for multiple light direction queries, like in the common case of multiple importance sampling (MIS).

For SBTF sampling, we currently use a simple Lambertian (cosine) pdf, like [Kuznetsov et al. 2021]. However, other importance sampling approaches have been proposed for neural materials, and these could be used with our method, such as the Gaussian proxy approach used by [Fan et al. 2021].

We assume the texture coordinate units match world units, i.e. a distance of 1 unit in texture space is approximately 1 world unit. Our materials are precomputed at a specific scale; sometimes we want the material to be applicable to geometries of a different scale. In addition to scaling texture coordinates, we need to correspondingly scale the curvature values; recall that the reciprocal curvature is the radius of the osculating circle, and therefore has world units. See Fig. 6 for an illustration of the importance of this correction.

Given we have a more complex architecture, our network evaluation times are about 30% higher than for NeuMIP (non-multi-resolution version), although we believe significant further optimizations are possible. We believe this is a very reasonable trade-off for capturing high-quality silhouette detail.

5 RESULTS

In this section, we showcase the ability of our neural method to render intricate geometric details on silhouettes that can otherwise be obtained only from a highly tessellated and displaced mesh.

Teaser. In Fig. 1, we render a scene consisting of five bunnies with different neural materials under area light illumination with our method. The bunny geometry presents highly curved surfaces which breaks NeuMIP on silhouette regions noticeably, leading to visibly smooth, unrealistic boundaries. On the other hand, our method is capable of learning the microstructure details of neural

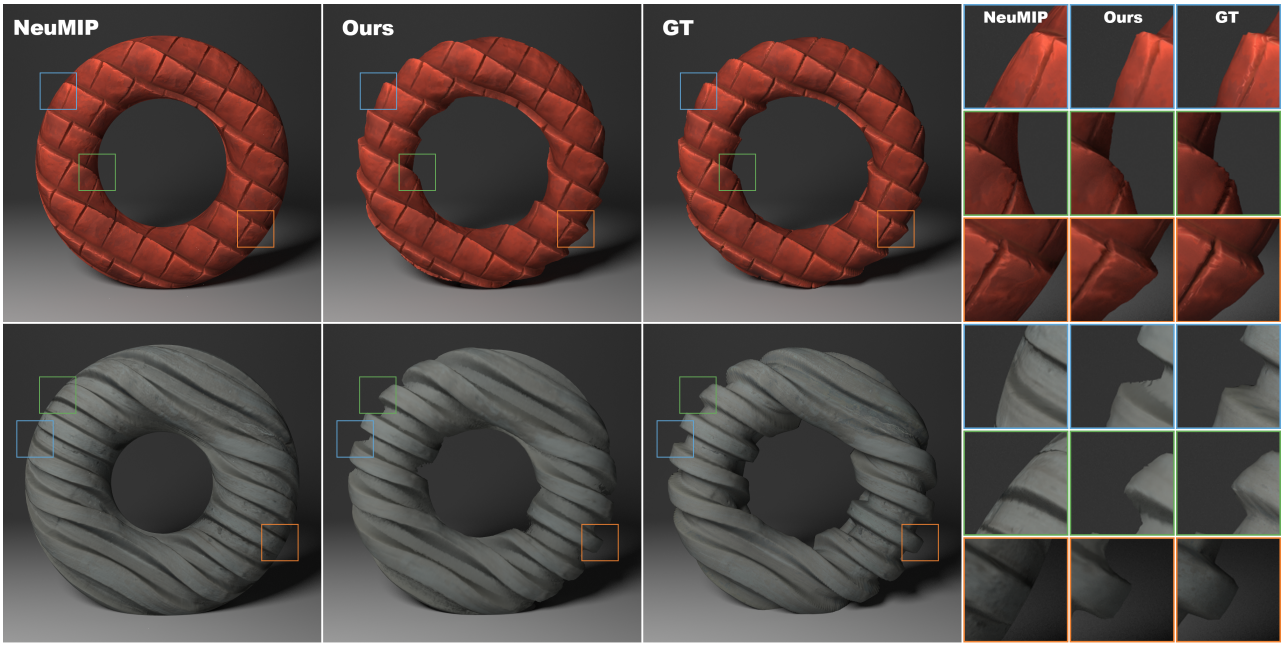


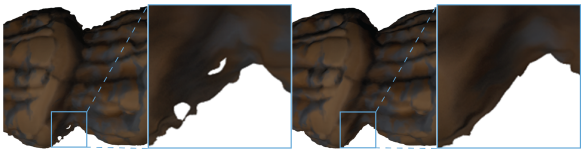
Figure 8: Qualitative results. We show a comparison of our method with NeuMIP and ground truth (GT) for four materials. The neural materials are applied to a simple torus mesh, while the ground truth is computed by fine tessellation and displacement. Our method is able to render silhouettes that retain fine detail from geometric displacement, while NeuMIP fails to produce any details resulting in smooth edges and a flatter look. See additional results in the supplemental materials.

materials during training, and produces renderings that match the ground truth more faithfully.

Curvature calculation. Curvature is not a single number per shading point, but is defined as two principal curvatures with corresponding principal directions. Instead of feeding this information to the network, we would like a single number to describe curvature. A standard way is to use mean curvature, but as Fig. 7 shows, it results in inaccurate appearance. This is because it collapses the local surface shape to a single number. Instead, in our method we calculate the curvature along the camera ray as a single number, which captures the curvature parameter better.

Silhouette cosine vs. direct alpha prediction. Instead of predicting the alpha value directly, we predict the *silhouette cosine* and then compare to cosine of the camera angle. As seen in Fig. 9, without this technique, the opacity is harder to learn, and the silhouette regions may contain artifacts such as holes and jagged edges. However, some materials (like with transparency) require direct alpha prediction as silhouette cosine is binary (see Figure 10).

Qualitative results. We compare our approach against the NeuMIP baseline and ground truth in Fig. 8. A set of neural materials are applied to a simple and smooth torus mesh, while the ground truth material is applied on a torus mesh that is subdivided and displaced. Unlike NeuMIP that produces smooth silhouettes without any details from displacements at grazing angles, our method successfully renders fine-level geometric displacements that more closely match the ground truth. In the video, we also show that our method is temporally coherent for changes of lighting and viewpoint. For additional materials and geometries, please refer to the supplemental document.



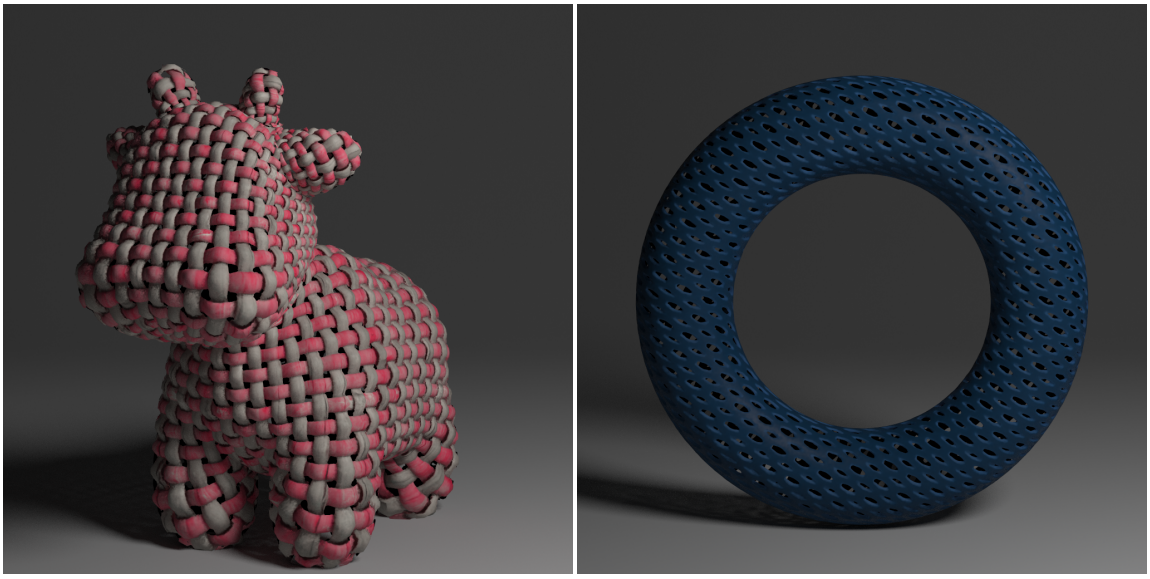
(a) Direct alpha prediction (b) Our method

Figure 9: Silhouette cosine vs. direct alpha prediction. Instead of predicting alpha value directly, we predict the *silhouette cosine* and then compare to the cosine of the camera angle. Without this technique, the silhouette regions may contain artifacts such as holes and jagged edges.

Table 1: Intersection over Union (IoU) of silhouettes.

	Cave	Foam	Leather	Oyster	Shell	Stone 1	Stone 2	Terracotta	Wall	Wool 1	Wool 2
NeuMIP	0.756	0.772	0.722	0.391	0.502	0.642	0.635	0.553	0.750	0.798	0.602
Ours	0.940	0.953	0.939	0.803	0.843	0.897	0.916	0.873	0.934	0.918	0.888

Quantitative silhouette error. In order to evaluate and compare our method with NeuMIP, we compute the silhouette errors in Table 1. The error is defined as the intersection over union (IoU) of the ground truth region with non-zero opacity and a given method. Here, value of 1 would mean the two silhouettes are exactly the same, and the value of 0 would mean that two silhouettes do not overlap at all. Our IoU values are substantially better than NeuMIP.



(a) Basket (b) Perforated Leather

Figure 10: Our model supports non-heightfield and see-through materials. The basket weave material on the left is made of red wicker, which runs horizontally, and grey wicker, which runs vertically. As such, there is no heightfield, because the threads superimpose on top of each other. Nevertheless, our method is able to represent the material accurately. The perforated leather on the right also breaks the heightfield assumption. Both basket and perforated leather materials are see-through, which is easily supported by our method; for these materials, we predict alpha directly, instead of using silhouette cosine.

Figure 11: Our method can be applied to real (measured) BTFs. Here we used Carpet 7 from the UBO [2014] dataset.

Limitations. There can be a slight blurriness compared to ground truth, as is common for any neural technique. Note that our method must learn a higher-dimensional space to match across all curvature k values, while considering 2D offsets. For thin objects, we see some "inflation" of the silhouette from the bounding geometry, though our silhouettes still match more closely than previous methods. Similar to many material models, our method requires a good UV mapping, free of major distortion.

6 CONCLUSION AND FUTURE WORK

We have presented a neural architecture capable of correctly handling grazing and silhouette effects for materials. To achieve this, we add an explicit concept of surface curvature and transparency to the neural model. Our solution uses the new concept of a Silhouette BTF (SBTF), which explicitly considers curvature as part of the material query, and returns a transparency value as part of the query output, in addition to reflectance. This allows query rays with grazing directions in a curved region to hit or miss the material microstructure, leading to a complex silhouette boundary that is very important for the illusion of material complexity. Building upon the NeuMIP architecture, we demonstrated complex materials including accurate silhouette effects that were not possible with any previous neural or BTF-based materials, and would traditionally require expensive techniques such as displacement mapping, shell mapping or volumetric ray marching.

Several exciting future work avenues could be explored. One of them would be the capture of real world objects, estimating a combination of a coarse mesh and an SBTF. We could also explore effects such as light transmission, hair/fur, or subsurface scattering in the SBTF.

ACKNOWLEDGMENTS

This work was funded in part by NSF grants 1703957 and 1730158, ONR DURIP N000141912293, the Ronald L. Graham Chair, Adobe and the UC San Diego Center for Visual Computing. We would like to thank The Stanford 3D Scanning Repository, Keenan Crane and Blender, for the geometries.

REFERENCES

2022. Principal Curvature. https://en.wikipedia.org/wiki/Principal_curvature.

Adobe. 2021. Substance Source library. https://source.substance3d.com.

Hendrik Baatz, Jonathan Granskog, Marios Papas, Fabrice Rousselle, and Jan Novák. 2021. NeRF-Tex: Neural Reflectance Field Textures. In *Eurographics Symposium on Rendering - DL-only Track*, Adrien Bousseau and Morgan McGuire (Eds.). The Eurographics Association.

Kristin J Dana, Bram Van Ginneken, Shree K Nayar, and Jan J Koenderink. 1999. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics (TOG)* 18, 1 (1999), 1–34.

Jean-François Dufort, Luc Leblanc, and Pierre Poulin. 2005. Interactive rendering of meso-structure surface details using semi-transparent 3d textures. In *Proc. of Vision, Modeling and Visualization*. 399–406.

Jiahui Fan, Beibei Wang, Miloš Hašan, Jian Yang, and Ling-Qi Yan. 2021. Neural BRDFs: Representation and Operations. arXiv:2111.03797 [cs.GR]

Jiří Filip and Michal Haindl. 2008. Bidirectional texture function modeling: A state of the art survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31, 11 (2008), 1921–1940.

Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. https://libigl.github.io/.

Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.

Tomomichi Kaneko, Toshiyuki Takahei, Masahiko Inami, Naoki Kawakami, Yasuyuki Yanagida, Taro Maeda, and Susumu Tachi. 2001. Detailed shape representation with parallax mapping. *In Proceedings of the ICAT* 2001 (01 2001).

Alexandr Kuznetsov, Milos Hasan, Zexiang Xu, Ling-Qi Yan, Bruce Walter, Nima Khademi Kalantari, Steve Marschner, and Ravi Ramamoorthi. 2019. Learning generative models for rendering specular microgeometry. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 225.

Alexandr Kuznetsov, Krishna Mullia, Zexiang Xu, Miloš Hašan, and Ravi Ramamoorthi. 2021. NeuMIP: multi-resolution neural materials. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–13.

Manuel M Oliveira, Gary Bishop, and David McAllister. 2000. Relief texture mapping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 359–368.

Manuel M Oliveira and Fabio Policarpo. 2005. An efficient representation for surface details. *UFRGS Technical Report* (2005).

Fábio Policarpo, Manuel M Oliveira, and Joao LD Comba. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*. 155–162.

Serban D. Porumbescu, Brian Budge, Louis Feng, and Kenneth I. Joy. 2005. Shell Maps. *ACM Trans. Graph.* 24, 3 (jul 2005), 626–633.

Gilles Rainer, Abhijeet Ghosh, Wenzel Jakob, and Tim Weyrich. 2020. Unified Neural Encoding of BTFs. *Computer Graphics Forum (Proceedings of Eurographics)* 39, 2 (2020), 167–178.

Gilles Rainer, Wenzel Jakob, Abhijeet Ghosh, and Tim Weyrich. 2019. Neural BTF Compression and Interpolation. *Computer Graphics Forum* (2019).

László Szirmay-Kalos and Tamás Umenhoffer. 2008. Displacement Mapping on the GPU — State of the Art. *Computer Graphics Forum* 27 (2008).

Theo Thonat, Francois Beaune, Xin Sun, Nathan Carr, and Tamy Boubekeur. 2021. Tessellation-Free Displacement Mapping for Ray Tracing. *ACM Trans. Graph.* 40, 6, Article 282 (dec 2021), 16 pages.

Jiaping Wang, Xin Tong, John Snyder, Yanyun Chen, Baining Guo, and Heung-Yeung Shum. 2005a. Capturing and rendering geometry details for BTF-mapped surfaces. *The Visual Computer* 21, 8 (2005), 559–568.

Jiaping Wang, Xin Tong, John Snyder, Yanyun Chen, Baining Guo, and Heung-Yeung Shum. 2005b. Capturing and rendering geometry details for BTF-mapped surfaces. *The Visual Computer* 21, 8 (2005), 559–568.

Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. 2003. View-dependent displacement mapping. *ACM Transactions on graphics (TOG)* 22, 3 (2003), 334–339.

Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. 2004. Generalized displacement maps. In *Eurographics Symposium on Rendering (EGSR)*. 227–233.

Michael Weinmann, Juergen Gall, and Reinhard Klein. 2014. Material Classification Based on Training Data Synthesized Using a BTF Database. In *ECCV 2014*. 156–171.

Shuang Zhao, Wenzel Jakob, Steve Marschner, and Kavita Bala. 2014. Building Volumetric Appearance Models of Fabric Using Micro CT Imaging. *Commun. ACM* 57, 11 (oct 2014), 98–105.

Supplemental Materials

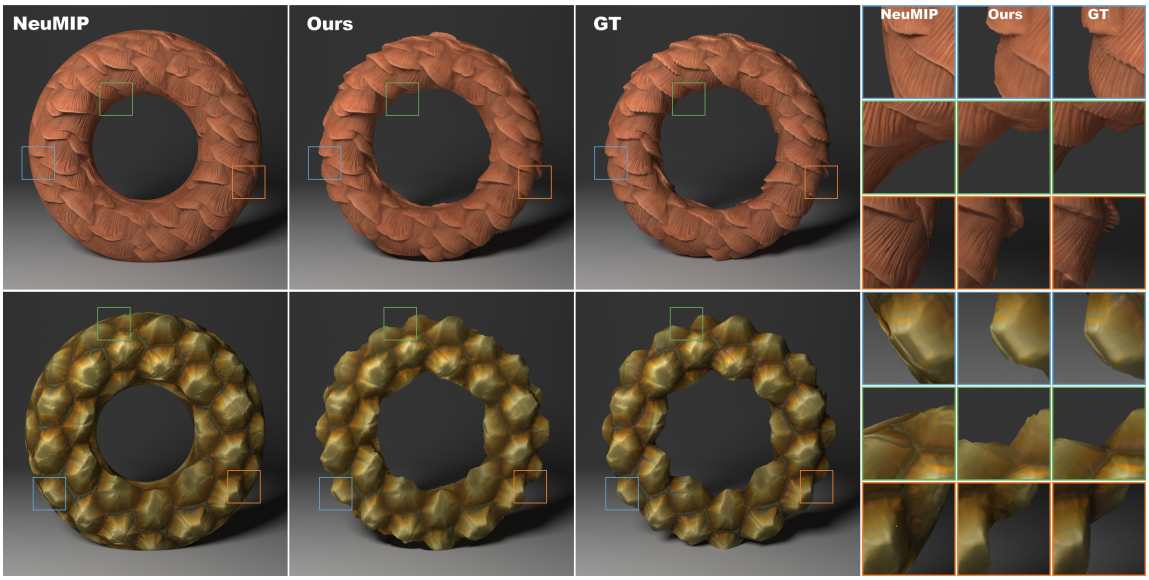


Fig. 1. **Additional qualitative results**. We show a comparison of our method with NeuMIP and ground truth (GT) for four materials. The neural materials are applied to a simple torus mesh, while the ground truth is computed by fine tessellation and displacement. Our method is able to render silhouettes that retain fine detail from geometric displacement (as shown in the ground truth), while NeuMIP fails to produce any details resulting in smooth edges and a flatter look.

Fig. 2. Our method can work without UV loss. Here, we learned terracotta material in a semi-supervised way, relying only on color and alpha for the loss. We used direct alpha predicting architecture as the ground truth cosine term might not be available.

Fig. 3. Our method works well on different geometries. Here are examples of our materials on a cow, a monkey, and an armadillo.

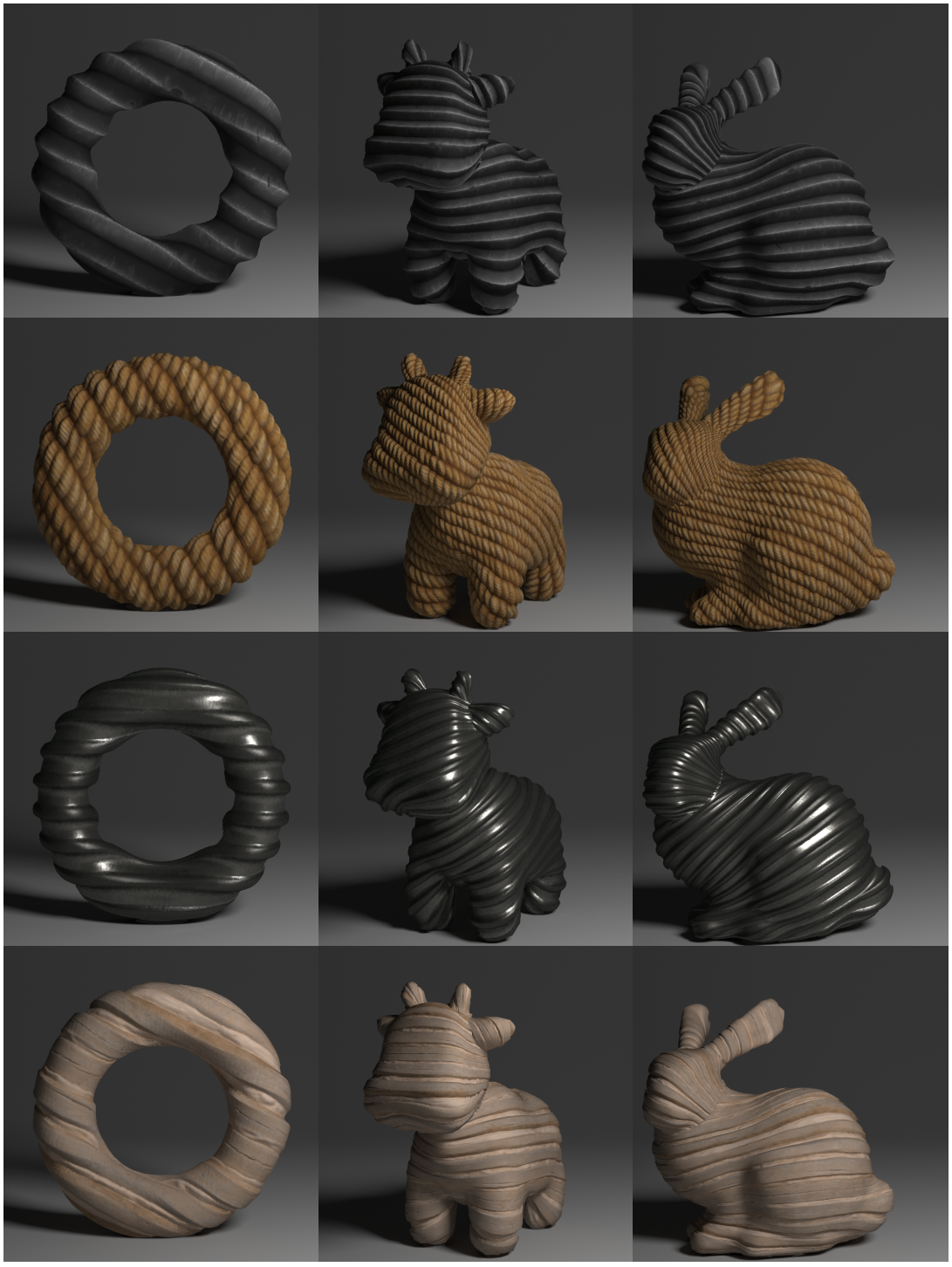


Fig. 4. Our method supports a wide variety of materials.