Scratch-based Reflection Art via Differentiable Rendering

PENGFEI SHEN*, Universty of Science and Technology of China, China
RUIZENG LI*, Universty of Science and Technology of China, China
BEIBEI WANG†, Nankai University and Nanjing University of Science and Technology, China
LIGANG LIU†, University of Science and Technology of China, China

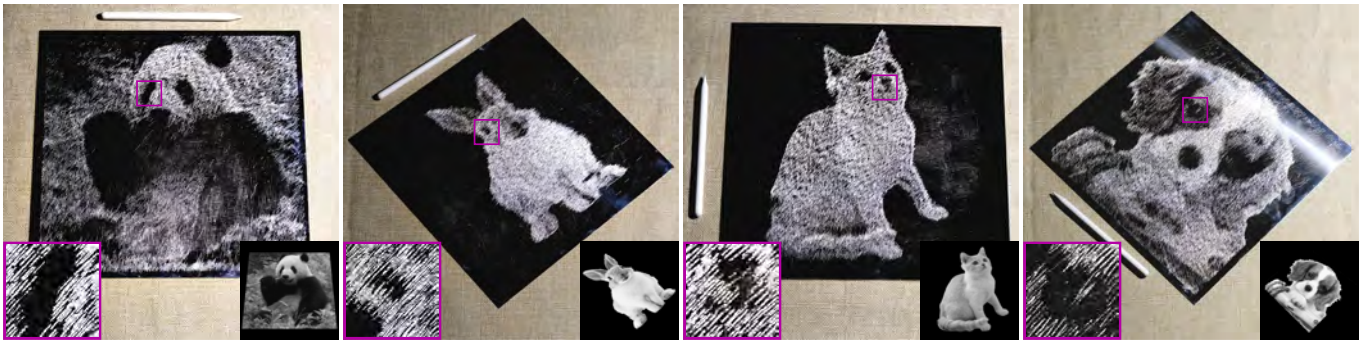


Fig. 1. We present a new type of 3D visual reflection art - scratch-based reflection art. Given four target images (bottom right), our method fabricates a set of scratches (closeup patches in the bottom left) on a single metallic board to display these images when viewed from four different directions. The photos (middle) are taken from four views respectively. To achieve that, we design analytical scratch geometric and shading models to enable differentiable rendering, which allows computationally efficient optimization eventually. An apple pencil is put near the board as a size reference.

The 3D visual optical arts create fascinating special effects by carefully designing interactions between objects and light sources. One of the essential types is 3D reflection art, which aims to create reflectors that can display different images when viewed from different directions. Existing works produce impressive visual effects. Unfortunately, previous works discretize the reflector surface with regular grids/facets, leading to a large parameter space and a high optimization time cost. In this paper, we introduce a new type of 3D reflection art - *scratch-based reflection art*, which allows for a more compact parameter space, easier fabrication, and computationally efficient optimization. To design a 3D reflection art with scratches, we formulate it as a multi-view optimization problem and introduce differentiable rendering to enable efficient gradient-based optimizers. For that, we propose an analytical scratch rendering approach, together with a high-performance rendering pipeline, allowing efficient differentiable rendering. As a consequence, we could display multiple images on a single metallic board with only several minutes for optimization. We demonstrate our work by showing virtual objects and manufacturing our designed reflectors with a carving machine.

*Joint first authors.
†Corresponding authors.

Authors' addresses: Pengfei Shen, Universty of Science and Technology of China, China, jerry_shen@mail.ustc.edu.cn; Ruizeng Li, Universty of Science and Technology of China, China, pb17061297@mail.ustc.edu.cn; Beibei Wang, Nankai University and Nanjing University of Science and Technology, China, beibei.wang@njust.edu.cn; Ligang Liu, University of Science and Technology of China, China, lgliu@ustc.edu.cn.

0730-0301/2023/8-ART $15.00
https://doi.org/10.1145/3592142

CCS Concepts: • **Computing methodologies** → **Rendering**; **Reflectance modeling**.

Additional Key Words and Phrases: Scratch Rendering, differentiable rendering, 3D reflection art

ACM Reference Format:
Pengfei Shen, Ruizeng Li, Beibei Wang, and Ligang Liu. 2023. Scratch-based Reflection Art via Differentiable Rendering. *ACM Trans. Graph.* 42, 4 (August 2023), 12 pages. https://doi.org/10.1145/3592142

1 INTRODUCTION

The 3D visual optical arts can produce impressive special effects by designing interactions between objects and light sources, displaying images in shadows, caustics, or reflectors. The design of such effects is challenging due to the complex interactions between the lights and objects. Therefore, computational 3D visual arts [Glasner et al. 2014; Levin et al. 2013; Mitra and Pauly 2009; Sakurai et al. 2018; Weyrich et al. 2009; Wu et al. 2022b; Yue et al. 2014] have attracted a great deal of attention to creating these special effects automatically.

Among various kinds of computational 3D visual arts, we focus on the 3D reflection arts [Wu et al. 2022b]. This type of art aims to create reflectors that can display different images when viewed from different directions. The existing works achieve this goal by modifying different properties on the surface, as shown in Figure 2. Some works manipulate the height field by placing small mirrors [Weyrich et al. 2009] or raised stripes [Sakurai et al. 2018; Snelgrove et al. 2013]. The others control the reflectance [Lan et al. 2013; Matusik et al. 2009], or colors [Pjanic and Hersch 2015a,b]. However, these works discretize reflector surfaces with regular grids/facets, leading to a huge parameter space, which raises difficulties for optimization and fabrication. Moreover, previous works use the genetic algorithm

(GA) or complex strategies (e.g., alternative optimizing) for optimization, yielding a high computational cost or inflexible optimization.

In this paper, we introduce a novel way to create 3D reflection art using scratches, called *scratch-based reflection art* to overcome the above issues. Compared to uniform grids/facets, much fewer scratches are required to cover the entire reflector surface, making the parameter space more compact. Meanwhile, scratches are much easier and more natural to manufacture with a simple carving machine. No existing works have utilized scratches for 3D reflection art to our best knowledge. Furthermore, we introduce differentiable rendering into our scratch optimization problem, allowing efficient gradient-based optimization algorithms. Since existing scratch rendering methods [Velinov et al. 2018; Werner et al. 2017] are infeasible for this task due to their discontinuity and complexity, we propose a novel analytical scratch rendering approach, as well as a high-performance rendering pipeline, which serves as a core in our scratch optimization. Consequently, we can display multiple images on a single metallic board with only several minutes for optimization. We demonstrate our method by showing the virtual objects and the manufactured real ones.

The major contributions of this work are as follows.

- We propose a novel way to create 3D reflection art with scratches, allowing more compact parameter space and easier fabrication.
- We propose an analytical scratch rendering approach and high-performance rendering pipeline, which are capable of differentiable rendering.
- We introduce differentiable rendering of scratches into 3D reflection art design, leading to flexible and computation-efficient optimization.

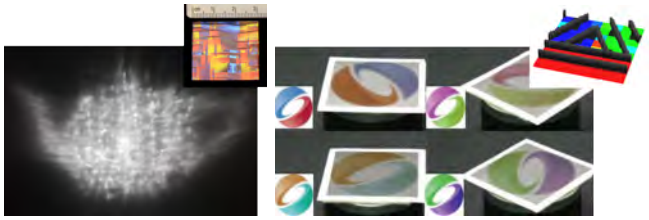
2 RELATED WORK

We will briefly review previous works related to microstructure and scratch appearance modeling, as well as 3D reflection art design.

Microstructure appearance modeling. The materials in the real world usually show imperfections with microstructure appearances, like car paints, metallic flakes, or brushed metal. To model such appearances, existing approaches [Yan et al. 2014] use high-resolution normal maps to model every fine detail explicitly. Yan et al. [2014] introduce patch-local normal distribution functions ($\mathcal{P}$-NDFs) to compute the spatially and directionally varying appearance accurately. Their work is later improved by Yan et al. [2016] with Gaussian elements to represent the position-normal distribution and by Yan et al. [2018] to handle wave optics effects. More efforts have been made to alleviate the storage cost with texture synthesis [Wang et al. 2020] or Generative Adversarial Networks (GAN) to generate NDF images [Kuznetsov et al. 2019]. The other works focus on improving the rendering performance with prefiltering [Atanasov et al. 2021; Deng et al. 2022; Gamboa et al. 2018; Tan et al. 2022].

The above approaches demand high-resolution normal maps to define the microstructures, which are challenging for optimization due to the high degree of freedom.

Scratch appearance rendering. Raymond et al. [2016] modeled the scratch appearance using a collection of 1D scratches. Their model relied on pre-integration, without an individual scratch representation. Later, Werner et al. [2017] managed to render iridescent microscale scratches, by combining with wave optics theory. Their model can show a realistic scratch appearance overall. However, this model is slow and impractical for optimization. Later, Velinov et al. [2018] improved their performance by introducing an analytical formulation for a surface patch, as well as an efficient GPU implementation, achieving a real-time frame rate, at the cost of low quality. However, their method did not take into account the precise boundary of scratches, which is incapable of differentiable rendering. Moreover, both models assume triangular or rectangular-shaped scratches, which are difficult to fabricate.



(a) Weyrich et al. [2009] (b) Sakurai et al. [2018]

Fig. 2. Previous works display images on a reflector by placing mirrors (left) or raised strips (right).

3D Reflection Art Design. Different kinds of visual reflection arts have been investigated, like anamorphosis [de Comite and Grisoni 2015], mirror cup and saucer art [Wu et al. 2022a], etc. Papas et al. [2011] manufactured surfaces which produce desired caustic images using a small collection of patches. Elek et al. [2017] reproducted color texture in 3D printing. A detailed survey on 3D visual reflection arts can be found in the literature [Wu et al. 2022b]. We only briefly discuss the most related ones, aiming at displaying images on reflectors.

One way to create desired appearances is to fabricate microstructures. Weyrich et al. [2009] manufactured a mirror height field to exhibit desired appearance with a milling machine. Snelgrove et al. [2013] manufactured the surface with parallax barriers and printed inks to display a single image. Recently, Sakurai et al. [2018] fabricated microstructure reflectors with raised stripes in subdivided cells to show different images under different views. Their method uses GA for optimization and has a high computational cost due to the large parameter space. Levin et al. [2013] used photolithography to fabricate microstructure on wafers by considering wave optics, producing different high-resolution images. Another work by Glasner et al. [2014] also considered wave optics and fabricated reflectors with a spatial light modulator. Both methods require extremely costly devices, making them less practical.

The other works [Pjanic and Hersch 2015a,b] modify colors on reflectors using inks rather than microstructures, which can only display at most two images.

Matusik et al. [2009] fabricated isotropic spatially-varying reflectance on a flat surface with a combination of inks. The color at a given position is immutable, because of the isotropic bidirectional reflectance distribution function (BRDF). Later, Lan et al. [2013]

propose to print surface appearance with anisotropic reflectance and normal variations. Different colors can be shown at the same position from different views. The combination of normal variations and BRDF reflectance leads to a vast parameter space. Thus, they design an alternative optimizing strategy to decrease the time cost.

3 METHOD

3.1 Problem and formulation

Problem statement. Our work aims to carve N scratches on a metallic board, so different desired appearances can be displayed under M view/light configurations. Each of these appearances follows a specific image. The reason for these characterized appearances is that the scratches change the light paths, as shown in Figure 3. We manually specify the view/light positions.

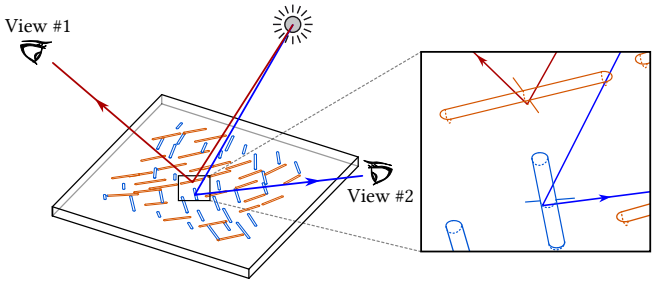


Fig. 3. Scratches change the light paths so that scratches with specific orientations can be seen from one view but hide from the other. For example, the orange scratches can be seen by the left camera, while the blue scratches are visible by the right camera.

Formulation. Assuming that the scratches are straight lines, each scratch's location can be represented with its two endpoints (l_0, l_1). Regarding the other parameters (e.g., geometric and optical properties), we will show them in the coming sections. The notations are listed in Table 1.

The key question is how to design these scratches to generate appearances that match the target images. The appearance of the reflector can be simulated by rendering the scratches under a given view/light configuration. Therefore, our problem can be formulated as an optimization problem, predicting scratches parameters by minimizing the difference between the multi-view rendered images I and the target images T:

$$\min_{\mathcal{S}} E_{\text{loss}}\left(\{I_j(\mathcal{S}; V_j, L_j), T_j\}_j\right), \tag{1}$$

where T_j is j^{th} target image, I_j is the rendered result of scratches with parameters $\mathcal{S}$ under view V_j and light L_j. E_{loss} is the loss function to measure similarities between rendered and target images.

Challenges. We hope to optimize the scratch parameters $\mathcal{S}$ to achieve our goal in Eqn. (1). Previous work [Sakurai et al. 2018] used a genetic algorithm to solve their optimization problem, leading to an enormous computation overhead (e.g., three hours). To overcome this issue, we resort to gradient-based optimization, which requires the gradients of all parameters. To this end, we introduce differentiable rendering into this optimization problem. To render scratches, we need to define the scratch shape as well as the interactions between the scratches and lights (BRDF). Both models must be differentiable and efficient to enable computationally efficient optimization. Since no existing scratch rendering approaches meet these requirements, we propose novel scratch geometric and shading models in our paper.

3.2 Scratch geometric model

Existing approaches [Velinov et al. 2018; Werner et al. 2017] define a scratch with a triangular or rectangular cross section, which is simple but raises several difficulties. First, both shapes are defined by piece-wise functions, which need more effort to enable differentiable rendering. Second, carving such shapes is unnatural for a carving machine. Third, the piece-wise flat shape imposes strong restrictions on the view/light elevation angle to form a valid path. For these reasons, we propose a smooth function to express a scratch's shape, which allows for differentiable rendering and easy fabrication.

We define scratches in the UV space, which has lower dimensionality than the object space. For each scratch, we model its shape with a height function parameterized in the local coordinate system. The direction along a scratch is set as the y-axis, and the normal of the surface is set as the z-axis, as shown in Figure 4. In the local frame, the height and normal along the y-axis remain constant and vary only along the x-axis. In this way, we only define the height for the cross-section of a scratch along the x-axis (see Figure 4b):

$$h(x; \widetilde{w}, d) = -\frac{1}{2}\widetilde{w}\log\left(1 - \frac{4x^2}{\widetilde{w}^2}\right) - d. \tag{2}$$

The height function is defined within $(-0.5\widetilde{w}, 0.5\widetilde{w})$ and goes to positive infinity where $|x| = 0.5\widetilde{w}$, so we treat $\widetilde{w}$ as the scratch width. d denotes the scratch depth. Parameters $\widetilde{w}$ and d establish a scratch's shape. Our height function can represent a wide range of shapes by setting $\widetilde{w}$ and d, from flat to steep, as shown in Figure 5. Thus, our height function has a high representation ability.

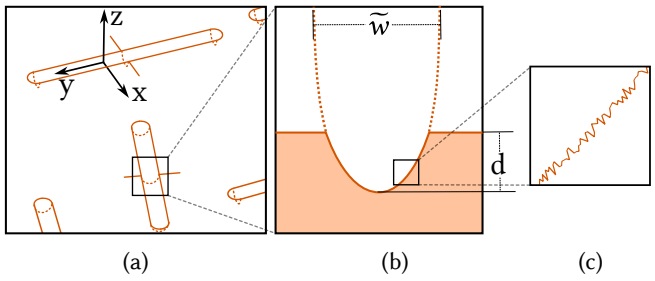


Fig. 4. (a) Each scratch is parameterized in its local coordinate frame. (b) The main parameters to define a scratch's shape include the depth d and width $\widetilde{w}$. (c) Our reflection model is a two-scale model: the macroscale is defined by the scratch height function and the microscale is defined by the intrinsic roughness.

3.3 Scratch shading model

Now, let us define our scratch shading model or BRDF. Like other microstructures, each pixel might cover many scratches, as shown in Figure 6. Therefore, we need to define the scratch BRDF on a

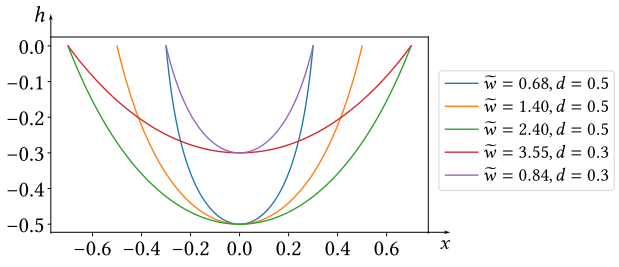


Fig. 5. Our model can represent a wide range of shapes with varying parameters.

Table 1. Notations. The referenced equations are shown in the last column.

Symbol	Definition	Eqn.
l_0, l_1	two endpoints of a scratch	-
$h(x;\widetilde{w},d)$	height function of a scratch	(2)
ω_i, ω_o	incoming/outgoing direction	(3)
$\mathbf{h}$	half vector	(3)
α	intrinsic roughness	(6)
f_{scratch}	scratch BRDF	(3)
f_{surface}	surface BRDF	(3)
$D_x(\mathbf{h}, n)$	NDF of a point on a scratch	(6)
$D_i(\mathbf{h})$	$\mathcal{P}$-NDF by scratch i	(8)
$\hat{D}_i(\mathbf{h})$	NDF integrated on scratch i's boundary	(8)
$D_{\mathcal{P}}(\mathbf{h})$	$\mathcal{P}$−NDF	(3)
$S_{\mathcal{P}}$	pixel's footprint size	(8)
$\mathcal{P}_i$	pixel's footprint covered by scratch i	(8)
$\mathbf{n}(x;\widetilde{w})$	shading normal at point x	(5)

surface patch seen from a pixel, or called a pixel's footprint, rather than on a single point.

Similar to microstructure models [Yan et al. 2016], we mainly focus on the normal distribution function (NDF) and leave the other terms the same as the microfacet model. Therefore, our scratch BRDF is defined as follows:

$$f_{\text{scratch}}(\omega_i, \omega_o) = \frac{D_{\mathcal{P}}(\mathbf{h}) G(\omega_i, \omega_o) F(\omega_o, \mathbf{h})}{4|\omega_i \cdot \mathbf{n}||\omega_o \cdot \mathbf{n}|}, \tag{3}$$

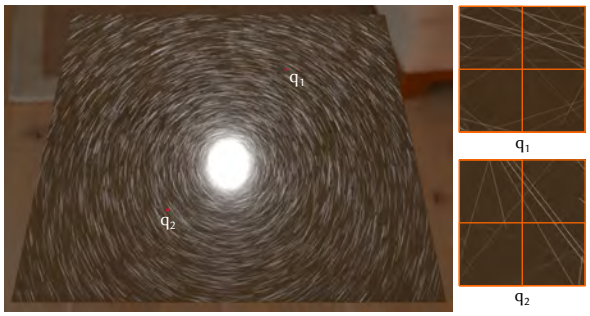


Fig. 6. Each pixel of a rendered image (left) actually includes a large number of scratches. Here, we visualize the detailed scratch distribution within four pixels around two places.

where ω_i and ω_o are the incoming/outgoing directions, respectively. $\mathbf{n}$ is the surface geometric normal, $\mathbf{h}$ is the half vector between the incoming and outgoing directions, G is the shadowing-masking function, and F is the Fresnel term. $D_{\mathcal{P}}$ denotes the patch NDF ($\mathcal{P}$-NDF) defined on the pixel's footprint, which aggregates NDFs of all scratches within the pixel's footprint:

$$D_{\mathcal{P}}(\mathbf{h}) = \sum_{i \in \mathcal{P}} D_i(\mathbf{h}), \tag{4}$$

where D_i is the NDF of scratch i locating in the patch $\mathcal{P}$. Before deriving the NDF for a scratch, we first formulate NDF of a point (D_x) on the scratch.

Point NDF. Each point on the scratch has a normal variation, which depends on the height function. Besides that, since the interior of the scratch surface can not be perfectly specular, we introduce an intrinsic roughness for scratches at the microscale (see Figure 4c). Therefore, the NDF of a point on the scratch is a combination of the macro-scale normal variation and microscale intrinsic roughness.

We first define the scratch normal at location x, from the height function (Eqn. (2)). Note that the normal $\mathbf{n}$ only depends on $\widetilde{w}$.

$$\begin{aligned}\mathbf{n}(x;\widetilde{w}) &= \frac{(-h_x(x;\widetilde{w},d),0,1)}{\|(-h_x(x;\widetilde{w},d),0,1)\|} \\ &= \left(-\frac{4\widetilde{w}x}{\widetilde{w}^2+4x^2}, 0, \frac{\widetilde{w}^2-4x^2}{\widetilde{w}^2+4x^2}\right),\end{aligned} \tag{5}$$

where h_x is the partial derivative of the height distribution function with respect to the x direction.

The microscale NDF is modeled with GGX [Walter et al. 2007]. Combining the two scales leads to our point NDF:

$$D_x(\mathbf{h}, \mathbf{n}(x)) = \frac{\alpha^2 \chi^+(\mathbf{n}(x)\cdot\mathbf{h})}{\pi\left((\mathbf{n}(x)\cdot\mathbf{h})^2(\alpha^2-1)+1\right)^2}, \tag{6}$$

where α is the intrinsic roughness. Substituting the scratch normal function Eqn. (5) into the above equation results in:

$$D_x(\mathbf{h}, \mathbf{n}(x)) = \frac{\alpha^2}{\pi\left(\frac{(\alpha^2-1)(\mathbf{h}_z(\widetilde{w}^2-4x^2)-4\mathbf{h}_x\widetilde{w}x)^2}{(\widetilde{w}^2+4x^2)^2}+1\right)^2}. \tag{7}$$

Scratch NDF. On top of the point NDF, the NDF of scratch i covered by a surface patch is an integral of the point NDF over the patch area:

$$D_i(\mathbf{h}) = \frac{1}{S_{\mathcal{P}}} \int_{\mathcal{P}_i} D_x(\mathbf{h}, \mathbf{n}(x))\, dxdy, \tag{8}$$

where $S_{\mathcal{P}}$ is the footprint size, and the integral domain $\mathcal{P}_i$ is the patch area intersected by scratch i.

Eqn. (8) could be solved by point sampling but requires more effort to be differentiable. Therefore, we derive an analytical model by transforming the integral of the scratch area into the integral on the boundaries, using Green's theorem [Riley et al. 2006] and then solve the boundary integral in an analytical manner.

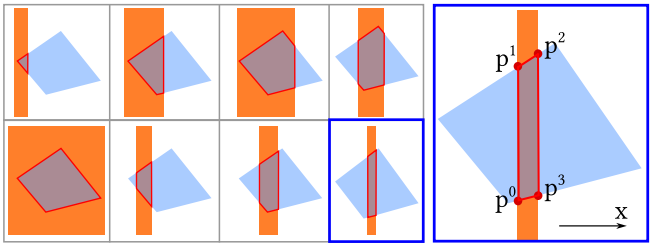


Fig. 7. There are eight types of intersection between a footprint (blue) and a scratch (orange). The integral boundaries are marked red in each type. The figure on the right is the close-up view of the one framed in blue.

Based on Green's theorem [Riley et al. 2006], the above equation can be rewritten as:

$$\begin{aligned} D_i(\mathbf{h}) &= \frac{1}{S_{\mathcal{P}}} \int_{\mathcal{P}_i} \frac{\partial}{\partial y}(y\, D_x(\mathbf{h}, \mathbf{n}(x)))\mathrm{d}x\mathrm{d}y \\ &= -\frac{1}{S_{\mathcal{P}}} \oint_{\partial \mathcal{P}_i} y\, D_x(\mathbf{h}, \mathbf{n}(x))\mathrm{d}x = -\frac{1}{S_{\mathcal{P}}} \hat{D}_i(\mathbf{h}), \end{aligned} \tag{9}$$

where $\oint_{\partial \mathcal{P}_i}$ denotes the integral on the boundary of the scratch-footprint intersection area and $\hat{D}_i(\mathbf{h})$ is the integral on the boundaries (Fig. 7). Suppose there are K segments on the boundary and each segment is represented as $y = S(x) = a_k x + b_k$, the boundary integral $\hat{D}_i(\mathbf{h})$ is defined as:

$$\hat{D}_i(\mathbf{h}) = \sum_{k=1}^{k=K} \int_{p_x^k}^{p_x^{k+1}} S(x) D_x(\mathbf{h}, \mathbf{n}(x))\mathrm{d}x, \tag{10}$$

where p_x^k and p_x^{k+1} are the x component of segment k's endpoints. Note that the segment where a_k goes to infinity (in other words, x remains a constant value) has zero contribution to the integral, but we keep this unified representation for simplicity. The integrand here is a linear function multiplied with Eqn. (7), indicating that it is a rational function and its closed-form indefinite integral can be calculated. In Figure 7, we show all the possible intersection types, which can be handled in the same way. The details can be found in the supplementary.

Combination with the surface BRDF. Besides the scratch BRDF, we also need to model the appearance on the surface without scratches. We define the final BRDF as:

$$f(\omega_i, \omega_o) = f_{\text{scratch}}(\omega_i, \omega_o) + \left(1 - \frac{S_{\text{scratch}}}{S_{\mathcal{P}}}\right) f_{\text{surface}}(\omega_i, \omega_o), \tag{11}$$

where S_{scratch} is the area covered with scratches within the pixel's footprint. We use the microfacet model with a GGX NDF for f_{surface} and scale it with the non-scratch surface area.

Note that our scratch BRDF model has a closed-form formulation, which allows for efficient evaluation and differentiable rendering. Both of these characteristics are critical for optimization.

3.4 Inverse parameter estimation

Now, we have a scratch geometric model and a shading model. Both models are analytical and differentiable. The next step is to optimize scratch parameters with these models to achieve the goal

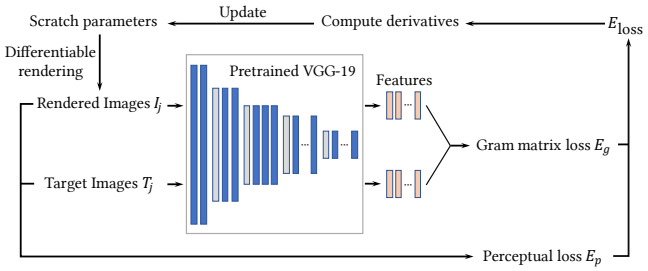


Fig. 8. We perform an optimization on the scratch parameters (endpoints, shapes and roughness) and light intensity, using two losses: a perceptual loss and a Gram matrix loss. Note that our parameter estimation does not rely on any network training, and the pre-trained VGG-19 network is used for loss evaluation only.

formulated in Eqn. (1). We will summarize all scratch parameters to be optimized and then present our loss function.

Optimized parameters. The complete parameters to define scratches include two endpoints for the location, the width and depth for the scratch shape, and the intrinsic roughness in the BRDF: (l_0, l_1, w, d, α). The light intensity is optimized as well. Light positions can be optimized, but we fix them and camera positions for simplicity.

Loss function. We perform a differentiable rendering of our configurations (scratches, light, and view). To measure the difference between the rendered and target images, we need to design an appropriate loss function suitable for images of scratches. Since the scratch appearance is high-frequency, pixel-wise loss functions lead to an over-blur appearance (see Figure 17). Thus, we introduce a perceptual loss and a Gram matrix loss [Gatys et al. 2015].

Inspired by Chizhov et al. [2022], we use halftoning metrics, which express the perceptual loss as a convolution of the error image with a kernel g to approximate the human visual system. The perceptual loss between target images and rendered images is defined as

$$E_p = \sum_j ||g * (I_j - T_j)||_2^2, \tag{12}$$

where the kernel g is low-pass, set as a 3×3 Gaussian kernel.

Then, we introduce a Gram matrix loss with a pre-trained VGG-19 network to represent an image's features. The Gram matrix loss E_g is defined as sum of the squared distances between features of the rendered images and the target images:

$$E_g = \sum_j ||\text{Gram}(\hat{I}_j) - \text{Gram}(T_j)||_2^2. \tag{13}$$

Our final loss function includes the above components:

$$E_{\text{loss}} = \lambda_p E_p + \lambda_g E_g, \tag{14}$$

where λ_P and λ_g denote weights of loss terms respectively. We set $\lambda_p = 1$ and $\lambda_g = 0.01$. We find that E_p makes the scratch's characteristics (sharp boundaries) more clear and E_g removes some scratch outliers. This joint loss will drive the back-propagation to optimize the predicted parameters. The optimization process is shown in Figure 8.

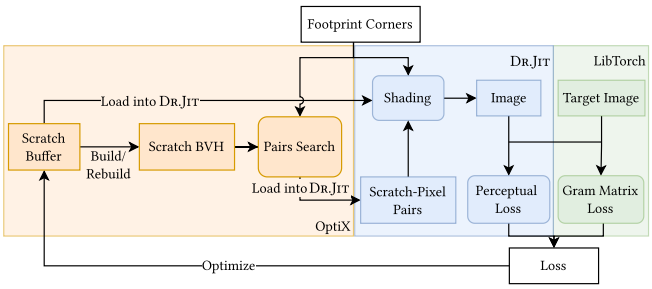


Fig. 9. Starting from an initialized *scratch buffer* described by scratch parameters, we build a scratch BVH in OptiX and shoot rays to find the intersecting scratches. Then, we load all the pixel-scratch pairs into Dr.Jit as a buffer, and evaluate the BRDF by accumulating the contribution of each scratch, resulting in a rendered image. Then a joint loss is computed between the rendered and target images.

3.5 High-performance optimization pipeline

The optimization time cost highly depends on the rendering cost, where the critical point is searching the scratches covered by the pixel's footprint. One typical way is treating each scratch as a curve, building a bounding volume hierarchy (BVH), and traversing this hierarchy to quickly locate the covered scratches by a pixel. Unfortunately, none of the existing high-performance differentiable renderers (e.g., diffrast [Laine et al. 2020]) support curves. Therefore, we design a hybrid framework by combining OptiX [Parker et al. 2013] and Dr.Jit [Jakob et al. 2022].

In our framework (Figure 9), we use OptiX for efficient scratch geometry intersection. This way, we can benefit from the modern graphics hardware's ray tracing cores on the computational-heavy scratch searching task. Then, we evaluate the BRDF with the found scratches to generate rendered images in Dr.Jit. Later, the loss is also computed in Dr.Jit, which drives the back-propagation to optimize scratch parameters. Thanks to our framework, a five-view optimization on 80,000 scratches takes 4 minutes only.

4 IMPLEMENTATION DETAILS

In this section, we provide the implementation details and then show the manufacturing settings.

4.1 Optimization pipeline

Starting from a set of scratches, we represent each one with a curve primitive in OptiX and organize the scratches with the built-in BVH. To make the built-in curve intersection module more effective, we dither the curves along the z-axis to avoid overlapping. During rendering, we shoot a ray from each footprint's center along the $-z$ direction and traverse the BVH to find all the intersecting scratches. Note that we set each scratch's width as the maximum radius of footprints so that the searching ray does not miss any intersecting scratches. The modified width is used for traversal only. For this step, we use the plain OptiX interface instead of the one integrated in Dr.Jit because it does not support curve intersection yet. Moreover, we use a queue data structure to collect the intersection pairs, which is not convenient in Dr.Jit. All the found scratch data are transferred to Dr.Jit for further BRDF evaluation.

The following steps are done in Dr.Jit (C++). We gather the contribution of each scratch, computed by Eqn. (9). Here, we evaluate the NDF value for each pixel-scratch pair and atomically add them to each pixel. Such an operation is named *scatter reduce* in Dr.Jit, whose auto differentiation is supported. Finally, we compute the final BRDF value by Eqn. (11) and set the pixel color as the product of the BRDF value, the light intensity, and the cosine term. Note that we only use point or directional light sources in our optimization, which do not need Monte Carlo sampling.

We perform a Gamma correction for each rendered image since the target images have low dynamic range. Then we compute the joint loss between the rendered and target images, where the VGG-19 network for the Gram matrix loss is called with LibTorch. We use the Adam optimizer with a learning rate of 0.001 for 100 iterations.

After the optimization, we discard the scratches with a low contribution to the rendered results, since we find that they have little influence on the loss function, as shown in Figure 21. In this way, many lines can be discarded, reducing manufacturing overhead.

View/light setup. Since we do not optimize the view and light positions in our implementation, we provide some tips on their setup. Assuming the light is placed with the elevation angle (the angle between the surface normal and light direction) set as θ, the elevation for all views should also be close to θ. Then, the j^{th} view's azimuth angle is set as $\frac{360j}{M}$, where M is the view count.

4.2 Manufacture setup

We use smooth aluminum boards as our reflectors, which are square-shaped with an edge length of 30 centimeters. Although perfectly flat metallic boards are desired, they still have some height variations, from 0.1 to 0.2 mm.

We manufacture the reflectors with a home-built carving machine (Figure 10). The machine is driven by stepper motors and leadscrews. The relocating precision is up to 5μm. One critical component of the carving machining is a *soft carving head*, which consists of a sphere-headed milling cutter and a self-designed spring holder. The purpose of the spring holder is to allow a 1 to 2 mm offset along the vertical direction to overcome the height variations on the metallic boards so that the milling cutter can reach the reflector surface.

The moving speed of the carving head is 1.5cm/s on average, and the average carving speed is one scratch per second. Thus, carving one board takes 5 to 10 hours, depending on the scratch count.

5 RESULTS

We first show rendered results of our forward model and analyze the main parameters. Next, we exhibit our designed reflectors with both virtual rendered results and manufactured ones. Lastly, we analyze the impacts of some essential components. Unless otherwise specified, we use the following settings: α is set as 0.04. $\widetilde{w}$ and d are set as 1×10^{-4}. All timings in this section are measured on a single NVIDIA 3090 GPU and an AMD Ryzen 9 5950X CPU with 16 cores and 32 threads.

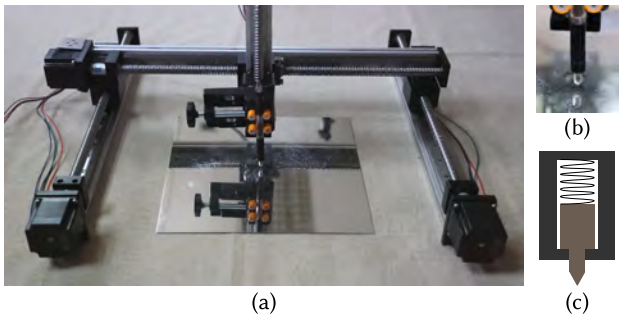


Fig. 10. (a) Our carving machine is driven by stepper motors and leadscrews. (b) Close-up view of the carving head. (c) The milling cutter is supported by a spring, to provide stable stress when carving.

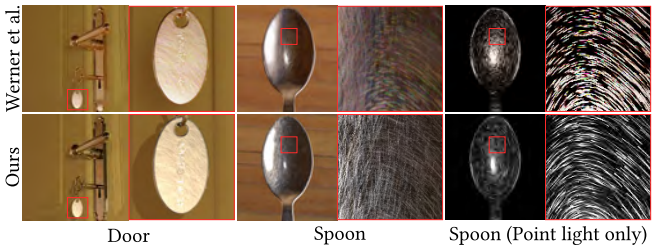


Fig. 11. Comparison between our scratch model and Werner et al. [2017] on two scenes. Our method produces similar scratch appearance (without considering the colors caused by wave optics) as Werner et al. [2017], with much less time cost, about 400× faster.

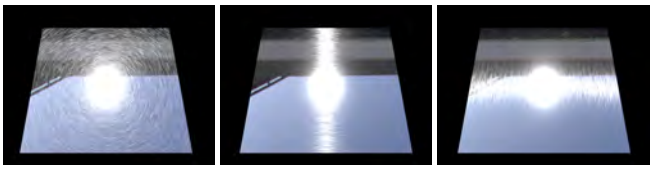


Fig. 12. Our forward scratch model can express different types of scratch distributions by setting scratches' endpoints, including (a) random, (b) horizontally-brushed, and (c) vertically-brushed scratches.

5.1 Results of our forward model

Comparison with previous works. We compare our forward model against Werner et al. [2017] on two different scenes. The implementation of Werner et al. [2017] is from the author's website. Our model can produce similar scratch shapes, while our BRDF evaluation speed is 400 times faster than theirs and purely analytical. The purpose of this comparison is that our scratch rendering method can produce a high-quality scratch appearance as the offline rendering method. Note that the rendered image of Werner et al. [2017] shows colored scratches since their model is based on wave optics. However, we do not consider wave optics, since fabricating with wave optical effects controlled demands expensive devices.

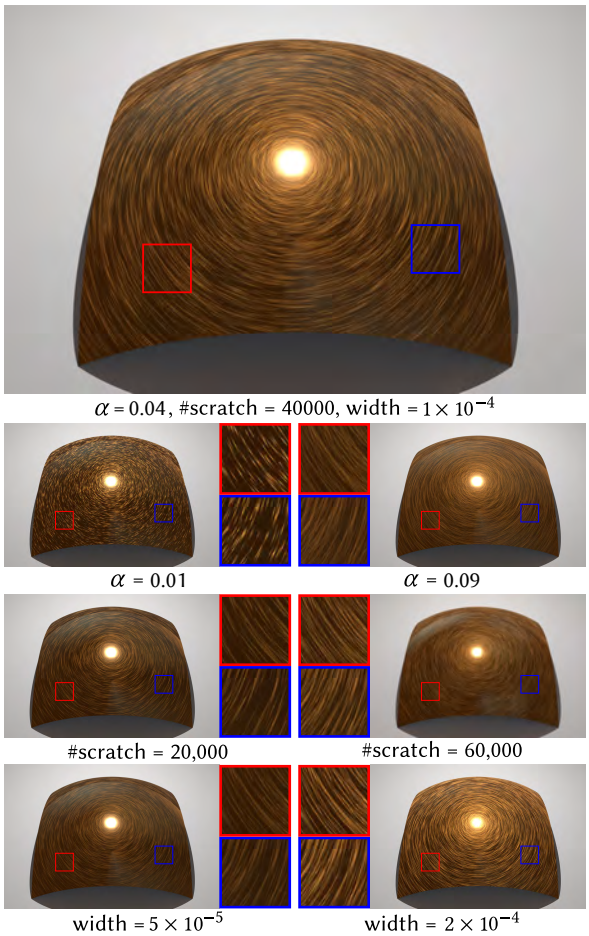


Fig. 13. Rendered results with varying parameters on the BentQuad scene. Our scratch shading model can express a broad range of scratch appearances. Note that unspecified parameters are set the same as the top row.

Parameter analysis. In Figure 12, we show three types of scratches on a PlanarBoard scene under an environment map, including random, horizontally-brushed, and vertically-brushed distributions. These different distributions are defined by setting scratches' endpoints. In Figure 13, we provide rendered results of a BentQuad scene under a point light and environment map with varying parameters, including the intrinsic roughness, width, and scratch count. By setting different scratch parameters, our scratch BRDF can express a variety of scratch appearances.

5.2 Results of our inverse model

Quality validation. In Figure 14, we show three-view rendered results of our inverse model on four examples with 60,000 scratches. For each example, we set three target images as inputs and then optimize scratch and light parameters to match these target images. As a result, three images made of scratches are displayed under different view/light settings on a single same planar board. These rendered images have a good agreement with the target images. It demonstrates the effectiveness of our inverse model.

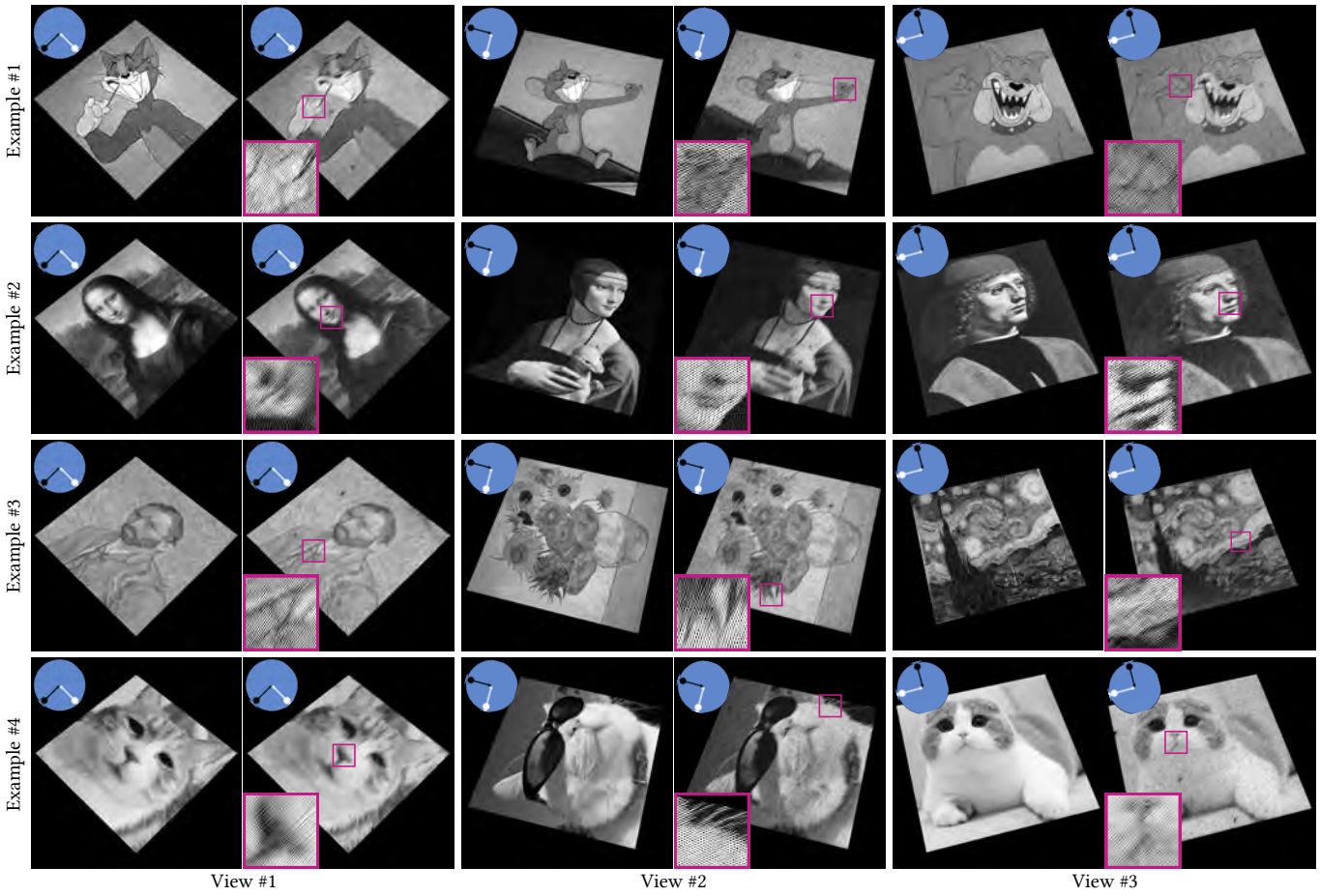


Fig. 14. Four examples of three-view optimization. Each row shows the target and rendered images for three views. Our rendered images have great agreement with the target images. The scratch count is set as 60,000. The black and white dots on the blue circle represent the settings of the view/light, respectively.

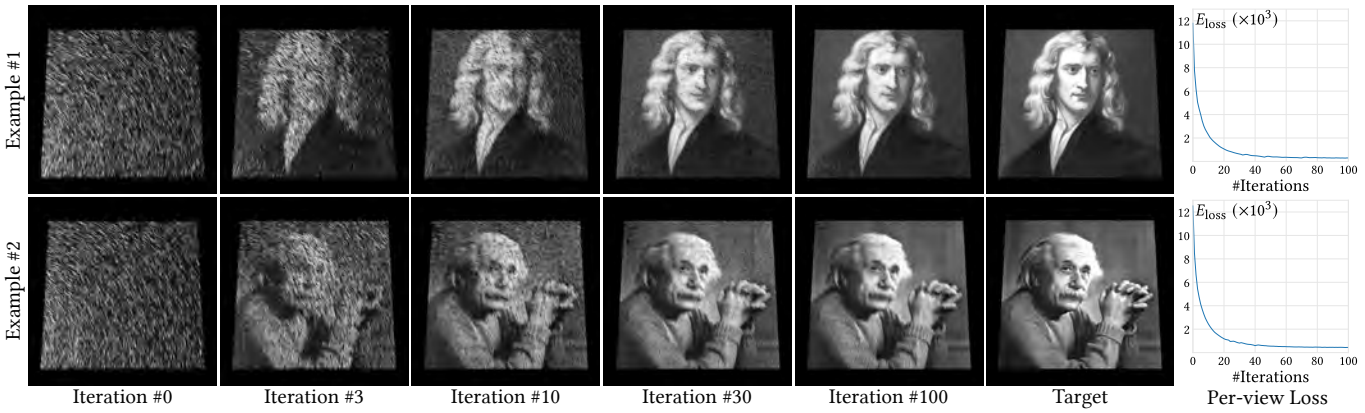


Fig. 15. The rendered results during optimization (#0, #3, #10, #30 and #100) on two examples. We also show the per-view loss as a function of iterations for these two examples. The scratch count is 40,000. An animated version, including all optimized views, can be seen in the supplemental video.

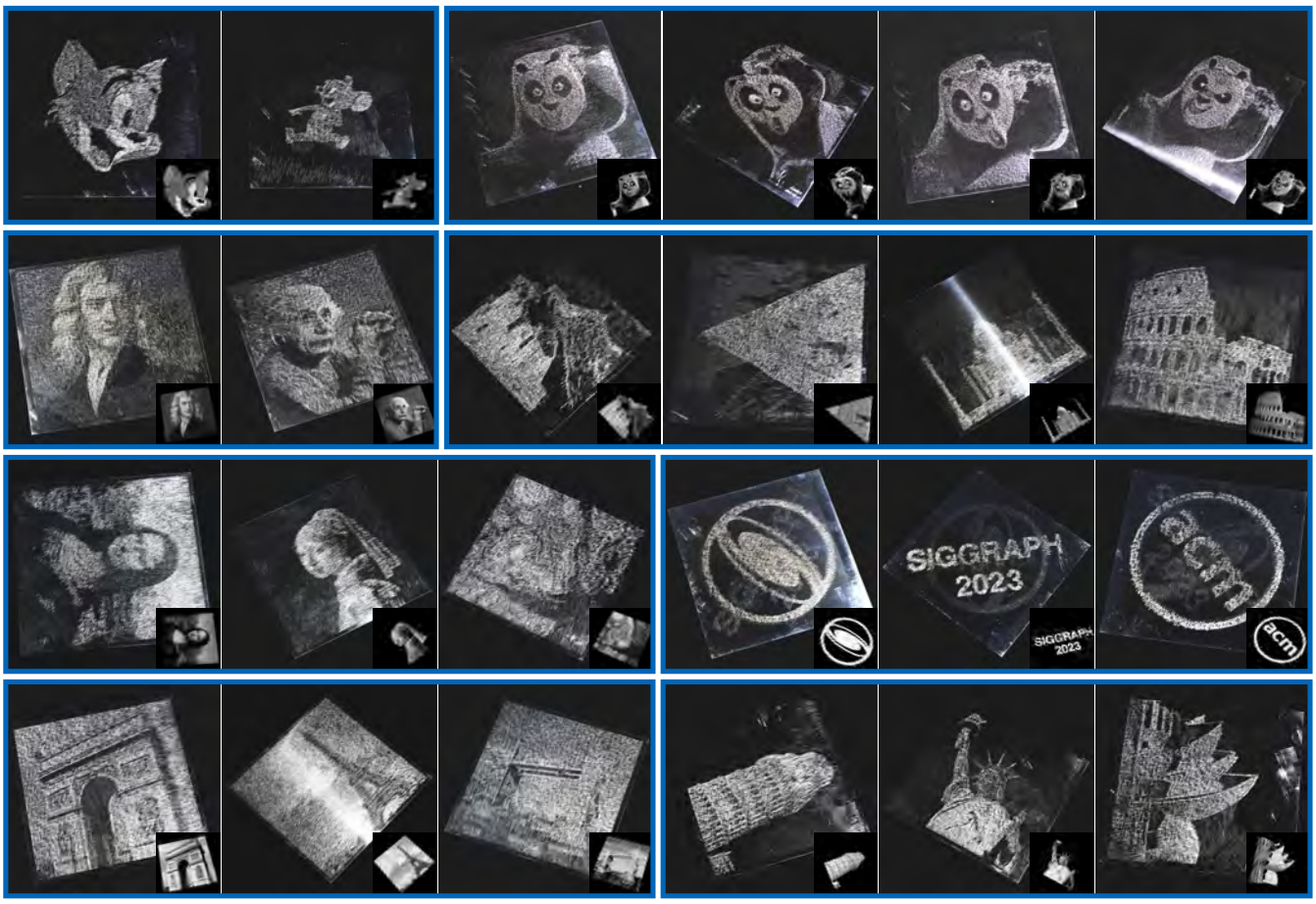


Fig. 16. We design 3D scratch-based reflection arts with optimization and manufacture them on metallic boards using a carving machine (Figure 10). Here, we show photos of eight manufactured metallic boards with different view settings. The views for a single board are marked in a blue quad.

In Figure 15, we show the rendered results during optimization (with iteration #0, #3, #10, #30 and #100) on two examples. Starting from an initial state ($\alpha = 0.04$, $\widetilde{w} = 2.5 \times 10^{-4}$, $d = 1.28 \times 10^{-4}$, randomly-distributed endpoints), the rendered image consisting of scratches approaches the target image with more iterations and almost matches the target state with 100 iterations. An animated version, including all optimized views, can be seen in the supplemental video. Note that this optimization costs 100 seconds. We also provide the loss curves for the given view over varying iterations, which converge smoothly.

We also manufacture our design with a simple setup (Section 4.1) and produce many pieces of 3D scratch reflection art. Since we cannot control the intrinsic roughness when manufacturing, we fix α to be 0.04 during optimization. In Figure 16, we show the photos of eight metallic boards designed with different views (two boards with two views, four boards with three views, and two boards with four views).

In Figure 18, we show the rendered images and photos of a manufactured reflector under non-optimal view/light settings, by rotating or zooming in the camera. As expected, the rendered image and the manufactured reflector no longer match the target image.

Ablation study. We validate the influence of each component in the optimization, including the perceptual loss and the Gram matrix loss. We compare the optimized results with the squared L_2 loss, the perceptual loss E_p only, and our joint loss. The squared L_2 loss produces an over-blur appearance, where the scratch characteristic is less obvious. Introducing the perceptual loss makes the scratches more visible. The reason is that the per-pixel nature of L_1 and L_2 losses tends to smoothen the high-frequency features of scratches, leading to an over-blur appearance. On the contrary, the perceptual loss can focus more on the overall luminance distribution by using low-pass kernels. However, some outliers are noticeable with E_p only. The Gram matrix loss removes these outliers, making the results more faithful and matching the target images better.

We show the influence of intrinsic roughness on our optimization quality in Figure 19. We compare the optimized results w/ and w/o intrinsic roughness. We set the intrinsic roughness as a low value ($\alpha = 4 \times 10^{-4}$) and treat it as pure-specular, since it produces a similar result but avoids singularity. By comparison, we find that

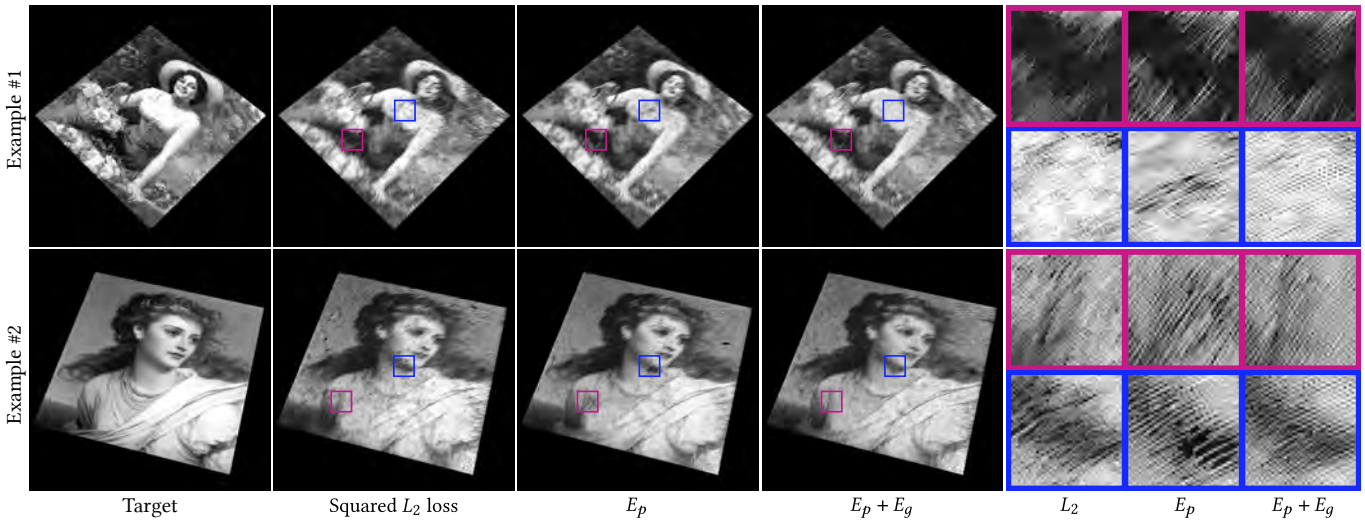


Fig. 17. The impact of each component in our loss function, by comparing squared L_2, our perceptual loss and our joint loss. We show two examples of three-view optimization. In all the results, we optimize 40,000 scratches with 100 iterations. As shown in the insets, the squared L_2 loss leads to a blurry appearance, and the perceptual loss E_p makes the scratches more clear but shows some outliers, which will be removed by introducing the Gram matrix loss E_g.

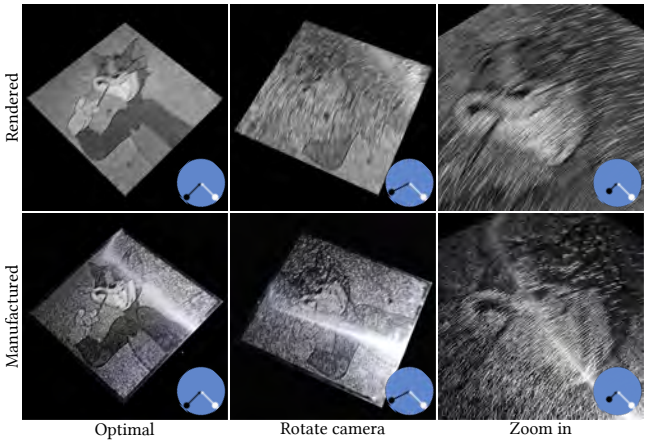


Fig. 18. The rendered and manufactured results using non-optimal settings by rotating the camera 20° (middle) and zooming in the camera (rightmost). As expected, the rendered results under these settings do not match the target images. The scratch count is set as 60,000. The black and white dots on the blue circle represent the settings of the view/light, respectively.

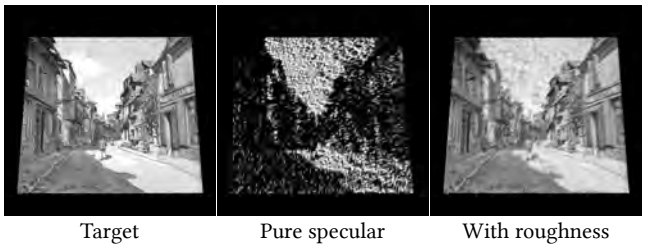


Fig. 19. The impact of intrinsic roughness on the optimization result. Without using intrinsic roughness (b), the optimized result is far from the target image. The scratch count is set as 20,000.

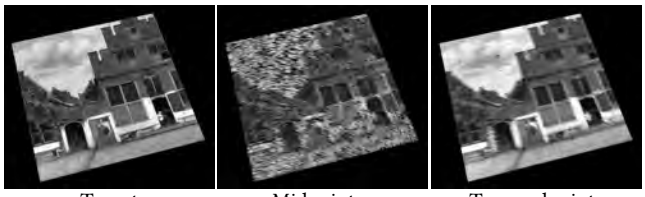


Fig. 20. Comparison between different straight scratch representations: (b) midpoint + direction + length and (c) two endpoints (our choice). Our choice has a better agreement with the target image. This is a single-view optimization of 20,000 scratches with 200 iterations.

the specular scratch material leads to an over-dark appearance while introducing the intrinsic roughness overcomes this issue.

We also show the impact of scratch representation in Figure 20 and the scratch discarding strategy in Figure 21. We compare our two-endpoint representation against the midpoint+direction+length representation. Our two-endpoint representation matches the target image better, since optimizing endpoints makes scratches move more freely and quickly. In Figure 21, we show that our discarding strategy helps remove about 26% of scratches, reducing the manufacturing overhead without any quality degradation.

5.3 Discussion and limitations

Expressiveness of our geometric and shading models. Our scratch BRDF encodes the single-bounce interactions between light and surfaces, similar to most microstructure models. But this simplification

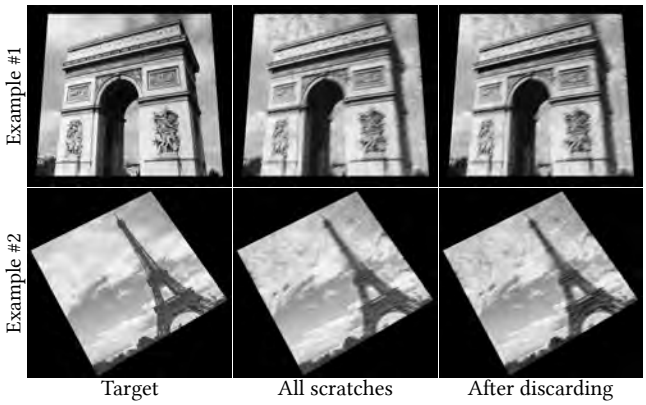


Fig. 21. The influence of scratch discarding strategy, by comparing between the results with all the scratches (60,000) (b) and after discarding (43,884) (c). About 26% of scratches are discarded, while the results are almost identical.

indeed introduces an energy loss. Thanks to our intrinsic roughness, the rendered results match the manufactured scratch appearances.

Conflicts among different views. In theory, our algorithm does not have a limited view count as long as the roughness of scratch can be low. In this case, each scratch only affects the current view without introducing ghosting artifacts in other views. However, in our experiments, we notice that an image pattern specified for one view might also be seen by another, caused by the conflicts of scratches. We tried to perform the decoupled (per-view) optimization, but it did not show any benefits. Since our work aims to introduce differentiable rendering for scratch-based reflector design, we did not handle conflicts with extra effort and leave this improvement for future work.

Advanced manufacturing machines. Since scratch is a simple fabricating element, we can create many interesting scratch reflectors even with a simple manufacturing machine. More professional machines could improve manufacturing quality since our optimization model is not specific to the current manufacturing setup.

Difference from hatching. Hatching [Philbrick and Kaplan 2019; Praun et al. 2001] is a classical art form, which creates flexible, simple patterns using parallel curves. Our work might look similar to hatching, but they are essentially different. At the core of our work is modifying the microstructure of the surface to create multiple desired patterns. Differently, hatching techniques form the patterns by using the curves' tone, density, and pattern.

UV mapping distortion. We do not consider the curvature and treat the footprint as locally flat, following most of the microstructure-based rendering models in the rendering domain. We do not explicitly handle the UV-to-world distortion and did not notice any artifacts as long as the parameterization has good angle/area-preserving properties.

Optimizing scratch count. We do not optimize the number of scratches for now, set as 40,000-80,000 in practice. Since optimizing a discrete value is not trivial, we leave it for future work.

6 CONCLUSION

In this paper, we have presented a new type of 3D visual reflection art: scratch-based reflection art, where different images made of scratches are exhibited under different view/light conditions. The scratch element allows a more compact parameter space and easier fabrication. Then, we formulate the design of scratch art as an optimization problem and introduce differentiable rendering to solve this problem. Thanks to our novel analytical scratch geometric and shading models, together with the high-performance rendering pipeline, our optimization has a low computation cost, and each view greatly agrees with the target image. We further fabricate our designed reflectors with a simple carving machine to demonstrate the usefulness of the proposed method.

To our best knowledge, it is the first time that a complex shape (in UV space) consisting of many curves and the appearance are optimized simultaneously. It is a challenging task since the geometric model and scratch BRDF must be differentiable and have high performance. We hope that our work will inspire many applications, like hair reconstruction. Although we focus on scratch-based reflection art in our paper, our framework is capable of optimizing other elements with a well-designed geometric model and BRDF. We are also interested in designing scratched-based reflection art on non-planar objects, which needs more consideration on the macro geometry.

ACKNOWLEDGMENTS

We thank the reviewers for the valuable comments. This work is supported by the National Key R&D Program of China (2022YFB3303400) and the National Natural Science Foundation of China under grant No. 62025207 and 62172220.

REFERENCES

Asen Atanasov, Alexander Wilkie, Vladimir Koylazov, and Jaroslav Křivánek. 2021. A Multiscale Microfacet Model Based on Inverse Bin Mapping. *Computer Graphics Forum* 40, 2 (2021), 103–113.

Vassillen Chizhov, Iliyan Georgiev, Karol Myszkowski, and Gurprit Singh. 2022. Perceptual Error Optimization for Monte Carlo Rendering. *ACM Trans. Graph.* 41, 3, Article 26 (mar 2022), 17 pages.

Francesco de Comite and Laurent Grisoni. 2015. Numerical Anamorphosis: An Artistic Exploration. In *SIGGRAPH ASIA 2015 Art Papers* (Kobe, Japan) *(SA '15)*. Association for Computing Machinery, New York, NY, USA, Article 1, 7 pages.

Hong Deng, Yang Liu, Beibei Wang, Jian Yang, Lei Ma, Nicolas Holzschuch, and Ling-Qi Yan. 2022. Constant-Cost Spatio-Angular Prefiltering of Glinty Appearance Using Tensor Decomposition. *ACM Transactions on Graphics* 41, 2 (2022), 22:1–22:17.

Oskar Elek, Denis Sumin, Ran Zhang, Tim Weyrich, Karol Myszkowski, Bernd Bickel, Alexander Wilkie, and Jaroslav Křivánek. 2017. Scattering-aware Texture Reproduction for 3D Printing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 36, 6 (2017), 241:1–241:15.

Luis E. Gamboa, Jean-Philippe Guertin, and Derek Nowrouzezahrai. 2018. Scalable Appearance Filtering for Complex Lighting Effects. *ACM Trans. Graph.* 37, 6, Article 277 (dec 2018), 13 pages.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. A Neural Algorithm of Artistic Style. *CoRR* abs/1508.06576 (2015). arXiv:1508.06576

Daniel Glasner, Todd Zickler, and Anat Levin. 2014. A Reflectance Display. *ACM Trans. Graph.* 33, 4, Article 61 (jul 2014), 12 pages.

Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. 2022. DR.JIT: A Just-in-Time Compiler for Differentiable Rendering. *ACM Trans. Graph.* 41, 4, Article 124 (jul 2022), 19 pages.

Alexandr Kuznetsov, Miloš Hašan, Zexiang Xu, Ling-Qi Yan, Bruce Walter, Nima Khademi Kalantari, Steve Marschner, and Ravi Ramamoorthi. 2019. Learning Generative Models for Rendering Specular Microgeometry. *ACM Trans. Graph.* 38, 6, Article 225 (nov 2019), 14 pages.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular Primitives for High-Performance Differentiable Rendering.

CoRR abs/2011.03277 (2020).

Yanxiang Lan, Yue Dong, Fabio Pellacini, and Xin Tong. 2013. Bi-Scale Appearance Fabrication. *ACM Trans. Graph.* 32, 4, Article 145 (jul 2013), 12 pages.

Anat Levin, Daniel Glasner, Ying Xiong, Frédo Durand, William Freeman, Wojciech Matusik, and Todd Zickler. 2013. Fabricating BRDFs at High Spatial Resolution Using Wave Optics. *ACM Trans. Graph.* 32, 4, Article 144 (jul 2013), 14 pages.

Wojciech Matusik, Boris Ajdin, Jinwei Gu, Jason Lawrence, Hendrik P. A. Lensch, Fabio Pellacini, and Szymon Rusinkiewicz. 2009. Printing Spatially-Varying Reflectance. *ACM Trans. Graph.* 28, 5 (dec 2009), 1–9.

Niloy J. Mitra and Mark Pauly. 2009. Shadow art. *ACM Trans. Graph.* 28, 5 (Dec. 2009), 1–7.

Marios Papas, Wojciech Jarosz, Wenzel Jakob, Szymon Rusinkiewicz, Wojciech Matusik, and Tim Weyrich. 2011. Goal-based Caustics. *Computer Graphics Forum (Proceedings of Eurographics)* 30, 2 (June 2011), 503–511. https://doi.org/10/cqjmhv

Steven G. Parker, Heiko Friedrich, David Luebke, Keith Morley, James Bigler, Jared Hoberock, David McAllister, Austin Robison, Andreas Dietrich, Greg Humphreys, Morgan McGuire, and Martin Stich. 2013. GPU Ray Tracing. *Commun. ACM* 56, 5 (may 2013), 93–101.

Greg Philbrick and Craig S. Kaplan. 2019. Defining Hatching in Art. In *ACM/EG Expressive Symposium*, Craig S. Kaplan, Angus Forbes, and Stephen DiVerdi (Eds.). The Eurographics Association.

Petar Pjanic and Roger D. Hersch. 2015a. Color Changing Effects with Anisotropic Halftone Prints on Metal. *ACM Trans. Graph.* 34, 6, Article 167 (nov 2015), 12 pages.

Petar Pjanic and Roger D. Hersch. 2015b. Color Imaging and Pattern Hiding on a Metallic Substrate. *ACM Trans. Graph.* 34, 4, Article 130 (jul 2015), 10 pages.

Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. 2001. Real-Time Hatching. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 581.

Boris Raymond, Gaël Guennebaud, and Pascal Barla. 2016. Multi-Scale Rendering of Scratched Materials Using a Structured SV-BRDF Model. *ACM Trans. Graph.* 35, 4, Article 57 (jul 2016), 11 pages.

K. F. Riley, M. P. Hobson, and S. J. Bence. 2006. *Mathematical Methods for Physics and Engineering: A Comprehensive Guide* (3 ed.). Cambridge University Press.

Kaisei Sakurai, Yoshinori Dobashi, Kei Iwasaki, and Tomoyuki Nishita. 2018. Fabricating Reflectors for Displaying Multiple Images. *ACM Trans. Graph.* 37, 4, Article 158 (jul 2018), 10 pages.

Xavier Snelgrove, Thiago Pereira, Wojciech Matusik, and Marc Alexa. 2013. Special Section on Advanced Displays: Parallax Walls: Light Fields from Occlusion on Height Fields. *Comput. Graph.* 37, 8 (dec 2013), 974–982.

Haowen Tan, Junqiu Zhu, Yanning Xu, Xiangxu Meng, Lu Wang, and Ling-Qi Yan. 2022. Real-Time Microstructure Rendering with MIP-Mapped Normal Map Samples. *Computer Graphics Forum* 41, 1 (2022), 495–506.

Z. Velinov, S. Werner, and M. B. Hullin. 2018. Real-Time Rendering of Wave-Optical Effects on Scratched Surfaces. *Computer Graphics Forum* 37, 2 (2018), 123–134.

Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. 2007. Microfacet Models for Refraction through Rough Surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (Grenoble, France) *(EGSR'07)*. Eurographics Association, Goslar, DEU, 195–206.

Beibei Wang, Miloš Hašan, Nicolas Holzschuch, and Ling-Qi Yan. 2020. Example-Based Microstructure Rendering with Constant Storage. *ACM Trans. Graph.* 39, 5, Article 162 (aug 2020), 12 pages.

Sebastian Werner, Zdravko Velinov, Wenzel Jakob, and Matthias B. Hullin. 2017. Scratch iridescence: wave-optical rendering of diffractive surface structure. *ACM Transactions on Graphics* 36, 6 (Nov. 2017), 1–14.

Tim Weyrich, Pieter Peers, Wojciech Matusik, and Szymon Rusinkiewicz. 2009. Fabricating Microgeometry for Custom Surface Reflectance. In *ACM SIGGRAPH 2009 Papers* (New Orleans, Louisiana) *(SIGGRAPH '09)*. ACM, Article 32, 6 pages.

Kang Wu, Renjie Chen, Xiao-Ming Fu, and Ligang Liu. 2022a. Computational Mirror Cup and Saucer Art. *ACM Trans. Graph.* 41, 5, Article 174 (jul 2022), 15 pages.

Kang Wu, Xiao-Ming Fu, Renjie Chen, and Ligang Liu. 2022b. Survey on computational 3D visual optical art design. *Visual Computing for Industry, Biomedicine, and Art* 5, 1 (Dec. 2022), 31.

Ling-Qi Yan, Miloš Hašan, Wenzel Jakob, Jason Lawrence, Steve Marschner, and Ravi Ramamoorthi. 2014. Rendering Glints on High-Resolution Normal-Mapped Specular Surfaces. *ACM Trans. Graph.* 33, 4, Article 116 (jul 2014), 9 pages.

Ling-Qi Yan, Miloš Hašan, Steve Marschner, and Ravi Ramamoorthi. 2016. Position-Normal Distributions for Efficient Rendering of Specular Microstructure. *ACM Trans. Graph.* 35, 4, Article 56 (jul 2016), 9 pages.

Ling-Qi Yan, Miloš Hašan, Bruce Walter, Steve Marschner, and Ravi Ramamoorthi. 2018. Rendering Specular Microgeometry with Wave Optics. *ACM Trans. Graph.* 37, 4, Article 75 (jul 2018), 10 pages.

Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. 2014. Poisson-Based Continuous Surface Generation for Goal-Based Caustics. *ACM Trans. Graph.* 33, 3 (May 2014), 1–7.