Appearance-aware Multi-view SVBRDF Reconstruction via Deep Reinforcement Learning

Pengfei Zhu
pfzhu@smail.nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

Jie Guo*
guojie@nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

Yifan Liu
yifan_liu@smail.nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

Qi Sun
qi.sun@smail.nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

Yanxiang Wang
yx_wang@smail.nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

Keheng Xu
kehengxu@smail.nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

Ligang Liu
lgliu@ustc.edu.cn
University of Science and Technology of China
Hefei, China

Yanwen Guo*
ywguo@nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

CCS Concepts

• **Computing methodologies** → **Reflectance modeling**.

ACM Reference Format:
Pengfei Zhu, Jie Guo, Yifan Liu, Qi Sun, Yanxiang Wang, Keheng Xu, Ligang Liu, and Yanwen Guo. 2025. Appearance-aware Multi-view SVBRDF Reconstruction via Deep Reinforcement Learning. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25), August 10–14, 2025, Vancouver, BC, Canada.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3721238.3730718

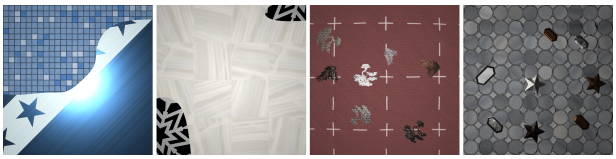

Figure 1: Examples from our MatAugSynth Dataset. The left two are mixture cases created using Perlin noise, while the right two are embedded cases produced by the pattern embedding method.

1 The MatAugSynth Dataset

We utilize two data augmentation methods to create our MatAugSynth dataset. Several examples are displayed in Figure 1.

Pattern embedding. We enhance the heterogeneity of base materials by integrating various patterns, which are assigned metallic material properties. These patterns are then utilized to partially replace a base material, adding complexity and diversity to its appearance. The surface of the base material is divided into a 3×3 grid. Within each section of this grid, we randomly select zero to two pattern centers and apply random scale factors to determine where the pattern will be integrated. The material parameters in these selected areas are replaced with those of the patterns. To introduce further variety, we design the replacement process in two ways. On one hand, we keep the normal map of the base material intact while substituting all other material parameters with those from the patterns. On the other hand, we calculate the unsigned distance function (UDF) of the patterns and convert these into height maps. By randomly adjusting these height values to be positive or negative for one entire pattern, we create effects of raised or recessed surfaces, which are then converted into a normal map. This approach enriches the data by varying the surface normals, thus altering the perceived texture of the material.

Mixture. To further enhance the data's heterogeneity, we adopt a method similar to that in Deschaintre et al. [2020], using low-frequency Perlin noise to generate several masks. Based on these masks, multiple material maps are stitched together.

2 Lightweight Capturing System

To facilitate the convenient collection and reconstruction of real materials, we have developed a multi-view AR scanning system by integrating the Unity AR Foundation environment [1]. In Figure 2, we display a frame from each of the two devices during the

*Joint corresponding authors

SIGGRAPH Conference Papers '25, August 10–14, 2025, Vancouver, BC, Canada

ACM ISBN 979-8-4007-1540-2/2025/08
https://doi.org/10.1145/3721238.3730718

[1] https://unity.com/solutions/xr/ar

Figure 2: The left side shows the view aligned with the axes indicating the camera direction, while the right side shows the view aligned with the axes indicating the light source direction.

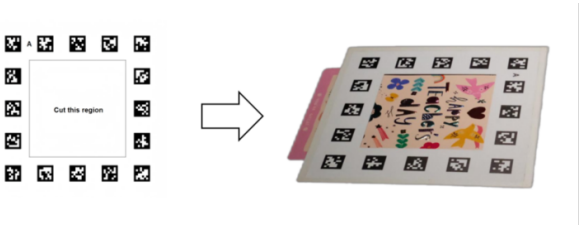

Figure 3: The left is the marker image imported into the Unity client app, and the right is the actual printed marker (with the central area being the material that needs to be photographed). Our client app uses this marker to locate the reference plane.

capturing process. A complete acquisition example is available in the supplementary video.

2.1 Client

The client is responsible for visualizing the sampling directions, capturing images, and uploading them. Based on the Unity AR Foundation framework, the core functionality involves cross-platform calls to underlying APIs such as ARKit and ARCore. These APIs capture the plane on which the marker image resides as the reference plane. The geometric center of this reference plane is used as the origin of a three-dimensional coordinate space, with the reference plane itself defining the X-Y plane, thereby uniquely determining a 3D coordinate system. This enables the acquisition of the real-time pose of the mobile device in space, allowing it to record the current relative position coordinates of the camera and light source to the reference plane during scanning, and to display the next position coordinates of the camera and light source as transmitted by the server. Specifically, as shown in Figure 3, we employ the marker image proposed by MaterialGAN [Guo et al. 2020], which is utilized for both AR recognition and further calibration and warping after the photos are captured.

Our user interface is designed to be minimalist, primarily showcasing the real-time video background captured by the AR camera, overlaid with several UI buttons or prompt texts. The red, blue, and green columns represent the XYZ axes. The yellow column indicates the direction of the next light source position (with the initial coordinates defaulting to being perpendicular to the reference plane), and the white column shows the direction of the next camera viewpoint. Once both the camera and light source are positioned correctly, pressing the "Take Photo" button on the camera device uploads the current image captured by the original camera (without the UI and coordinate axes) to the server. Then the client waits for the next set of coordinates for the camera and light source (or a signal to stop sampling).

2.2 Server

After the "Take Photo" button is clicked, the Unity client uploads the image data to the server using Unity's built-in network interface. The server, built using Express and Node.js [2], receives the photo from the Unity client, and then utilizes the detection method from the open-source project apriltags [3] to recognize the marker image, and extract valid information from the central region, obtaining the input image for the reinforcement learning model. It pushes the calibrated image and directions to the agent through sockets, and waits for the next set of sampling directions (or a stop signal if requirements are met). Once the next sampling directions are received, the server updates the locally stored coordinates, allowing the clients to obtain the latest parameters through continuous request updates.

References

Valentin Deschaintre, George Drettakis, and Adrien Bousseau. 2020. Guided Fine-Tuning for Large-Scale Material Transfer. *Computer Graphics Forum* 39, 4 (July 2020), 91–105. https://doi.org/10.1111/cgf.14056

Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. 2020. MaterialGAN: reflectance capture using a generative svbrdf model. *arXiv preprint arXiv:2010.00114* (2020).

[2] https://expressjs.com/

[3] https://april.eecs.umich.edu/software/apriltag