Supplemental Material for a Unified Spatial-Angular Structured Light for Single-View Acquisition of Shape and Reflectance

In this document, we describe more details about our pipeline, which cannot be squeezed into the main paper due to the page limit.

1. Calibrations

Below we describe the formulation/key steps to calibrate the extrinsic parameters of LEDs and the LCD mask in our imaging pipeline. The basic idea is to minimize the distance of each 2D observation and the corresponding simulated center of a ”light cross”, produced by casting the light from an LED through a cross-shaped opening in the LCD mask onto a planar board. Note that we cast a cross instead of a single pixel here, as it is considerably brighter and thus easier to localize.

Initialization. We first randomly pose a board with ARTags [1] on its surface. Its extrinsic parameters are computed with the tags from a photograph captured with our intrinsic-calibrated DSLR under environment lighting. Then we turn on one LED.

Rough Transform. We further set up the mask to cast two pairs of well-separated, single-pixel-wide horizontal and vertical lines onto the board, resulting in 4 light crosses (see Fig. 1 for an example). The image-space locations of the cross centers are estimated with the algorithm described in Sec. 1.1. Based on the mask-space and image-space coordinates of the cross centers, we call OpenCV.getPerspectiveTransform() to compute the perspective transformation from 2D coordinates in the mask space to the image space for subsequent processing.

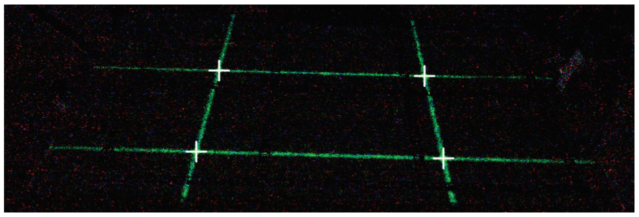

Figure 1. Casting 4 crosses onto a board. The image-based localization results are marked with white crosses. Note that the intensity of photograph has been substantially increased for visualization purpose.

Cross Check. To capture data for extrinsic calibration, we cast a much denser set of crosses for every 10 pixels in the mask (see Fig. 2 for an example). With the aforementioned rough transform, we obtain the 2D location of each cross from its mask-space coordinates. This location is then used to find its nearest neighbor among all image-based cross-center localization results (Sec. 1.1) to establish a correspondence.

Next, we compute the distances between the corresponding pairs of the image-based cross locations and the centers computed with our imaging model. If the maximum distance is larger than 10 or the root mean square error is larger than 5, the entire photograph is discarded due to the large errors. Otherwise, we store these pairs for the next step.

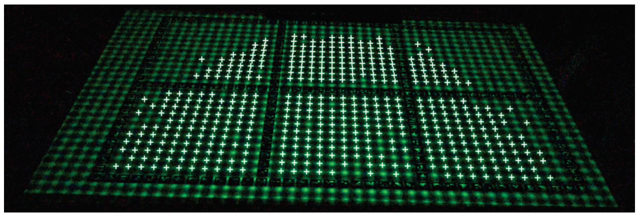

Figure 2. Casting a dense set of crosses onto a board. The detected results are marked with white crosses.

Differentiable Calibration. We randomly pose the board/turn on different LEDs one at a time, and repeat the above process to obtain sufficient pairs of data for calibration. Next, we jointly optimize the LED positions and the transformation from the mask plane to the camera coordinate system in a differentiable manner, by minimizing the sum of squared distances between the image-based cross locations and the corresponding predictions using our imaging model, with the Adam optimizer. Note that in a similar fashion, we can estimate the blur kernel $L(\mathbf{x}_l)$ by minimizing the pixel-wise differences between the photographs and rendering results with our imaging model.

1.1. Cross-center Localization

Image-based cross-center localization consists of two main steps: contour detection and Gaussian fitting. First, we trace contours in the image via OpenCV.findContours(). Then we remove the contours whose areas are less than 10 pixels, and calculate the geometric centers of the remaining contours using the image moments $(M_{10}/M_{00}, M_{01}/M_{00})$ by calling OpenCV.moments(). Due to the acquisition noise, the same cross may be split into a few sets of shorter contours, corresponding to multiple geometric centers that are close to each other. To tackle this, we merge contours whose geometric centers are within a distance threshold of 40 pixels and then recompute the centers for newly merged contours.

Next, initialized with the above centers, we fit anisotropic 2D Gaussians to image patches to more robustly and precisely localize the cross centers. We perform standard nonlinear optimization to find the Gaussian center as well as the covariance matrix, to fit to an 80×80 image patch from the photograph, whose center coincides with the previously computed geometric center. The optimized Gaussian centers are stored as the final localization results.

2. Training Light Patterns for Reflectance

We pre-optimize the light patterns for reflectance acquisition. First, due to the linearity of light transport with respect to sources, the reflected radiance B of a 3D point $\mathbf{x}$ under a light pattern L_k can be expressed as a linear combination of the reflected radiance with one-light-at-a-time (OLAT):

$$B(L_k, \mathbf{x}) = \sum_s L_k(s) m(s; \mathbf{x}). \quad (1)$$

Here s loops over all LEDs, and $L_k(s)$ stores the intensity for the LED with an index of s. $m(s; \mathbf{x})$ is the reflected radiance from $\mathbf{x}$ to the camera, with only one light with an index of s turned on and set to its maximum intensity. The array $\{L_k(s)\}_s$ corresponds to a light pattern, and m is called a lumitexel. Essentially, the physical measurement B can be modeled as the dot product between the light pattern and the lumitexel.

Our goal is to find a small set of $\{L_k\}_k$ that can efficiently probe the physical lumitexel in an compressive manner. To do so, we follow [3] to map both physical acquisition and computational reconstruction to an autoencoder, which takes as input as well as outputs a lumitexel. The weights in the encoder correspond to the light patterns. This allows us to automatically learn the light patterns via minimizing the lumitexel reconstruction error. A large amount of training data can be easily synthesized with random 3D positions in the valid volume as well as random GGX BRDF parameters. Please refer to the original paper [3] for more details.

3. Runtime Optimization of Reflectance

We first apply Xavier initialization [2] to a 1024×1024 neural texture T with 16 channels per texel. For each valid pixel in the reconstructed depth map, we can look up its corresponding *uv* coordinates $\mathbf{u}_i$ to bilinearly sample T to obtain a 16D vector $T(\mathbf{u}_i)$.

Next, we jointly train 5 object-specific networks along with T. Each network is responsible for transforming a 16D neural vector to one set of the following GGX parameters: normal, tangent, diffuse albedo, specular albedo and roughnesses. In terms of architecture, each network is a simple MLP, consisting of 3 hidden layers with 256 neurons in every layer. We employ a leaky ReLU activation for each hidden layer and a sigmoid activation for the final output. With the 3D position $\mathbf{x}_i$ derived from the depth and the GGX parameters, we can simulate the reflectance measurements under different light patterns. Our joint training aims to minimize the per-pixel differences between the photographs and the simulations:

$$T, f = \arg\min \sum_{i,k} ||I^r_{k,i} - R(\mathbf{x}_i, f(T(\mathbf{u}_i)), L_k)||_2. \quad (2)$$

Here i loops over all valid pixels in the depth map, and k over all light patterns. $I^r_{k,i}$ is the captured reflected radiance from $\mathbf{x}_i$ under the light pattern L_k. f is the ensemble of all object-specific networks. Finally, R is a physically-based rendering operation, which takes the 3D position $\mathbf{x}_i$, GGX parameters produced by $f(T(\mathbf{u}_i))$, the light pattern L_k as input, and outputs the simulated result.

References

[1] Mark Fiala. Artag, a fiducial marker system using digital techniques. In *CVPR*, June 2005. 1

[2] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010. 2

[3] Kaizhang Kang, Zimin Chen, Jiaping Wang, Kun Zhou, and Hongzhi Wu. Efficient reflectance capture using an autoencoder. *ACM Trans. Graph.*, 37(4):127:1–127:10, July 2018. 2