Supporting Information for

Fast Non-line-of-sight Imaging with Two-step Deep Remapping

Dayu Zhu[1], Wenshan Cai[1,2]*

[1] School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0250

[2] School of Materials Science and Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0295

* Correspondence should be addressed to W.C. (wcai@gatech.edu)

15 pages

2 figures

3 tables

Contents:

1. Functioning details of the Lidar and experimental details

The device used in the experiment is Intel RealSense L515, with a Lidar, a RGB camera and an IMU (not used in our experiment) integrated in one device. The maximum laser power is 240 mW, which is designed for short-distance indoor applications. During the experiment, the Lidar is set static (< 0.3 m to the wall, facing angle to the wall normal is between 15 to 75 degree). The occluder is set > 0.05 m to the wall to guarantee enough aperture for NLOS imaging. For the detection of every point on the depth map, the Laser diode emits a pulse (~1 ns), then the IR photodiode will receive a train of scattered photons (within 100 ns). The ASIC on the device will process the temporal distribution of received photons and denote the ToF with the highest intensity and further calculate the corresponding depth. Next, the MEMS mirror will alter the Laser to another direction, sequentially scan the FOV in this way and generate the maps. The one-bounce path does not always overshadow multi-bounced paths. As an evidence, if only direct reflections are counted, all the depth maps in Figure 3 of the main text should be the same and show even or gradient depths which represent distances to the wall. However, they are not the case and distinct for different objects, which means multi-reflection information is involved.

In our experiments, the NLOS objects are miniature 3D printed models of daily items. There are 54 objects in total, which fall into the categories of: airplane, baskets, birdhouse, bowl, oven, motorcycle, tower, soundbox, table, rocket, cup, lamp, bell, cupboard, computer, hat, car, and boat. As for the experimental setup, to guarantee enough aperture for NLOS imaging, the distance of the Lidar to the wall should be between 0.15 and 0.30 m, and the facing angle of the Lidar to the wall should be between 15 and 75 degrees. In our experiment, The Lidar is placed at 0.2 m to the wall, 0.2 m to the occluder, with the

facing angle of the Lidar as 45 degree. We set the distance between the occluder and the wall to 0.1 m. In practice, this distance should be large enough (> 0.05 m) to guarantee sufficient aperture for imaging, and in the meantime not too wide (< 0.2 m) to avoid leakage of line-of-sight photons into the Lidar detector.

2. Implementation of NLOS renderer

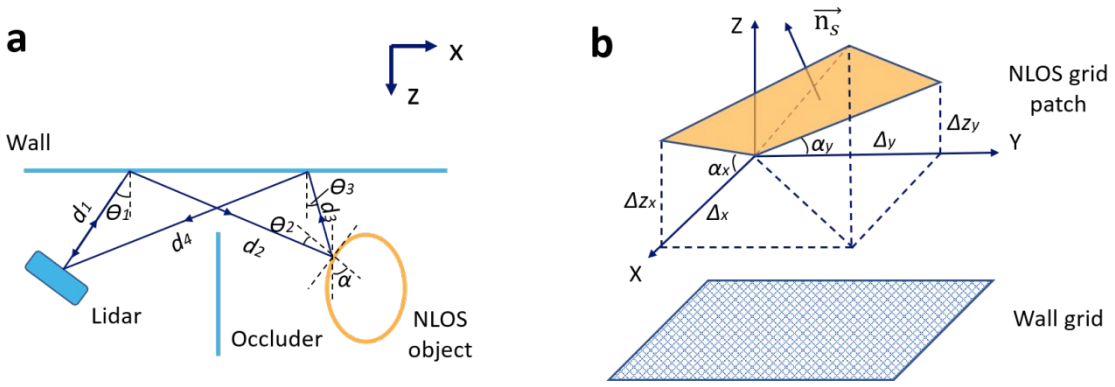

Figure S1. (a) Sketch of renderer operation for direct-reflection and multi-reflection. (b) Details for the calculation of one patch on the NLOS target.

In the implementation of the renderer, we assume all surfaces of the wall and the objects are Lambertian. The surfaces can also be set as any customized BRDF, while here we use uniform Lambertian for general purpose. The renderer assumes millions of beams will be sent along all directions in the FOV of the Lidar, and for each beam the depth and intensity will be calculated based on the reflected light. In the definition of the coordinate system, the wall is on the *xy* plane with z=0. Meanwhile, the NLOS object can be represented by a depth map parallel to the *xy* plane, with the pixel value of each point shows the orthogonal distance to the *xy* plane. We can digitalize both the wall and the NLOS object as grids, thus the bouncing paths between the wall and the object will be the direct mappings between the two grids. The direction and length of the bouncing path can be readily obtained by the

vector connecting the points of the wall and of the object, which will favor fast matrix operations. For a beam sent to a specific direction, as shown in Figure S1a, the reflection path can be both direct-reflected or undergoing multiple bounces. There is only one direct-reflection path but numerous multi-reflection paths. Here we only consider the three-bounce paths and ignore any higher order reflection. Thus, the optical path length for the direct-reflection case is $L_{direct}=2d_1$, while the length for three-bounce multi-reflection case is $L_{multi}=d_1+d_2+d_3+d_4$. As for the light intensity, since the incidence is the same for both cases, we only need to consider the reflection conditions. The direct reflection photon can be expressed as:

$$I_{direct} = \frac{I_0 cos\,\theta_1 S_{unit\_wall}}{2\pi d_1^2}, \tag{S1}$$

where I_0 is the incidence intensity to the wall and we can explicitly set I_0=1. θ_1 is the angle between the incidence direction and the wall normal. S_{unit_wall} is the unit area of the wall grid. Similarly, the case for multi-reflection is:

$$I_{multi} = \frac{I_0 cos\,\theta_1 cos\,\theta_2 cos\,\theta_3 S_{unit\_wall}^2 S_{unit\_target}\beta}{(2\pi)^3 d_2^2 d_3^2 d_4^2}, \tag{S2}$$

where θ_2 is the angle between the second-bounce direction and the surface normal ($\vec{n}_s$) of the incidence point on the NLOS object, and θ_3 is the angle between the third-bounce direction and the wall normal. S_{unit_target} represents the unit area of the NLOS object grid. Since the unit cell on the object may not parallel to *xy* plane, we introduce a surface area ratio, β, to compensate the difference. As depicted in Figure S1b, assume a point on the depth map with coordinate (x_0, y_0, z_0). Then the next grid points along *x* and *y* directions can be denoted as ($x_0+\Delta_x$, y_0, $z_0+\Delta z_x$) and (x_0, $y_0+\Delta_y$, $z_0+\Delta z_y$), respectively, where Δ_x and Δ_y

are the unit lengths of the object grid along x and y directions ($S_{unit\_target} = \Delta x \Delta y$), and Δz_x and Δz_y are the depth changes along x and y directions. Then the patch area ratio around this grid point can be calculated as

$$\beta = \frac{\sqrt{\Delta x^2 \Delta z_y^2 + \Delta y^2 \Delta z_x^2 + \Delta x^2 \Delta y^2}}{\Delta x \Delta y}. \quad \text{(S3)}$$

Since there are millions of possible multi-reflected paths, we collectively group them into bins with resolution of 0.01m, and the intensity for one bin is the intensity summation of all the possible light paths whose lengths fall within the bin. In this way, the intensity for a given optical path length L_{multi} can be expressed as

$$I_{multi}\ (L_{multi}) = \sum^{i}_{(d1 + d2 + d3 + d4 = L_{multi})} I_{multi}(i) V_{multi}(i), \quad \text{(S4)}$$

where V_{multi} is a visibility function: if there is no obstacle in the light path, $V_{multi} = 1$; while if there is a patch of the target blocks the light path, then this multi-bounce path will not be possible and $V_{multi} = 0$.[1] After that, the simulated light intensity of this scanning point will be the maximum between all the multi-reflection and direct-reflection intensities,

$$I_{current\_incidence\_point} = max(I_{direct}, I_{multi\_1}, I_{multi\_2}, I_{multi\_3}, \ldots), \quad \text{(S5)}$$

$for\ all\ L_{multi} < c \times 100\ ns$ with $c = 3 \times 10^8\ m/s$. Then the simulated depth will be the corresponding distance multiplied by the cosine between the incidence direction and the normal vector of the Lidar plane. The final depth and intensity maps consist of 80 by 64 pixels. Since the renderer conducts matrix operations and multiprocessing calculation, the rendering for one object takes only seconds, which is much more efficient than traditional ray-tracing renderers.

3. Structure and training details of the generator

The generator is used for the synthesis of non-line-of-sight (NLOS) scenes, which is the decoder of a variational autoencoder (VAE).[2] To guarantee that sufficient details can be recovered in the generated images, the VAE is not based on the vanilla version which only has linear layers. Instead, the VAE used in this work is a convolutional neural network (CNN), and the skip-connection blocks in ResNet are introduced.[3, 4] The detailed structure of the Res-VAE is presented in Table S1.

During the training phase, the depth maps are reshaped to 64×64 images. Data augmentation techniques, such as random rotation by 90 or 180 degrees and random horizontal flip, are involved. The image pixels are normalized to between 0 and 1. One vital trade-off to be considered is to balance the reconstruction loss and the Kullback–Leibler divergence (KL loss). As the priority is to achieve decent reconstruction quality and the requirement on the distribution of the latent space is relaxed, we can lower the weight of KL loss and set it to be 0.05. As for the training strategy, Adam optimizer is applied, with β_1 = 0.5 and β_2 = 0.999. The original learning rate is 1×10^{-3}, and the ReduceLROnPlateau (PyTorch) scheduler is utilized to carry out the adaptive training. The learning rate will change to 0.3 of the current value if the loss does not drop for 5 epochs. The training stops when the learning rate is reduced to below 1×10^{-8}.

The structure of the Res-VAE for full-color image generation is nearly identical to the above description, with the dimensions of input and output images are 64×64×3. The weight of KL loss is set to be 0.03 for this case. To deal with the increased information in complex full-color scenes, the dimension of latent space is doubled, from 256 to 512.

Encoder	**Res-block 1**	Conv2d (1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (32, eps=1e-05, momentum=0.1)
		Conv2d (32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (64, eps=1e-05, momentum=0.1)
		Conv2d (1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		AvgPool2d (kernel_size=2, stride=2, padding=0)
	Res-block 2	Conv2d (64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (64, eps=1e-05, momentum=0.1)
		Conv2d (64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (128, eps=1e-05, momentum=0.1)
		Conv2d (64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		AvgPool2d (kernel_size=2, stride=2, padding=0)
	Res-block 3	Conv2d (128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (128, eps=1e-05, momentum=0.1)
		Conv2d (128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (256, eps=1e-05, momentum=0.1)
		Conv2d (128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		AvgPool2d (kernel_size=2, stride=2, padding=0)
	Res-block 4	Conv2d (256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (256, eps=1e-05, momentum=0.1)
		Conv2d (256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (512, eps=1e-05, momentum=0.1)
		Conv2d (256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		AvgPool2d (kernel_size=2, stride=2, padding=0)
	Res-block 5	Conv2d (512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (256, eps=1e-05, momentum=0.1)
		Conv2d (256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (512, eps=1e-05, momentum=0.1)
		Conv2d (512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		AvgPool2d (kernel_size=2, stride=2, padding=0)
	Encoder output	(mean) Conv2d (512, 256, kernel_size=(2, 2), stride=(2, 2))
		(variance) Conv2d (512, 256, kernel_size=(2, 2), stride=(2, 2))

Decoder	**Res-up-block1**	Conv2d (256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (256, eps=1e-05, momentum=0.1, affine=True)
		Conv2d (256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (512, eps=1e-05, momentum=0.1, affine=True)
		Conv2d (256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		Upsample (scale_factor=2.0, mode=nearest)
	Res-up-block2	Conv2d (512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (256, eps=1e-05, momentum=0.1)
		Conv2d (256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (512, eps=1e-05, momentum=0.1)
		Conv2d (512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		Upsample (scale_factor=2.0, mode=nearest)
	Res-up-	Conv2d (512, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

	block3	BatchNorm2d (128, eps=1e-05, momentum=0.1)
		Conv2d (128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (256, eps=1e-05, momentum=0.1)
		Conv2d (512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		Upsample (scale_factor=2.0, mode=nearest)
	Res-up-block4	Conv2d (256, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (64, eps=1e-05, momentum=0.1)
		Conv2d (64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (128, eps=1e-05, momentum=0.1)
		Conv2d (256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		Upsample (scale_factor=2.0, mode=nearest)
	Res-up-block5	Conv2d (128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (32, eps=1e-05, momentum=0.1)
		Conv2d (32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (64, eps=1e-05, momentum=0.1)
		Conv2d (128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		Upsample (scale_factor=2.0, mode=nearest)
	Res-up-block6	Conv2d (64, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (16, eps=1e-05, momentum=0.1)
		Conv2d (16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		BatchNorm2d (32, eps=1e-05, momentum=0.1)
		Conv2d (64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
		Upsample (scale_factor=2.0, mode=nearest)
	Decoder output	Conv2d (32, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

Table S1. Structure of the Res-VAE

4. Structure and training details of the compressor

To extract the NLOS information from detected maps and achieve dimension reduction, a neural network-based compressor is pivotal. The compressor is adapted from MobileNetV2, which is a compact network with only 2.5 million trainable parameters.[5] The structure details of the compressor are clarified in Table S2.

When trained on the synthetic dataset, the input to the compressor is 4-channel tensors with split-and-stack depth and intensity maps. The left-depth and left-intensity channels are added with random Gaussian noise with standard deviation of 0.2. The tensor is resized to 100×100×4 with channel-wise normalization. The original learning rate is 0.01, and the

ReduceLROnPlateau scheduler is utilized to perform adaptive training, with factor of 0.3 and patience of 5 epochs. The training process would stop when the learning rate is reduced to below 1×10^{-8}.

As for the training on the real-world dataset, transfer learning is utilized. The preprocessing for the real-world dataset is the same as the synthetic one. The model pretrained on the synthetic dataset is used, and further trained on the real-world dataset for 3-5 epochs with a learning rate of 1×10^{-3}. Furthermore, all the layers are frozen except for the last classifier (the dropout layer and the final linear layer), then the training process carries on until the learning rate is reduced to lower than 1×10^{-8}. Based on our observation, the loss will not decrease further after 10 epochs, so early-stop of the training is also practical.

The compressor for full-color NLOS imaging is based on ResNet50, with 24.5 million trainable parameters. The model is widely used and we adopted this model with limited modifications (only the input and output layers are adapted to fulfill the dimensions of data). The training details are similar to those for the synthetic dataset.

First convolution	Conv2d (4, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
	BatchNorm2d (32, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
InvertedResidual_1	Conv2d (32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (32, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (32, 16, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (16, eps=1e-05, momentum=0.1)
InvertedResidual_2	Conv2d (16, 96, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (96, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (96, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
	BatchNorm2d (96, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)

	Conv2d (96, 24, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (24, eps=1e-05, momentum=0.1)
InvertedResidual_3	Conv2d (24, 144, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (144, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (144, 144, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (144, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (144, 24, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (24, eps=1e-05, momentum=0.1)
InvertedResidual_4	Conv2d (24, 144, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (144, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (144, 144, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
	BatchNorm2d (144, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (144, 32, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (32, eps=1e-05, momentum=0.1)
InvertedResidual_5	Conv2d (32, 192, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (192, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (192, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (192, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (192, 32, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (32, eps=1e-05, momentum=0.1)
InvertedResidual_6	Conv2d (32, 192, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (192, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (192, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (192, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (192, 32, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (32, eps=1e-05, momentum=0.1)
InvertedResidual_7	Conv2d (32, 192, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (192, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (192, 192, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
	BatchNorm2d (192, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (192, 64, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (64, eps=1e-05, momentum=0.1)
InvertedResidual_8	Conv2d (64, 384, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (384, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (384, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (384, 64, kernel_size=(1, 1), stride=(1, 1))

	BatchNorm2d (64, eps=1e-05, momentum=0.1)
InvertedResidual_9	Conv2d (64, 384, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (384, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (384, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (384, 64, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (64, eps=1e-05, momentum=0.1)
InvertedResidual_10	Conv2d (64, 384, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (384, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (384, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (384, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
	BatchNorm2d (64, eps=1e-05, momentum=0.1)
InvertedResidual_11	Conv2d (64, 384, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (384, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (384, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (384, 96, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (96, eps=1e-05, momentum=0.1)
InvertedResidual_12	Conv2d (96, 576, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (576, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (576, 576, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (576, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d(576, 96, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (96, eps=1e-05, momentum=0.1)
InvertedResidual_13	Conv2d (96, 576, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (576, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (576, 576, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (576, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (576, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
	BatchNorm2d (96, eps=1e-05, momentum=0.1)
InvertedResidual_14	Conv2d (96, 576, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (576, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (576, 576, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
	BatchNorm2d (576, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (576, 160, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (160, eps=1e-05, momentum=0.1)

InvertedResidual_15	Conv2d (160, 960, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (960, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (960, 960, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (960, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (960, 160, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (160, eps=1e-05, momentum=0.1)
InvertedResidual_16	Conv2d (160, 960, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (960, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (960, 960, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (960, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (960, 160, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (160, eps=1e-05, momentum=0.1)
InvertedResidual_17	Conv2d (160, 960, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (960, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (960, 960, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
	BatchNorm2d (960, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
	Conv2d (960, 320, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (320, eps=1e-05, momentum=0.1)
Final Convolution	Conv2d (320, 1280, kernel_size=(1, 1), stride=(1, 1))
	BatchNorm2d (1280, eps=1e-05, momentum=0.1)
	ReLU6 (inplace=True)
Classifier	Dropout (p=0.5, inplace=False)
	Linear (in_features=1280, out_features=256, bias=True)

Table S2. Structure of the compressor.

5. Comparison with other methods

A qualitative comparison with other methods is as following:

Methods	LCT[6]	Phasor field[7]	Steady-state[8]	Direct CNN[9]	Deep correlography[10]	Feature embedding[11]	Single pixel[12]	Real-time dynamic[13]	Ours
Category	Physical model	Physical model	Deep learning	Deep learning	Physical model + Deep learning	Physical model + Deep learning	Physical model	Physical model	Deep learning
Equipment	Lab-built confocal system	Lab-built transient system	Lab-built RGB system	Lab-built transient system	Lab-built transient system	Lab-built transient system	Single pixel carema	Lab-built transient system	Commercial Lidar
Data type	Transient data	Transient data	Images	Transient data	Transient data	Transient data	Transient data	Transient data	Depth and intensity maps
Data collection time	Minutes-hours	Minutes-hours	Minutes-hours	Minutes-hours	Milliseconds	Milliseconds	Seconds	Real-time	Milliseconds
Computation time	Seconds-minutes	Seconds-minutes	Milliseconds	Milliseconds	Milliseconds	Milliseconds	Seconds	Real-time	Milliseconds
Reconstruction results	Sharp	Sharp	Blurry, with RGB	Blurry	Sharp	Sharp	Blurry, with RGB	Sharp, with RGB	Sharp, with RGB
Output resolution	Limited by input data	Limited by input data	Unlimited	Unlimited	Unlimited	Unlimited	Limited by input data	Limited by input data	Unlimited

Table S3. Comparison with other NLOS solutions.

6. Error analysis

In this part we analyze the quality of the NLOS reconstruction of the examples (Figure 3b) presented in the main text. The first three rows of Figure S2 are the same as Figure 3b, while the last row presents the difference between the reconstructed depth maps and the ground truth. Provided the very high reconstruction performance (98% accuracy), the error is not evenly or randomly distributed. The results indicate our methodology performs impressive localization capability, since the positions of objects are mostly recovered. Besides, the error is minimized on large or smooth surfaces, while the reconstruction on small surfaces or fine details shows larger error. This performance is also expected, since the reflection conditions are far more complex on small surfaces, thus the NLOS signal may not be collected by the Lidar.

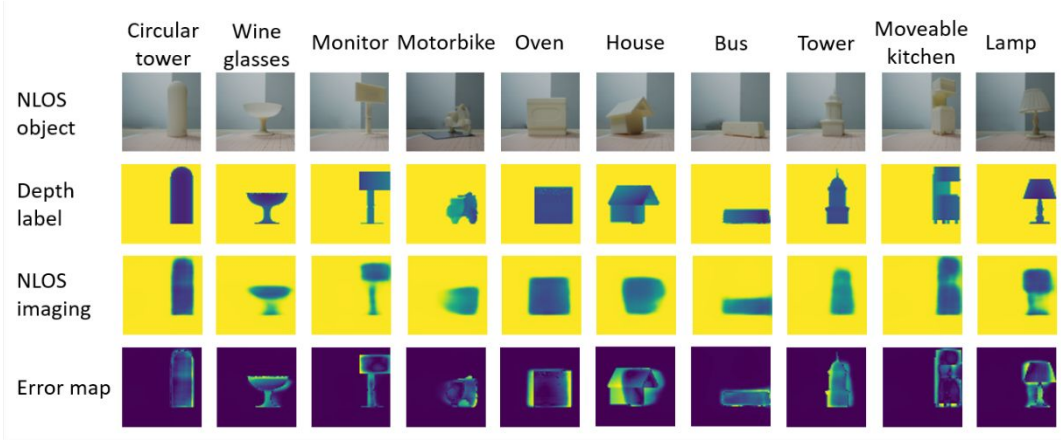

Figure S2. Error analysis of reconstructed maps.

References:

(1) Iseringhausen, J.; Hullin, M. B., Non-Line-of-Sight Reconstruction Using Efficient Transient Rendering. *ACM Transactions on Graphics (TOG)* **2020,** *39* (1), 1-14.

(2) Kingma, D. P.; Welling, M., Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114* **2013**.

(3) Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M., Imagenet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vision* **2015,** *115* (3), 211-252.

(4) He, K.; Zhang, X.; Ren, S.; Sun, J. In *Deep Residual Learning for Image Recognition*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, USA, June; Las Vegas, USA, 2016; pp 770-778.

(5) Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.-C. In *Mobilenetv2: Inverted Residuals and Linear Bottlenecks*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2018; pp 4510-4520.

(6) O'Toole, M.; Lindell, D. B.; Wetzstein, G., Confocal Non-Line-of-Sight Imaging Based on the Light-Cone Transform. *Nature* **2018,** *555* (7696), 338-341.

(7) Liu, X.; Guillén, I.; La Manna, M.; Nam, J. H.; Reza, S. A.; Le, T. H.; Jarabo, A.; Gutierrez, D.; Velten, A., Non-Line-of-Sight Imaging Using Phasor-Field Virtual Wave Optics. *Nature* **2019,** *572* (7771), 620-623.

(8) Chen, W.; Daneau, S.; Mannan, F.; Heide, F. In *Steady-State Non-Line-of-Sight Imaging*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019; pp 6790-6799.

(9) Chopite, J. G.; Hullin, M. B.; Wand, M.; Iseringhausen, J. In *Deep Non-Line-of-Sight Reconstruction*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020; pp 960-969.

(10) Metzler, C. A.; Heide, F.; Rangarajan, P.; Balaji, M. M.; Viswanath, A.; Veeraraghavan, A.; Baraniuk, R. G., Deep-Inverse Correlography: Towards Real-Time High-Resolution Non-Line-of-Sight Imaging. *Optica* **2020,** *7* (1), 63-71.

(11) Chen, W.; Wei, F.; Kutulakos, K. N.; Rusinkiewicz, S.; Heide, F., Learned Feature Embeddings for Non-Line-of-Sight Imaging and Recognition. *ACM Transactions on Graphics (TOG)* **2020,** *39* (6), 1-18.

(12) Musarra, G.; Lyons, A.; Conca, E.; Altmann, Y.; Villa, F.; Zappa, F.; Padgett, M. J.; Faccio, D., Non-Line-of-Sight Three-Dimensional Imaging with a Single-Pixel Camera. *Phys. Rev. Appl* **2019,** *12* (1), 011002.

(13) Nam, J. H.; Brandt, E.; Bauer, S.; Liu, X.; Sifakis, E.; Velten, A., Real-Time Non-Line-of-Sight Imaging of Dynamic Scenes. *arXiv preprint arXiv:2010.12737* **2020**.