Supplementary Material: RADU: Ray-Aligned Depth Update Convolutions for ToF Data Denoising

Michael Schelling, Pedro Hermosilla, Timo Ropinski
Ulm University, Germany
{michael-1.schelling, pedro-1.hermosilla-casajus, timo.ropinski}@uni-ulm.de

1. Further Insights on ToF Working Principle

A convenient formulation for the sinusoidal part v of the measurements m_θ , is to represent it in the complex plane $\mathbb{C}$ via

$$v = A \cdot e^{i\Delta\varphi}, \tag{1}$$

$$Re(v) = \sum_\theta -\sin(\theta) \cdot m_\theta, \tag{2}$$

$$Im(v) = \sum_\theta cos(\theta) \cdot m_\theta. \tag{3}$$

Which allows to express the measurements m_θ as [3]

$$m_\theta = I + A\cos(\Delta\varphi + \theta), \tag{4}$$

$$= I + Re\left(v \cdot e^{i\theta}\right), \tag{5}$$

Using this notation the amplitude A, the intensity I and the the phase delay $\Delta\varphi$ can be recovered as

$$A = \|v\|_2. \tag{6}$$

$$I = \frac{1}{N}\sum_\theta m_\theta, \tag{7}$$

$$\Delta\varphi = arg(v) \tag{8}$$

$$= \arctan\left(\frac{Re(v)}{Im(v)}\right), \tag{9}$$

where N is the number of phase shifted measurements m_θ. Note that this is not an exact solution but a least-squares optimal solution [3].

Given measurements at two different frequencies f_1, f_2 a computationally cheap solution [6] to unwrap the distances is by minimizing

$$\min_{m,n} \left| d(f_1) + m \cdot d_{max}(f_1) - d(f_2) + n \cdot d_{max}(f_2) \right|. \tag{10}$$

We use this approach to compute the phase unwrapped high frequency ToF depth in the experiment on our Cornell-Box dataset.

2. Extension of MC-Convolutions with RADU

Monte-Carlo Convolution The Monte-Carlo point convolution of Hermosilla *et al*. [7] approximates the convolution of two continuous functions, the features f and the kernel function g, where f is only known at the discrete positions $\{p_i\}$, the point cloud, using the Monte-Carlo numerical integration. Formally, let a point cloud $\{p_i^{in}\} \subset \mathbb{R}^3$, input features $\{f_i^{in}\} \subset \mathbb{R}^{C_{in}}$ and a convolution radius $r \in \mathbb{R}$ be given. Further let $\mathcal{N}_j$ denote the neighborhood of p_j^{in}, given as

$$\mathcal{N}_j = \{i \ : \ \|p_j - p_i\|_2 \leq r\} \subset \mathbb{N}. \tag{11}$$

Then the Monte-Carlo convolution on point p_j with radius r is defined as

$$(f * g)(p_j) := |\mathcal{N}_j|^{-1} \sum_{i \in \mathcal{N}_j} \frac{f_i \cdot g\left(\frac{p_i - p_j}{r}\right)}{pde(p_i \mid p_j)}, \tag{12}$$

where $pde(p_i \mid p_j)$ denotes a point-density estimation of p_i inside the receptive field of p_j, and the kernel function $g : \mathbb{R}^3 \to \mathbb{R}^{C_{in} \times C_{out}}$ is represented implicitly using one or multiple MLPs. In analogy to 2D convolutions, the evaluation of g on the relative position $(p_i - p_j)/r$ yields the convolution weight matrix W_{ij}.

RADU To predict the additional point update $p_j^u \in \mathbb{R}$ we extend g to predict an additional output channel, *i.e.*

$$g : \mathbb{R}^3 \to \mathbb{R}^{C_{in} \times C_{out}} \times \mathbb{R}^{C_{in} \times 1} \simeq \mathbb{R}^{C_{in} \times (C_{out}+1)}, \tag{13}$$

$$\frac{p_i - p_j}{r} \mapsto \left(W_{ij}, W_{ij}^u\right). \tag{14}$$

Using this kernel function in Eq. (12), the dimensionality of the output of the convolution is increased by one, *i.e.* $(f * g)(p_j) \in \mathbb{R}^{C_{out}+1}$. We split the output into features $f_j^{out} \in \mathbb{R}^{C_{out}}$, and the point update p_j^u. To summarize the point update is computed as

$$p_j^u := |\mathcal{N}_j|^{-1} \sum_{i \in \mathcal{N}_j} \frac{f_i \cdot W_{ij}^u}{pde(p_i \mid p_j)}. \tag{15}$$

The final update along the associated camera ray is performed as described in the main paper

$$p_j^{out} = p_j^{in} + \alpha \cdot \tanh(p_j^u) \cdot r_j. \tag{16}$$

Computational Complexity of RADU Let C_{in}, C_{out} be the number of input and output features, respectively. The number of parameters $\#P$ of the above described a 3D RADU convolution with hidden dimension h in the kernel MLP is given as

$$\#P_{3D} = h \cdot C_{in} \cdot (C_{out} + 1) + 3h + C_{out}, \tag{17}$$
$$\in \mathcal{O}(h \cdot C_{in} C_{out}). \tag{18}$$

The #P of a 2D convolution with kernel size k is

$$\#P_{2D} = k^2 \cdot C_{in} \cdot C_{out} + C_{out}, \tag{19}$$
$$\in \mathcal{O}(k^2 \cdot C_{in} C_{out}). \tag{20}$$

The leading terms are $h \cdot C_{in}C_{out}$ and $k^2 \cdot C_{in}C_{out}$. In our RADU network we use a hidden MLP dimension of $h = 16$ which places its number of parameters between a 2D convolution with $k = 3$ and $k = 5$.

3. Network Architecture - Hyperparameters

We provide additional parameters of our network architecture for a full description. Our networks consists of three initial 2D convolutions, followed by a 2.5D pooling layer, three 3D RADU convolutions, a 3D-2D projection with up-scaling and a final stack of three 2D convolutions.

The first three 2D convolutions have feature channels of sizes [64, 64, 128]. The 2.5D pooling layer uses a stride of 8×8 and applies average pooling on both the point depths and the features. The 3D RADU convolutions use single MLPs with 16 hidden neurons as kernel functions, and a regularization of the point updates with $\alpha = 0.1$ m. The receptive fields are [0.1 m, 0.2 m, 0.4 m] and the feature channels are of sizes [128, 256, 128]. The up-scaling layer uses bilinear interpolation. The extracted depth of the points is projected back into a 2D depth image and is used as additional input to the following 2D layers. The final 2D convolutions have feature channels of sizes [64, 64, 1].

In between all convolutional layers, 2D and 3D, we use leaky ReLU with $\alpha = 0.1$ as non-linear activation. All 2D convolutions further have a kernel size of 3×3 and use *'same'* padding.

Finally we use a skip connection between the two 2D blocks with feature concatenation.

4. Cornell-Box - Dataset Generation

We generate our dataset using the transient renderer of Jarabo *et al.* [8], which has been deployed for ToF data generation in previous works [5, 10].

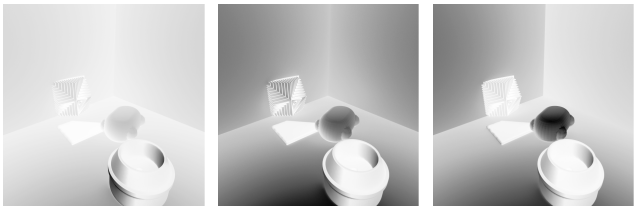


Figure 1. An example scene with three different material properties. The images show the intensity I at a frequency of 20 MHz as given by Eq. (7). In the left scene the box material is highly reflective, the right scene contains an object with a dark material.

Camera Properties Inspired by the work of Miller *et al.* [11], which propose an approach to modify standard cameras for ToF captures, we simulate the properties of a RaspBerry Pi 3 camera equipped with an Electro-Absorption Modulator (EAM), which modulates the received signal s_r in front of the camera sensor.

Scene Generation To ensure challenging scenes with high MPI levels we design our scenes inspired by the Cornell-Box [4] layout, and place a random number of objects, between 1 and 10, in the scene. The used objects meshes are taken from a subset of the Thingi10k dataset [14], containing 3D models under CC license. For each 3D object a material property is randomly sampled, the assignment of dark materials allows to simulate regions with lower SNR values. Further the material of the surrounding box allows, to some degree, control over the level of MPI in the scene. An example of a scene with different materials is shown in Fig. 1.

ToF Simulation Using the transient renderer we compute the impulse response $h(t)$ of the scene, which contains the received signal s_r in a time resolved format after illuminating the scene with a light impulse. Given the impulse response the measurements m_θ can be simulated for different frequencies f and phase offsets θ using

$$s_r(t) = h(t) * s_e(t), \tag{21}$$

in the formula from the main paper

$$m_\theta = \frac{1}{\delta t} \int_{\delta t} s_r(t) \cdot s_e\left(t + \frac{\theta}{2\pi f}\right) dt. \tag{22}$$

The resulting measurements for an example scene are shown in Fig. 2. Further a scene table can be found at the end of this document in Fig. 13, 14 and 15.

Additional Noise We simulate the combination of shot noise, thermal noise and read noise as an Additive White Gaussian Noise (AWGN) using the specifications of the RaspBerry Pi 3 camera of Pagnutti *et al.* [12], who use the linear noise model of the EMVA Standard 1288 [2]. We use the ISO 100 measurements of the former as reference, who measured a gain K of 0.33 and a Y-intercept b of -18.4 on

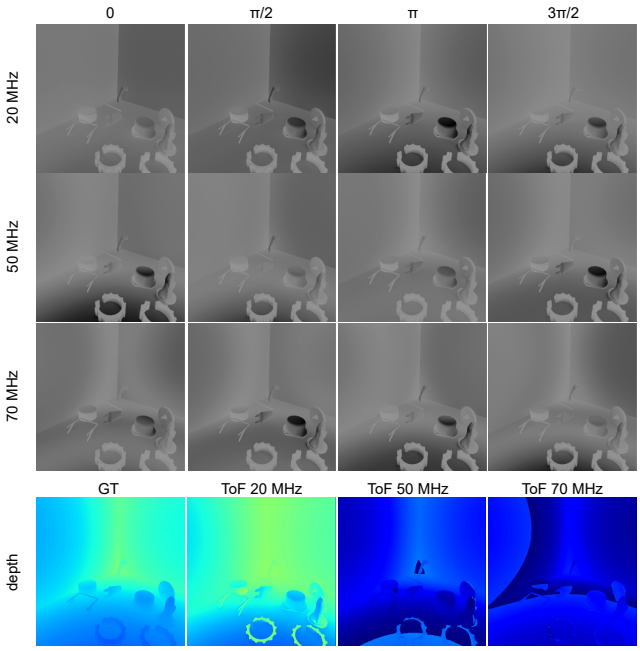


Figure 2. Example scene from our Cornell-Box dataset. Simulated measurements m_θ for different values of $\theta \in \{0, \pi/2, \pi, 3\pi/2\}$ and frequencies $f \in \{20, 50, 70\}$ are shown in the top rows. The reconstructed ToF depths, including phase wrapping, are shown in the bottom row.

the mean-variance curves

$$\sigma^2 = K \cdot m + b, \tag{23}$$

where m is the mean value across multiple measurements and σ^2 is the variance of the measurements. From Eq. (23) we infer the pixel-wise variance $\sigma^2_{x,y}$ dependent on the measurement m_θ on pixel (x, y) for the AWGN $\mathcal{N}(0, \sigma^2_{x,y})$.

In our dataset, we provide measurements m_θ without the AWGN to allow the online generation of AWGN during training for data augmentation, and to allow future researchers to simulate other noise models.

5. Experiments

We briefly describe additional details about the experiments, including data augmentation and hyperparameter settings.

The ADAM optimizer [9] is used during backpropagation in all trainings described below.

5.1. Data Augmentation

To increase the variety in the datasets we use the following data augmentation strategies on the input features.
Mirroring Random mirroring along image axes.
Image Rotation Random rotation by $0°, 90°, 180°, 270°$.
Small Rotation Additional rotation with a random angle in the range $[-5°, 5°]$. Values outside the image boundaries are interpolated with a *nearest* strategy, as implemented in the preprocessing pipeline of `tensorflow.keras`.
Noise Additive Gaussian noise with a relative standard deviation of 0.02.
Random Cropping In the case of training on image patches we crop random regions of the images every epoch.

We further experimented with the MPI augmentation of Agresti *et al*. [1], but found it did not improve the performance in our experiments.

5.2. Soft-Kinect - Datasets S1-S5

Since no raw measurements are available in the dataset, we do not compare to the End2End method in this setting. While in theory a reconstruction of m_θ could be done using Eq. (4), the error accumulation results in data unfit for training.

The input resolution of the Soft-Kinect is 320×240, which results in a latent point cloud of $1.2k$ points after the 2.5D pooling in our proposed network architecture. On the real datasets of size 320×239 we pad the input to the using reflective padding to the full resolution 320×240.

We use the same input features as CFN, by setting $f_1 = 70\text{MHz}, f_2 = 20\text{MHz}, f_3 = 40\text{MHz}$. Note, the provided ToF depth at 70MHz is phase unwrapped in this experiment.
Pre-Training Our network is pre-trained on the synthetic dataset S1 using a learning rate of 1e-3 and an exponential learning rate decay of 0.1 every 100 epochs and a batch size of 8. The network converged after 300 epochs.
Cyclic Self-Training After pre-training our network on synthetic data, we train the network on the unlabeled real dataset S2 using pseudo-labels as described in the main paper. In each training step, we choose real examples with a probability of $p = 0.5$ and update the pseudo-labels every $n_{cycle} = 20$ epochs. We train with a small learning rate of 5e-5 and a batch size of 4 for 100 epochs.
Supervised Domain Adaptation For comparison we finetune our network, after pre-training on synthetic data, using the labeled real dataset S3. We train with a small learning rate of 1e-5 and a batch size of 4 for 100 epochs.

5.3. RaspBerry-Pi 3 - Cornell-Box Dataset

We use a resolution of 512×512 during training which results in 4096 points in the latent point clouds. The input features for our network are computed by using $f_1 = 20\text{MHz}, f_2 = 50\text{MHz}, f_3 = 70\text{MHz}$. The network is trained with an initial learning rate of 1e-3 and an exponential learning rate decay of 0.1 every 100 epochs, and a batch size of 4. The network converged after 300 epochs.
DeepToF includes an Auto-Encoder (AE) training stage on real data for domain adaptation. As there is no real data in this experiment, we train models with pre-training the AE, as described in the original paper, and training the entire

network combined, without pre-training. We also tune the learning rate with initial learning rates from {1e-3, 1e-4} and decay steps from {50, 75, 100}, The learning rate in the AE stage is set constant at 1e-4 for 15 epochs, as suggested in the original paper [10]. The learning rate decay is not fully specified in the original paper, we assume a decay of 0.1 every 75 epochs, which matches the authors description. We use an L2-Loss, a batch size of 16, and the low frequency 20MHz ToF-depth as input as in the original paper.

CFN was investigated in two papers, we use the more recent version [1] as reference for our experiments, which predicts the depth directly and does not use additional filtering algorithms. As no real data is available we drop the unsupervised adversarial part of the training, as with our network architecture. The original paper [1] used a fixed learning rate of 5e-6. We investigate the static learning rates {1e-4, 1e-5, 5e-6} and also in combination with a learning rate decay after {100, 150} epochs. We use a coarse-fine L1-loss and a batch size of 4 as in the original paper [1]. The input features for CFN are the same as for our network.

End2End predicts depths directly from raw correlations, using a generative approach, which we also incorporate into our trainings. The original network uses a static learning rate of 5e-4 for 50 epochs before decaying the learning rate linearly to zero for 100 epochs [13]. We additionally investigate using exponential learning rate decays with initial learning rates from {5e-4, 5e-3} at decay steps from {50, 100}. We further use the combination of adversarial, total-variation and L1-loss of the original paper. The first two raw measurements at phase offsets $0, \pi/2$ of the two higher frequencies 60MHz, 70MHz are used as input.

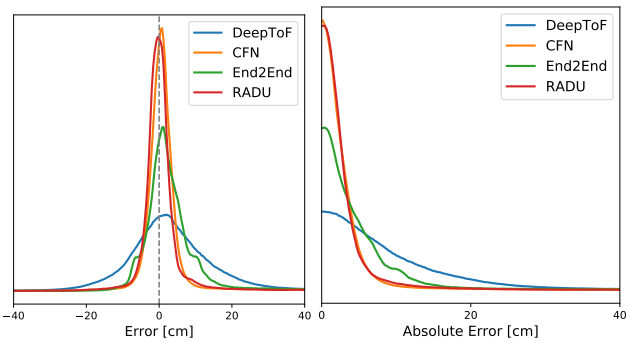
We also investigate the influence of training on image patches of resolution 128×128, as used by CFN and End2End originally. During training we ensure that the cropped images inside a batch are from different scenes.

We compare the resulting MAE on the validation set after tuning and using the orignal hyperparameters (vanilla):

	DeepToF	CFN	End2End
vanilla	11.97	4.72	9.14
tuned	10.10	3.83	8.19

The hyperparameters after tuning are:
DeepToF: LR 5e-4, decay 0.1 every 100 epochs, AE pre-training, full resolution.
CFN: LR 1e-4, decay 0.1 every 100 epochs, on patches.
End2End: 5e-4, decay: 0.1 every 100 epochs, full resolution.

Error distributions and statistical values are compared in Fig. 3.



Method	Median	σ
DeepToF	2.27	15.61
CFN	0.42	8.80
End2End	1.72	14.97
RADU	**-0.28**	**7.64**

Figure 3. Error distributions with median and standard deviation σ of the examined networks on our Cornell-Box dataset. An optimal distribution would be a single peak at 0. Both CFN and RADU have a similar distribution, where the median and standard deviation of CFN are slightly worse.

5.4. Kinect2 - FLAT Dataset

The FLAT dataset contains raw measurements simulating a Kinect2 camera for sinusoidal modulations at frequencies at 40MHz and ∼58.8MHz, and for a non-sinusoidal modulation at ∼30.3MHz. For each modulation three measurements were performed for phase offsets with a spacing of approximately $2\pi/3$.

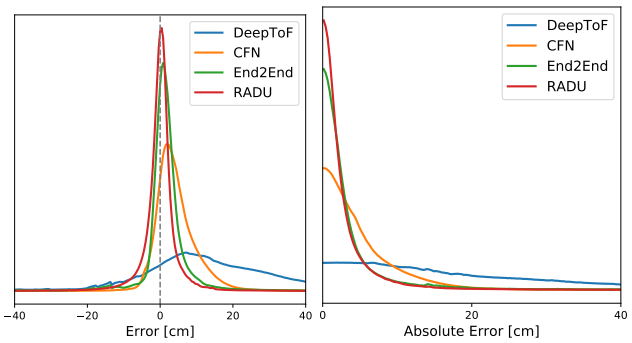
The resolution of the Kinect2 is 424×512 which results in 3392 points in the latent point clouds of our network. We train our network with an initial learning rate of 1e-3 and an exponential learning rate decay of 0.3 every 100 epochs, and a batch size of 2. The network converged after 600 epochs.

For DeepToF we use the low frequency ToF-depth as input. For CFN and our network we compute the input features using $f_1 = 30.3$MHz, $f_2 = 40$MHz, $f_3 = 58.8$MHz. For End2End we use the first two raw measurements of the two higher frequencies 40MHz, 58.8MHz as input.

We perform the same hyper parameter tuning as in the previous section and achieve the following MAE on the validation set:

	DeepToF	CFN	End2End
vanilla	11.52	4.30	6.21
tuned	8.66	3.57	5.90

The hyperparameters after tuning are:
DeepToF: LR: 1e-3, decay: 0.1 every 50 epochs, combined training, on patches.



Method	Median	σ
DeepToF	12.19	33.08
CFN	3.22	19.08
End2End	1.07	25.55
RADU	**-0.08**	**7.83**

Figure 4. Error distributions with median and standard deviation σ of the examined networks on the FLAT dataset. An optimal distribution would be a single peak at 0.

CFN: static LR: 1e-4, on patches.
End2End: LR: 5e-4, decay: 0.1 every 100 epochs, full resolution.

Error distributions and statistical values are compared in Fig. 4.

Code optimizations for FLAT dataset. The synthetic data used for training in the FLAT dataset contains a high ratio of background pixels, which we masked in the loss functions during training.

When training on patches we ensure to have non-empty images inside the batches.

As the background pixels are associated with depth 0, they are all projected to the same point $\vec{0} \in \mathbb{R}^3$. This introduces a heavy memory usage in 3D convolutions as these identical points are all considered as neighbors to each other.

To reduce the memory impact we apply a filter during training, which drops all masked points in the 3D projection $\mathcal{P}_{C\to G}$, which results in a varying number of points per image. After the 3D block of our network the points are projected back into 2D by ordering the points in a grid using a sparse data format with pixel ids inferred from the masking.

6. Ablations

We provide additional information about the ablation from the main paper and additional ablations on the hyperparameter α in our RADU convolutions, the coarse-fine-loss and on the U-DA strategy.

6.1. Ablation 1: Latent 3D Representation

For the 2D and 3D variants of our network architecture we use the same number of features in the 3D bottleneck block, namely $[128, 256, 128]$. For the 2.5D convolutions, which performs 3 convolutions for foreground, neighborhood and background, we change the feature dimensions to $[129, 258, 129] = [3\cdot 43, 3\cdot 86, 3\cdot 43]$, in order to make them divisible by 3.

The neighborhood radii are equal for all 3D convolutions at $[0.1m, 0.2m, 0.4m]$, for the 2D convolutions we use pixel neighborhoods of $[3, 5, 9]$, which is equal to doubling the pixel L0-distances $[1, 2, 4]$. For the 2.5D convolutions we use a fixed neighborhood size of 3 pixels, as larger kernels were too demanding in memory consumption.

We use the following additional hyperparameters for the 3D convolutions:
KPConv: 15 kernel points.
PointConv: 16 hidden units in the kernel MLP.
MCConv: 1 kernel MLP with 16 hidden units.

We train all variants with different hyperparameters and choose the run which achieved the best validation loss. The initial learning rate is chosen from {1e-2, 1e-3, 1e-4} and decayed with an exponential learning rate decay with rates {0.1, 0.3} every {50, 100, 150} epochs. The following settings achieved the best validation MAE on S3:

type	init LR	decay rate	decay steps
2D	1e-2	0.1	100
2.5D	1e-3	0.1	100
KPConv	1e-3	0.3	150
PointConv	1e-3	0.1	150
MCConv	1e-3	0.3	100

6.2. Ablation 2: Hyperparameter α

As discussed in the main paper the RADU layers receive direct gradients from the coarse loss, which can increase the risk of overfitting. We investigate the influence of the regularization hyperparameter α of the RADU layer, and train instances of our network, again using S1 and S3, for different values of α, including a dynamic value choice, by using the convolution radius $\alpha_l = r_l$, in our case 0.1 m, 0.2 m, and 0.4 m. Results are reported in Tab. 1. While the result indicate that a wrong choice of α yields the risk of overfitting on the training data, we found that $\alpha = 0.1\,\text{m}$ leads to good performance on the three datasets of the main paper.

6.3. Ablation 3: Coarse-Fine-Loss

To investigate the influence of the coarse-fine-loss, we train an instance of our network using only the L1-Loss $\|d_{gt} - \hat{d}_{out}\|_1$, again using S1 for training and S3 for validation.

α [m]	Training (S1) MAE [cm]	Validation (S3) MAE [cm]
0.0	8.38	2.51
0.1	7.87	2.28
0.2	7.51	2.81
1.0	6.93	3.42
r	7.48	2.45

Table 1. Influence of the hyperparameter α in the RADU convolutions. We report MAE on training (synthetic) and validation (real) data. The case $\alpha = r$ uses the receptive field r as scale. The value $\alpha = 0$ corresponds to a standard MCConv layer.

In this setting the number of epochs until the network converged during training increased by 100 epochs, while the performance on the validation set S3 decreased to a MAE of 2.63cm, +0.35 compared to the proposed coarse-fine-loss.

6.4. Ablation 4: U-DA Strategy

We repeat the U-DA finetuning in the real world data experiment from the main paper using an adversarial setup as proposed in other works [1, 13]. In this setting no pseudo-labels are created, instead a discriminator network is employed to distinguish predictions on real and synthetic data. We implement the discriminator network and U-DA algorithm as described by Agresti *et al*. [1].

Using this setup the performance on S5 decreases to a MAE of 2.13cm (+0.5). As with our cyclic self-training approach we repeat the training 10 times and measure a standard deviation of the MAE at 0.057cm on S5, which is notably higher (+0.036) then when fine-tuning with our proposed cyclic self-training approach using pseudo labels.

7. Implementation

All network implementations were done in `TensorFlow 2.3.0-gpu` and `Python 3.6`. The dataset generation was done using `Python 3.6` and the transient renderer of Jarabo *et al*. [8] in version `26 February 2019 - Release v1.2`.

8. Qualitative Results

8.1. Cornell-Box Dataset

We show predictions for one view point per scene in Fig. 7, 8, 9 and 10.

We further show a larger version of Figure 8 of the main paper in Fig. 6, which shows the iterative denoising on the latent point clouds.

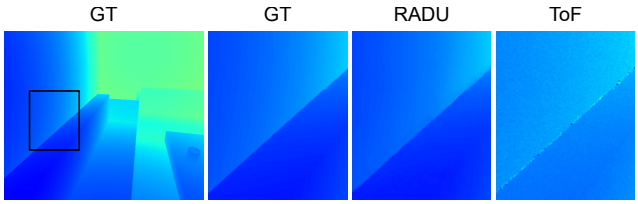


Figure 5. Example of flying pixel correction. Left shows the ground truth (GT) depth image, the right images show a zoomed in detail. The ToF image exhibits flying pixels at the object boundary, which do not occur in the prediction of our RADU network.

8.2. FLAT Dataset

We show predictions for a subset of the images in the dataset in Fig. 11 and 12.

Lastly, we show an example of the correction of the mixed-pixel / flying pixel effect in 5.

References

[1] Gianluca Agresti, Henrik Schaefer, Piergiorgio Sartor, and Pietro Zanuttigh. Unsupervised domain adaptation for ToF data denoising with adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5584–5593, 2019. 3, 4, 6

[2] European Machine Vision Association et al. Standard for characterization of image sensors and cameras. *EMVA Standard*, 1288, 2010. 2

[3] Mario Frank, Matthias Plaue, Holger Rapp, Ullrich Köthe, Bernd Jähne, and Fred A Hamprecht. Theoretical and experimental error analysis of continuous-wave time-of-flight range cameras. *Optical Engineering*, 48(1):013602, 2009. 1

[4] Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *ACM SIGGRAPH computer graphics*, 18(3):213–222, 1984. 2

[5] Qi Guo, Iuri Frosio, Orazio Gallo, Todd Zickler, and Jan Kautz. Tackling 3D ToF artifacts through learning and the FLAT dataset. In *The European Conference on Computer Vision (ECCV)*, September 2018. 2

[6] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Patrice Horaud. *Time-of-flight cameras: principles, methods and applications*. Springer Science & Business Media, 2012. 1

[7] Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics (TOG)*, 37(6):1–12, 2018. 1

[8] Adrian Jarabo, Julio Marco, Adolfo Muñoz, Raul Buisan, Wojciech Jarosz, and Diego Gutierrez. A framework for transient rendering. *ACM Transactions on Graphics (SIGGRAPH Asia 2014)*, 33(6), 2014. 2, 6

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3

[10] Julio Marco, Quercus Hernandez, Adolfo Munoz, Yue Dong, Adrian Jarabo, Min H Kim, Xin Tong, and Diego Gutierrez.

DeepToF: off-the-shelf real-time correction of multipath interference in time-of-flight imaging. *ACM Transactions on Graphics (ToG)*, 36(6):1–12, 2017. 2, 4

[11] Markus Miller, Hongwang Xia, Mina Beshara, Susanne Menzel, Karl Joachim Ebeling, and Rainer Michalzik. Large-area transmission modulators for 3d time-of-flight imaging. In *Unconventional Optical Imaging II*, volume 11351, page 113511F. International Society for Optics and Photonics, 2020. 2

[12] Mary A Pagnutti, Robert E Ryan, Maxwell J Gold, Ryan Harlan, Edward Leggett, James F Pagnutti, et al. Laying the foundation to use raspberry pi 3 v2 camera module imagery for scientific and engineering purposes. *Journal of Electronic Imaging*, 26(1):013014, 2017. 2

[13] Shuochen Su, Felix Heide, Gordon Wetzstein, and Wolfgang Heidrich. Deep end-to-end time-of-flight imaging. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6383–6392, 2018. 4, 6

[14] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016. 2

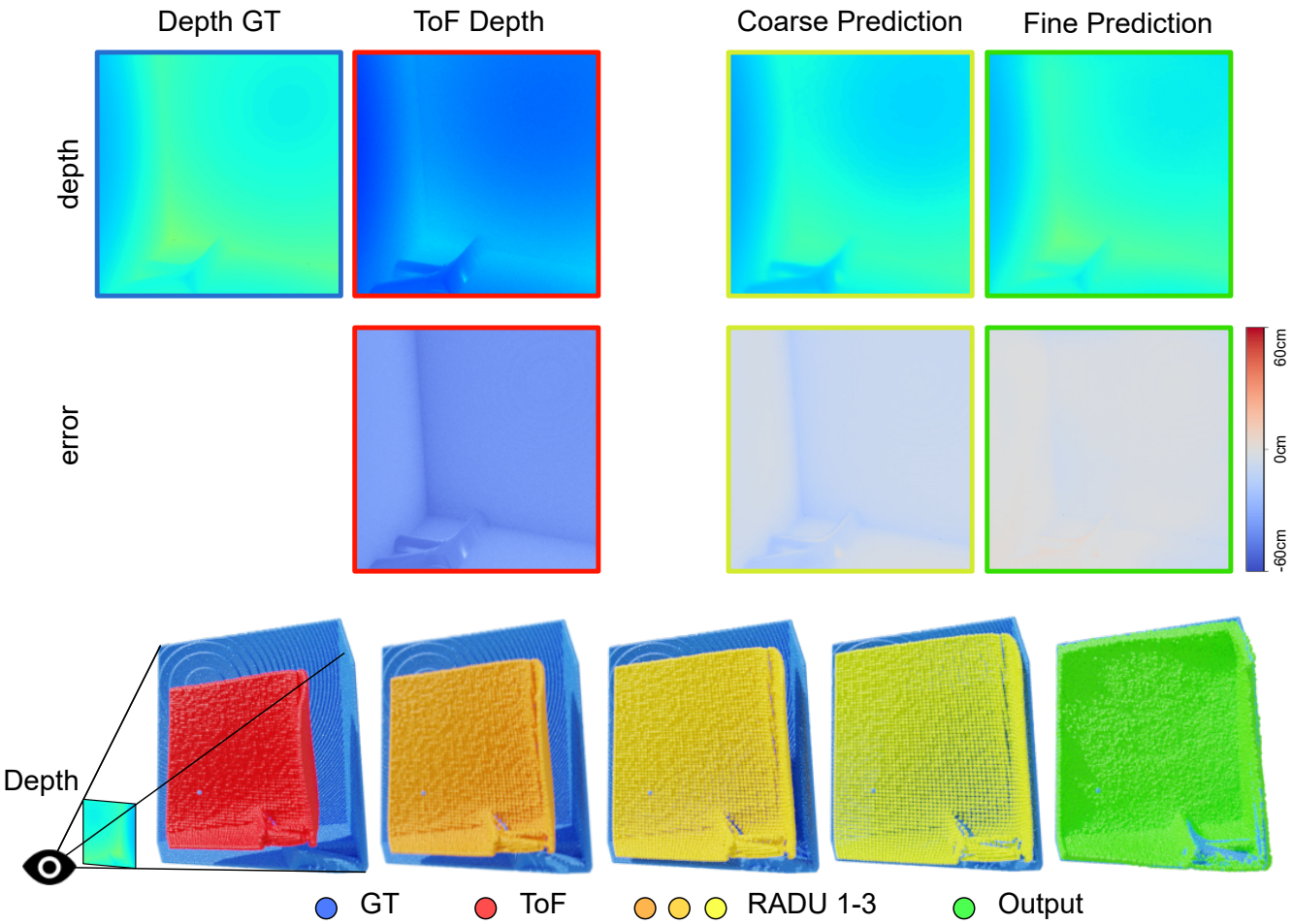


Figure 6. Larger version of Figure 8 in the main paper. The top row shows depth and error maps, the bottom row shows the point clouds in 3D space. The initial ToF depth reconstruction (red) is far from the ground truth depth (blue). After each RADU convolution the latent point clouds (orange to yellow) move closer to the correct depth. The final latent point cloud (yellow) already yields a good coarse reconstruction of the scene, which is further refined in the 2D block of the network (green).

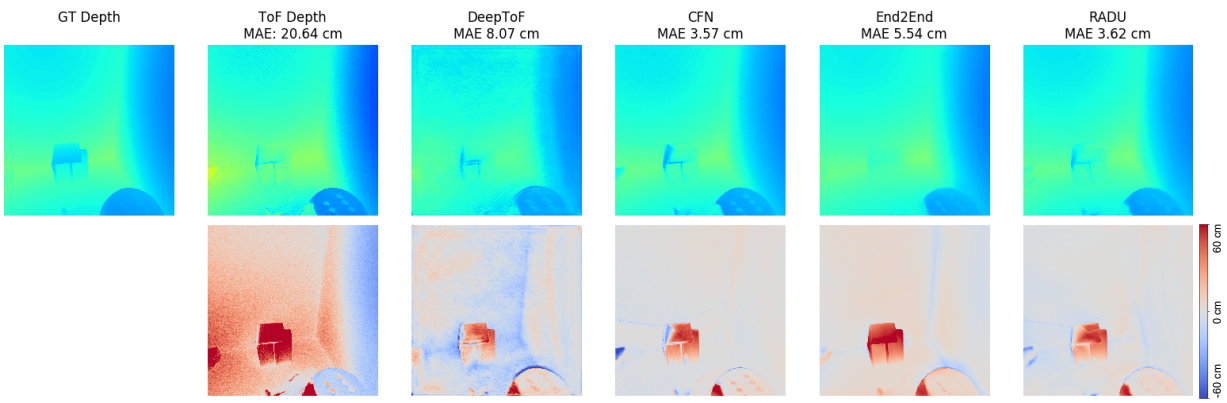


Figure 7. Results on the Cornell-Box Dataset. First row shows depths, second row shows error maps.

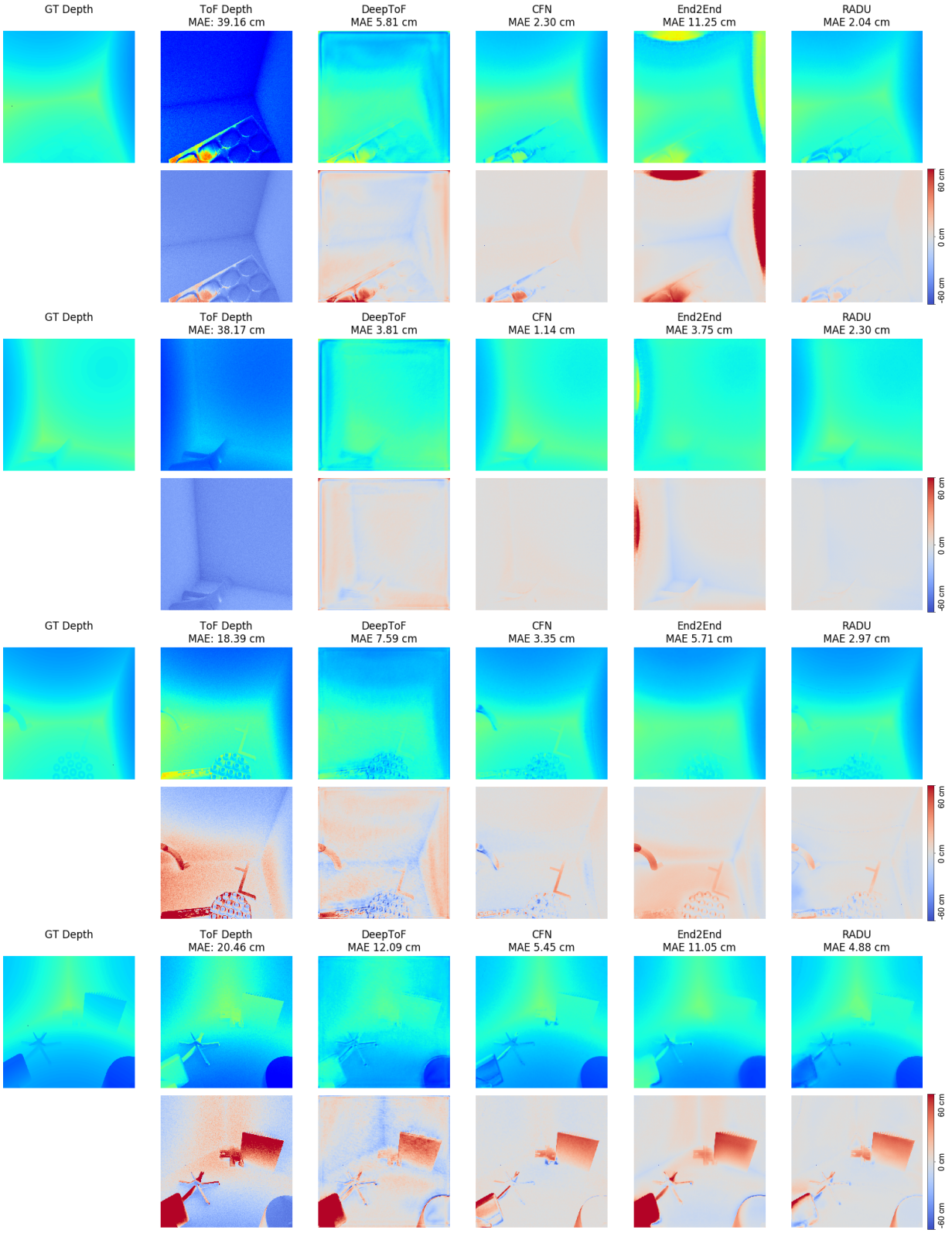


Figure 8. Results on the Cornell-Box Dataset. First rows show depths, second rows show error maps.

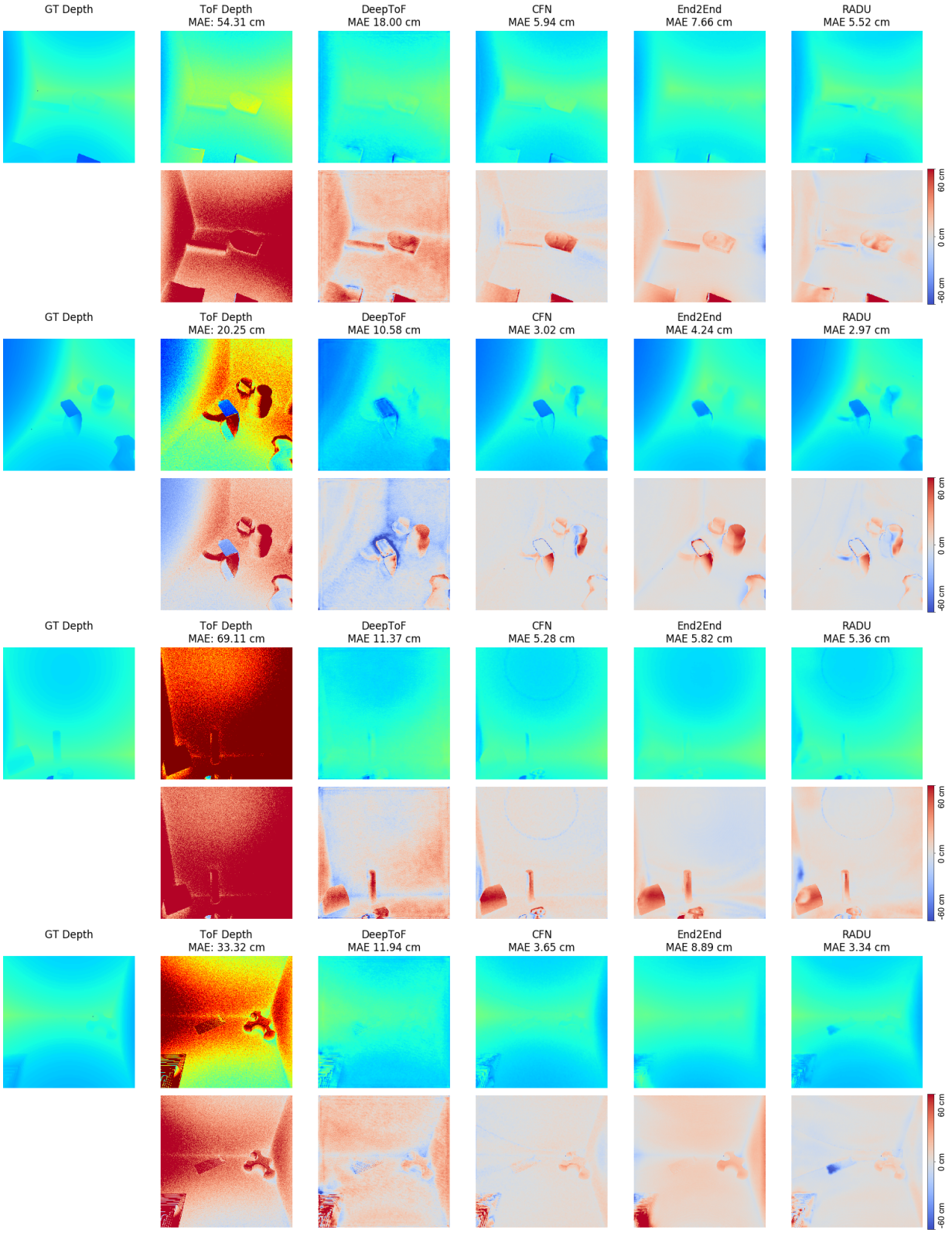


Figure 9. Results on the Cornell-Box Dataset. First rows show depths, second rows show error maps.

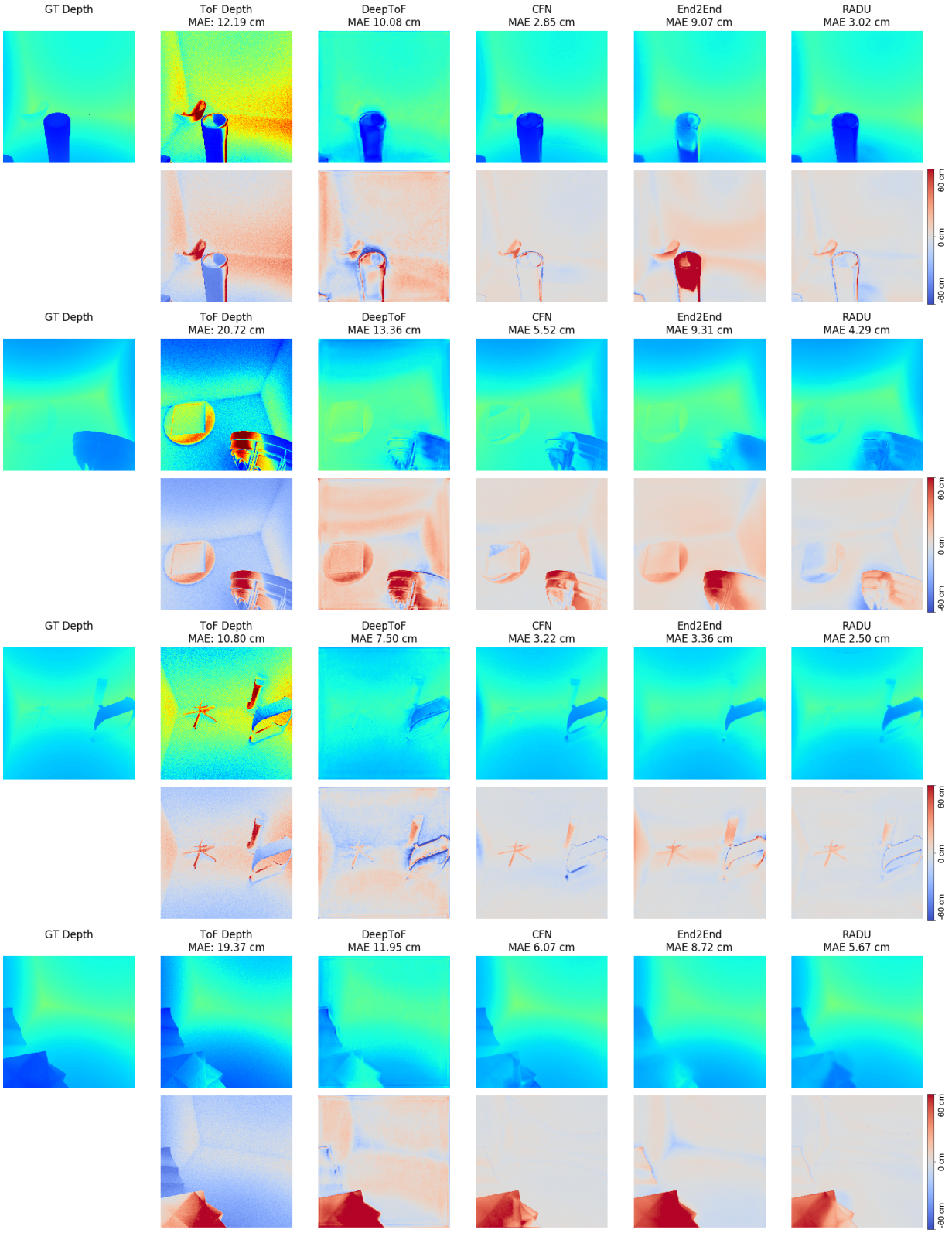


Figure 10. Results on the Cornell-Box Dataset. First rows show depths, second rows show error maps.

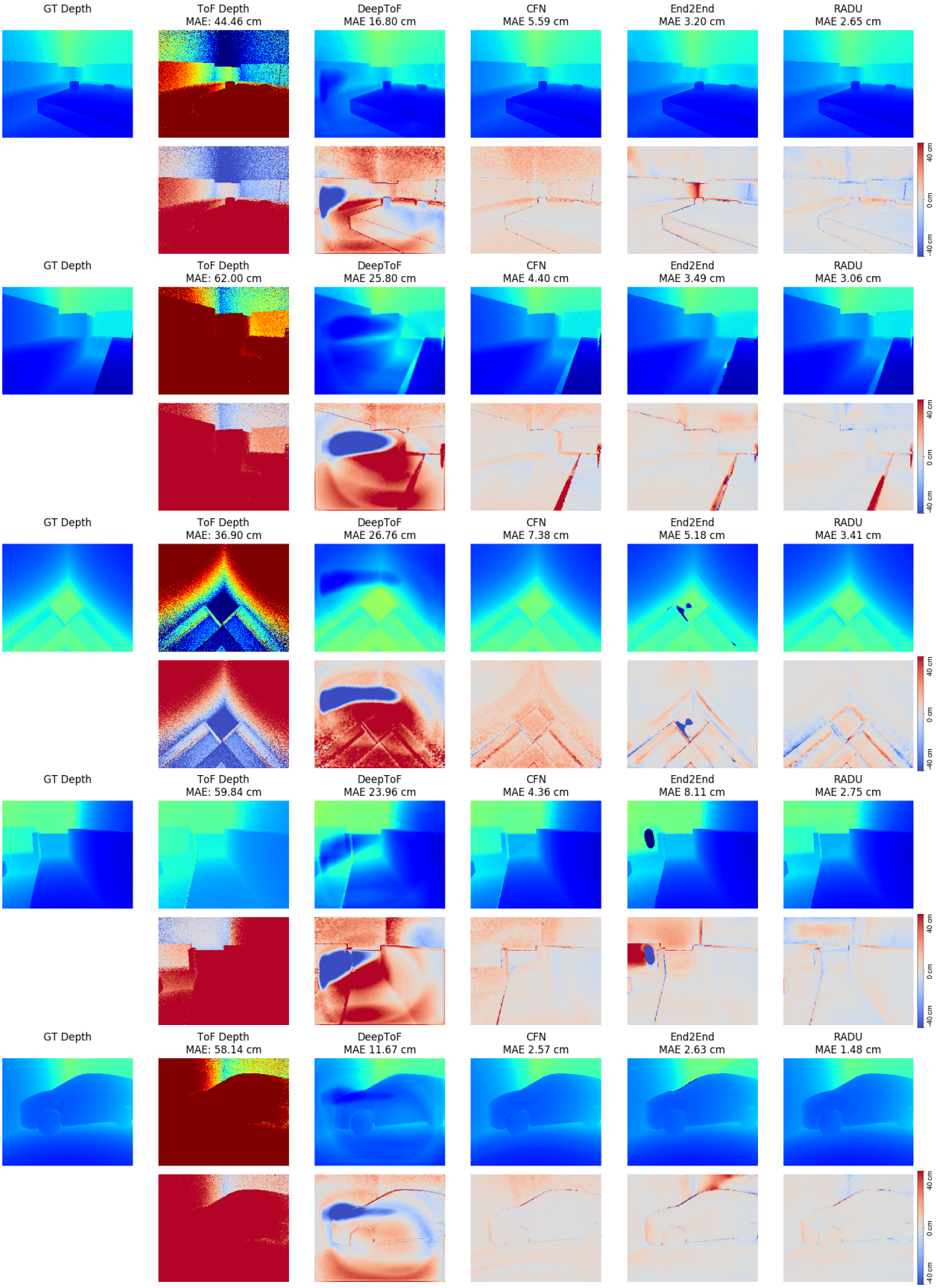


Figure 11. Results on the FLAT Dataset. First rows show depths, second rows show error maps.

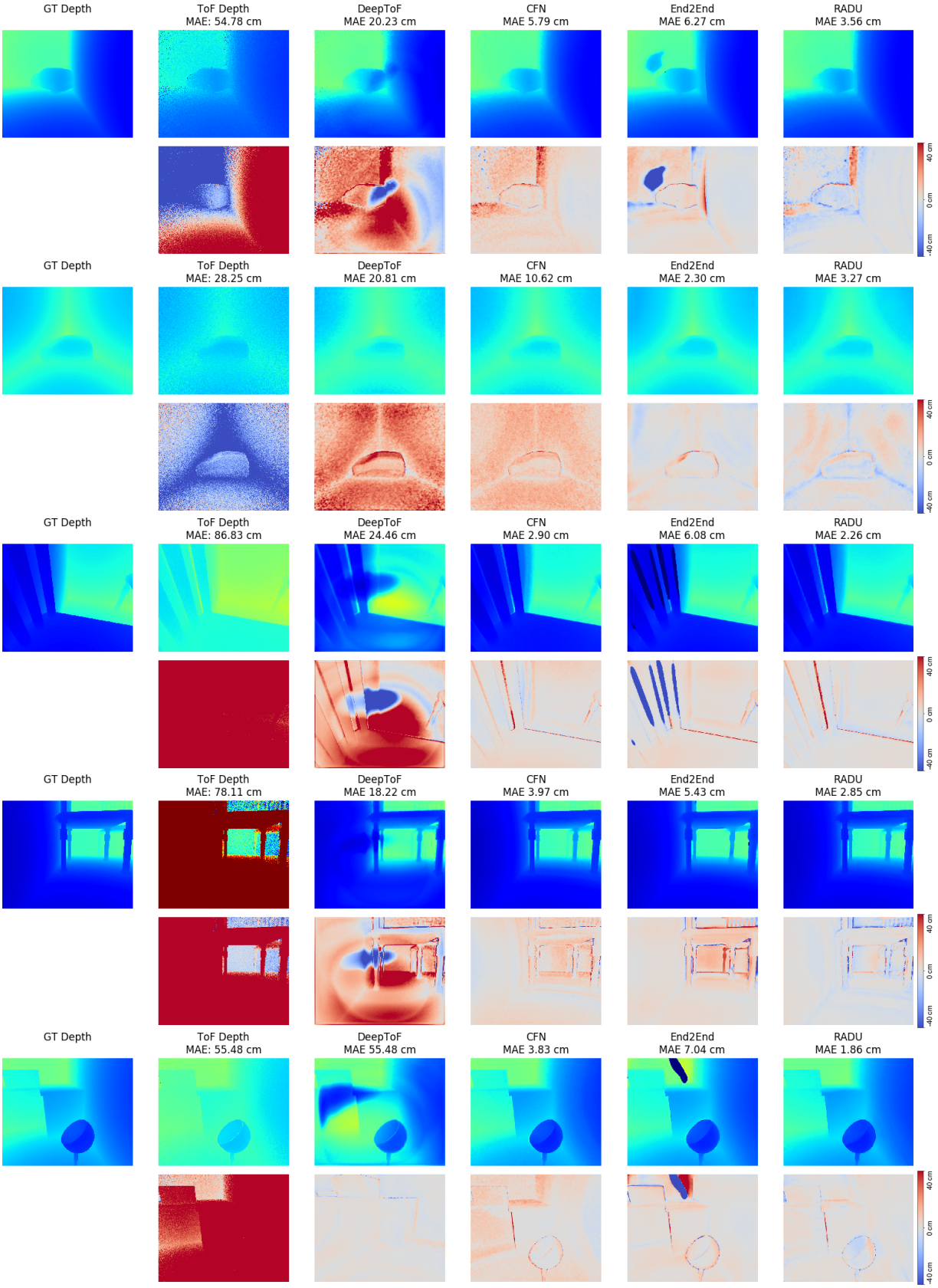


Figure 12. Results on the FLAT Dataset. First rows show depths, second rows show error maps.

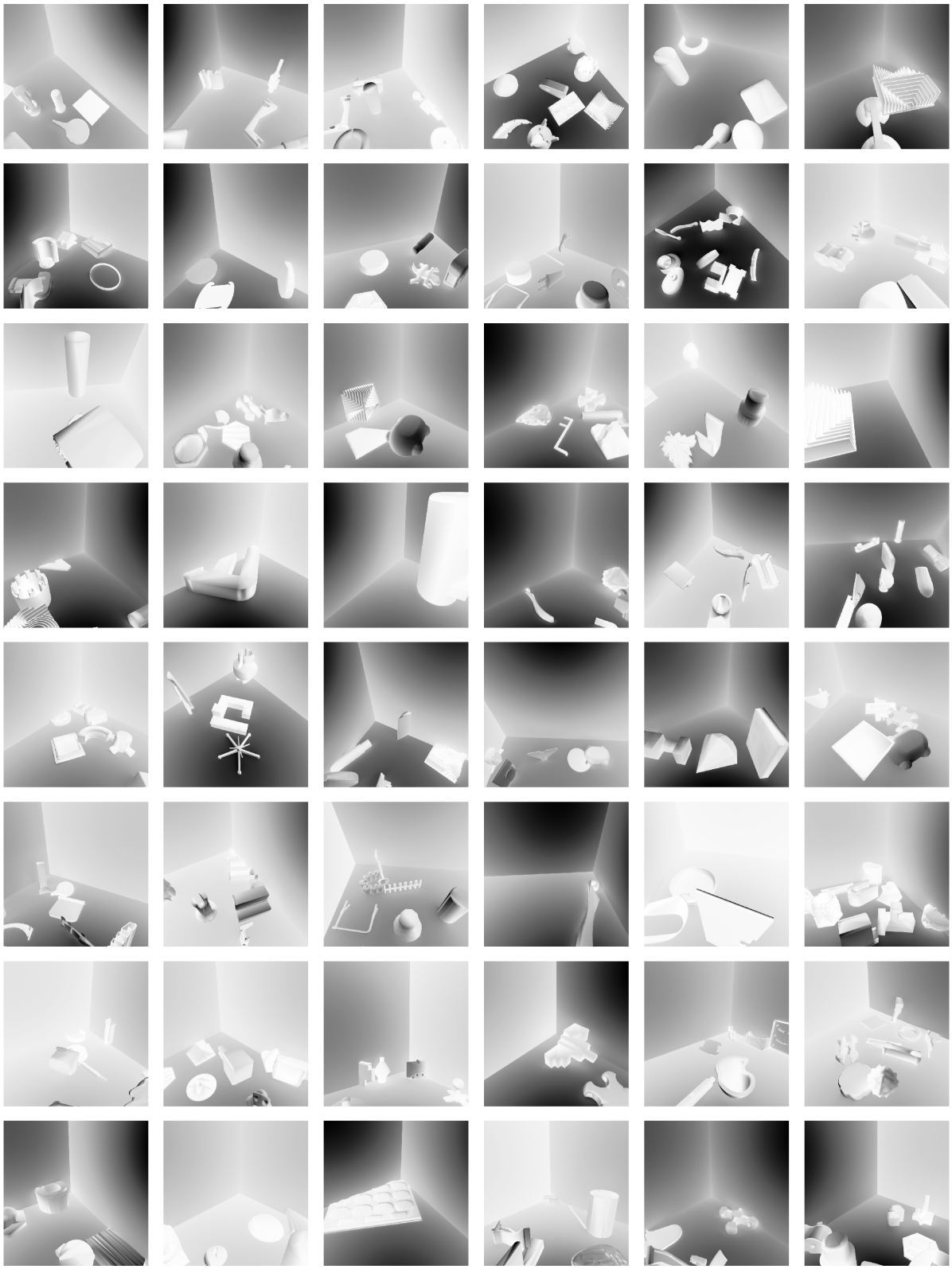


Figure 13. Example images showing one of the 50 view points with one of the three material configurations for each of the scenes in our Cornell-Box dataset. We show the intensity I at a frequency of 20 MHz as given by Eq. (7).

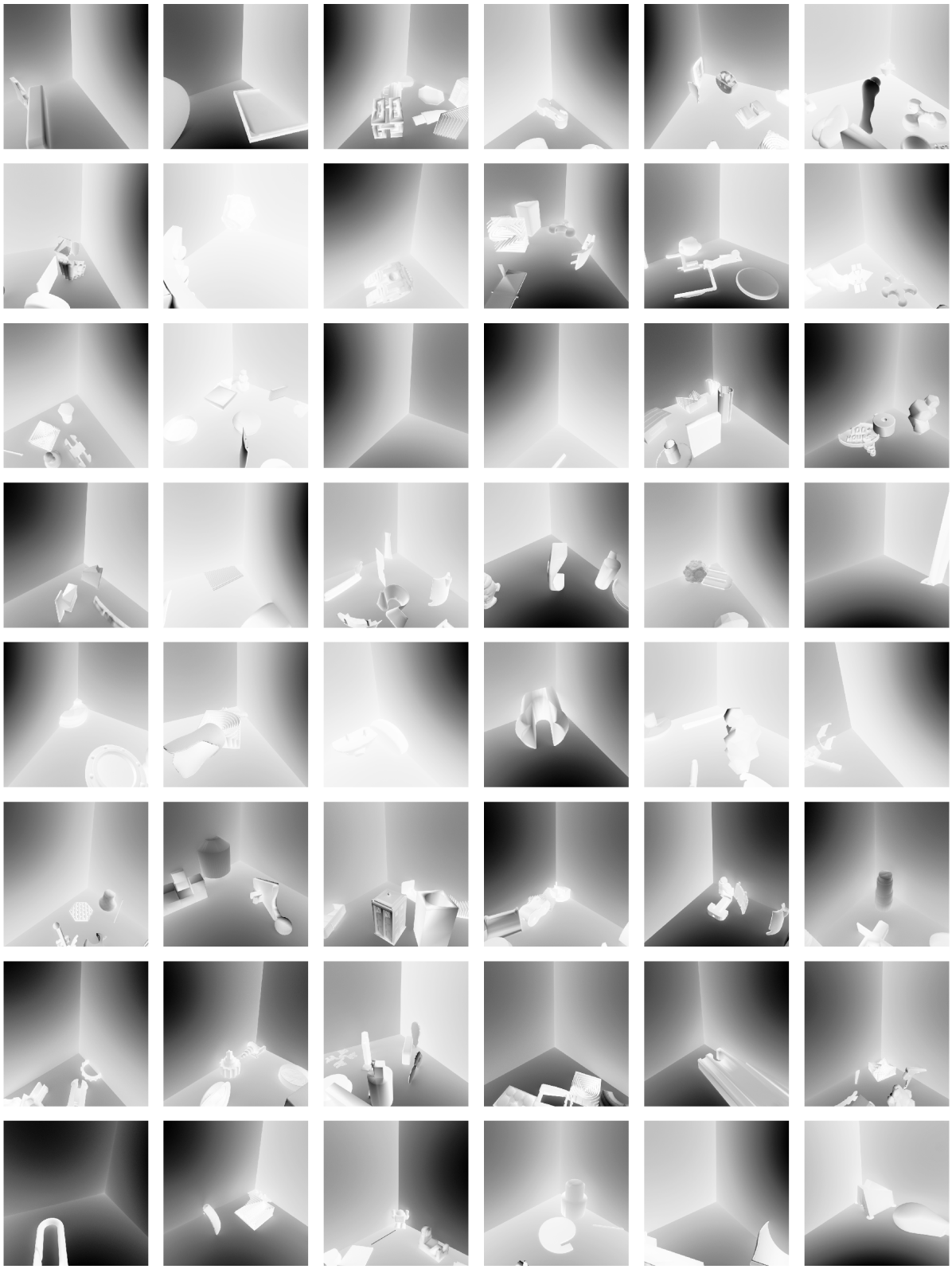


Figure 14. Example images showing one of the 50 view points with one of the three material configurations for each of the scenes in our Cornell-Box dataset. We show the intensity I at a frequency of 20 MHz as given by Eq. (7).

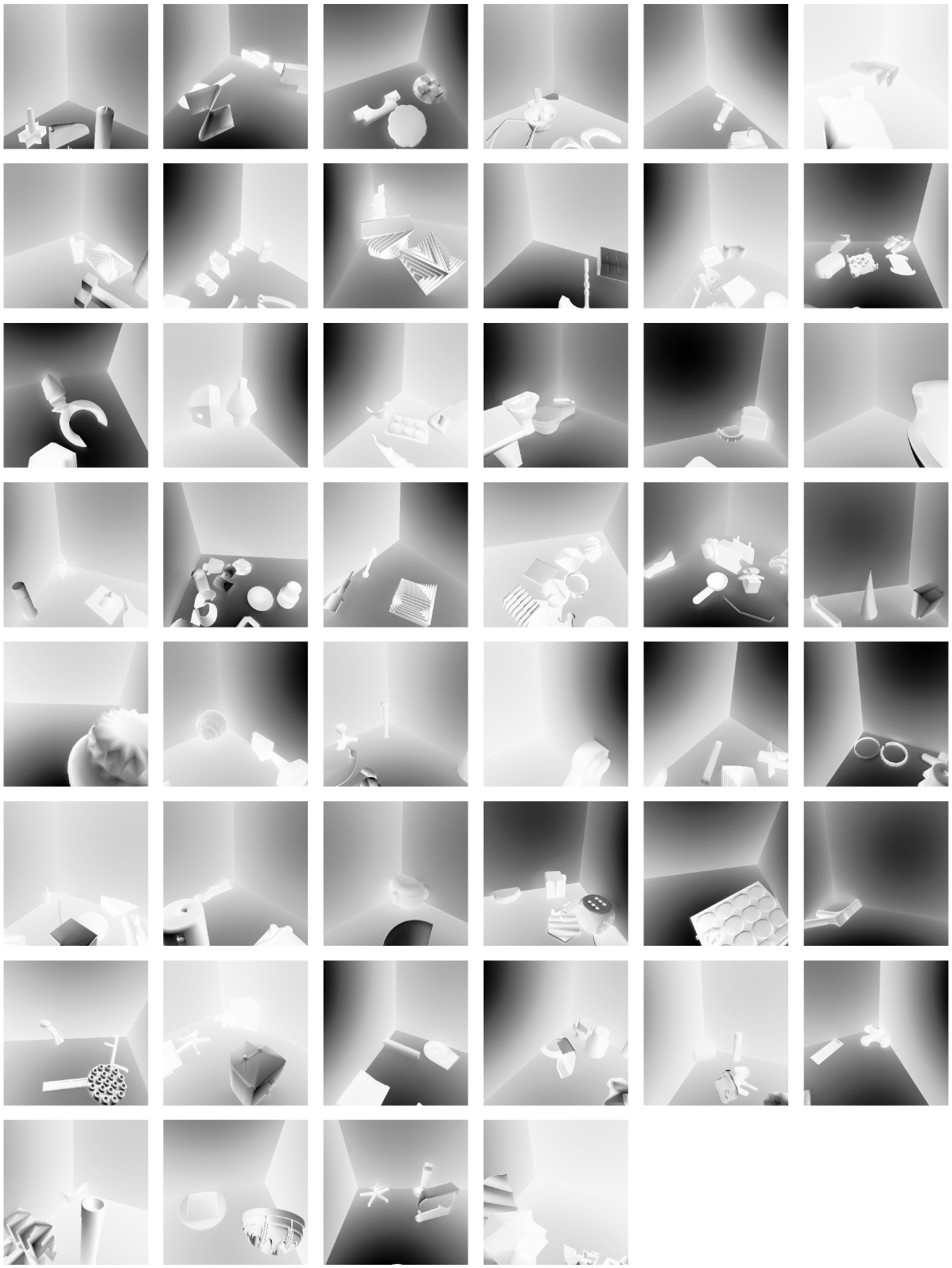


Figure 15. Example images showing one of the 50 view points with one of the three material configurations for each of the scenes in our Cornell-Box dataset. We show the intensity I at a frequency of 20 MHz as given by Eq. (7).