Deep Depth Completion of a Single RGB-D Image (Supplementary Material)

Yinda Zhang
Princeton University

Thomas Funkhouser
Princeton University

This document contains further implementation details and results of ablation studies, cross-dataset experiments, and comparisons to other inpainting methods that would not fit in the main paper.

1. Further Implementation Details

This section provides extra implementation details for our methods. All data and code will be released upon the acceptance to ensure reproducibility.

1.1. Mesh reconstruction and rendering

For every scene in the Matterport3D dataset, meshes were reconstructed and rendered to provide "completed depth images" using the following process. First, each house was manually partitioned into regions roughly corresponding to rooms using an interactive floorplan drawing interface. Second, a dense point cloud was extracted containing RGB-D points (pixels) within each region, excluding pixels whose depth is beyond 4 meters from the camera (to avoid noise in the reconstructed mesh). Third, a mesh was reconstructed from the points of each region using Screened Poisson Surface Reconstruction [7] with octree depth 11. The meshes for all regions were then merged to form the final reconstructed mesh M for each scene. "Completed depth images" were then created for each of the original RGB-D camera views by rendering M from that view using OpenGL and reading back the depth buffer.

Figure 1 shows images of a mesh produced with this process. The top row shows exterior views covering the entire house (vertex colors on the left, flat shading on the right). The bottom row shows a close-up image of the mesh from an interior view. Though the mesh is not perfect, it has 12.2M triangles reproducing most surface details. Please note that the mesh is complete where holes typically occur in RGB-D images (windows, shiny table tops, thin structures of chairs, glossy surfaces of cabinet, etc.). Please also note the high level of detail for surfaces distant to the camera (e.g., furniture in the next room visible through the doorway).

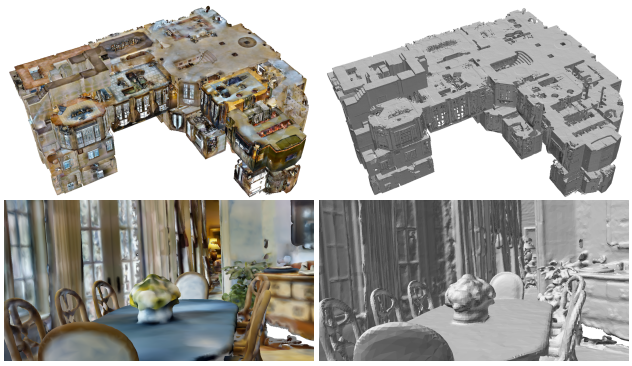

Figure 1. **Reconstructed mesh for one scene.** The mesh used to render completed depth images is shown from an outside view (top) and inside view (bottom), rendered with vertex colors (left) and flat shading (right).

1.2. Network architecture

All the networks used for this project are derived from the surface normal estimation model proposed in Zhang et.al [13] with the following modifications.

Input Depending on what is the input, the network takes data with different channels at the first convolution layer.

- Color. The color is a 3-channel tensor with R,G,B for each. The intensity values are normalized to [-0.5 0.5]. We use a bi-linear interpolation to resize color image if necessary.

- Depth. The absolute values of depth in meter are used as input. The pixels with no depth signal from sensor are assigned a value of zero. To resolve the ambiguity between "missing" and "0 meter", a binary mask indicating the pixels that have depth from sensor is added as an additional channel as suggested in Zhang et.al [12]. Overall, the depth input contains 2 channels (absolute depth and binary valid mask) in total. To prevent inaccurate smoothing, we use the nearest neighbor search to resize depth image.

- Color+Depth. The input in this case is the concatenation of the color and depth as introduced above. This results in a 5-channel tensor as the input.

Output The network for absolute depth, surface normal, and depth derivative outputs results with 1, 3, and 8 channels respectively. The occlusion boundary detection network generates 3 channel outputs representing the probability of each pixel belonging to "no edge", "depth crease", and "occlusion boundary".

Loss Depth, surface normal, and derivative are predicted as regression tasks. The SmoothL1 loss[1] is used for training depth and derivative, and the cosine embedding loss[2] is used for training surface normal. The occlusion boundary detection is formulated into a classification task, and cross entropy loss[3] is used. The last two batch normalization layers are removed because this results in better performance in practice.

1.3. Training schema

The neural network training and testing are implemented in Torch. For all the training tasks, RMSprop optimization algorithm is used. The momentum is set to 0.9, and the batch size is 1. The learning rate is set to 0.001 initially and reduce to half every 100K iterations. All the models converge within 300K iterations.

2. Further Experimental Results

This section provides extra experimental results, including ablation studies, cross-dataset experiments, and comparisons to other depth completion methods.

2.1. Ablation Studies

Section 4.1 of the paper provides results of ablation studies aimed at investigating how different test inputs, training data, loss functions, depth representations, and optimization methods affect our depth prediction results. This section provides further results of that type.

More qualitative results about surface normal estimation model trained from different setting are shown in Figure 2. Comparatively, training the surface normal estimation model with our setting (i.e. using only color image as input, all available pixels with rendered depth as supervision, the 4-th column in the figure) achieves the best quality of prediction, and hence benefits the global optimization for depth completion.

[1]https://github.com/torch/nn/blob/master/doc/criterion.md#nn.SmoothL1Criterion

[2]https://github.com/torch/nn/blob/master/doc/criterion.md#nn.CosineEmbeddingCriterion

[3]https://github.com/torch/nn/blob/master/doc/criterion.md#nn.CrossEntropyCriterion

Input	Rep	Rel↓	RMSE↓	1.05↑	1.10↑	1.25↑	1.25^2↑	1.25^3↑
C	D	0.408	0.500	6.49	12.80	30.01	54.44	72.88
C	1/D	0.412	0.492	6.86	12.88	28.99	54.51	73.13
D	D	0.167	0.241	16.43	31.13	57.62	75.63	**84.01**
D	1/D	0.199	0.255	14.06	27.32	53.70	74.19	83.85
Ours		**0.089**	**0.116**	**40.63**	**51.21**	**65.35**	**76.74**	82.98

Table 1. **Comparison of different depth representations.** Predicting either depth (D) or disparity (1/D) provides worse results than predicting surface normals and solving for depth (Ours) for either color or depth inputs.

What kind of ground truth is better? This test studies what normals should be used as supervision for the loss when training the surface prediction network. We experimented with normals computed from raw depth images and with normals computed from the rendered mesh. The result in the top two rows of Table 2 (Comparison:Target) shows that the model trained on rendered depth performs better the the one from raw depth. The improvement seems to come partly from having training pixels for unobserved regions and partly from more accurate depths (less noise).

What loss should be used to train the network? This test studies which pixels should be included in the loss when training the surface prediction network. We experimented with using only the unobserved pixels, using only the observed pixels, and both as supervision. The three models were trained separately in the training split of our new dataset and then evaluated versus the rendered normals in the test set. The quantitative results in the last three rows of Table 2 (Comparison:Pixel) show that models trained with supervision from both observed and unobserved pixels (bottom row) works slightly better than the one trained with only the observed pixels or only the unobserved pixels. This shows that the unobserved pixels indeed provide additional information.

What kind of depth representation is best? Several depth representations were considered in the paper (normals, derivatives, depths, etc.). This section provides further results regarding direct prediction of depth and disparity (i.e. one over depth) to augment/fix results in Table 2 of the paper.

Actually, the top row of Table 2 of the paper (where the Rep in column 2 is 'D') is mischaracterized as direct prediction of depth from color – it is actually direct prediction of complete depth from input depth. That was a mistake. Sorry for the confusion. The correct result is in the top line of Table 1 of this document (Input=C, Rep=D). The result is quite similar and does not change any conclusions: predicting surface normals and then solving for depth is better than predicting depth directly (Rel = 0.089 vs. 0.408).

We also consider prediction of disparity rather than depth, as suggested in Chakrabarti et.al and other papers

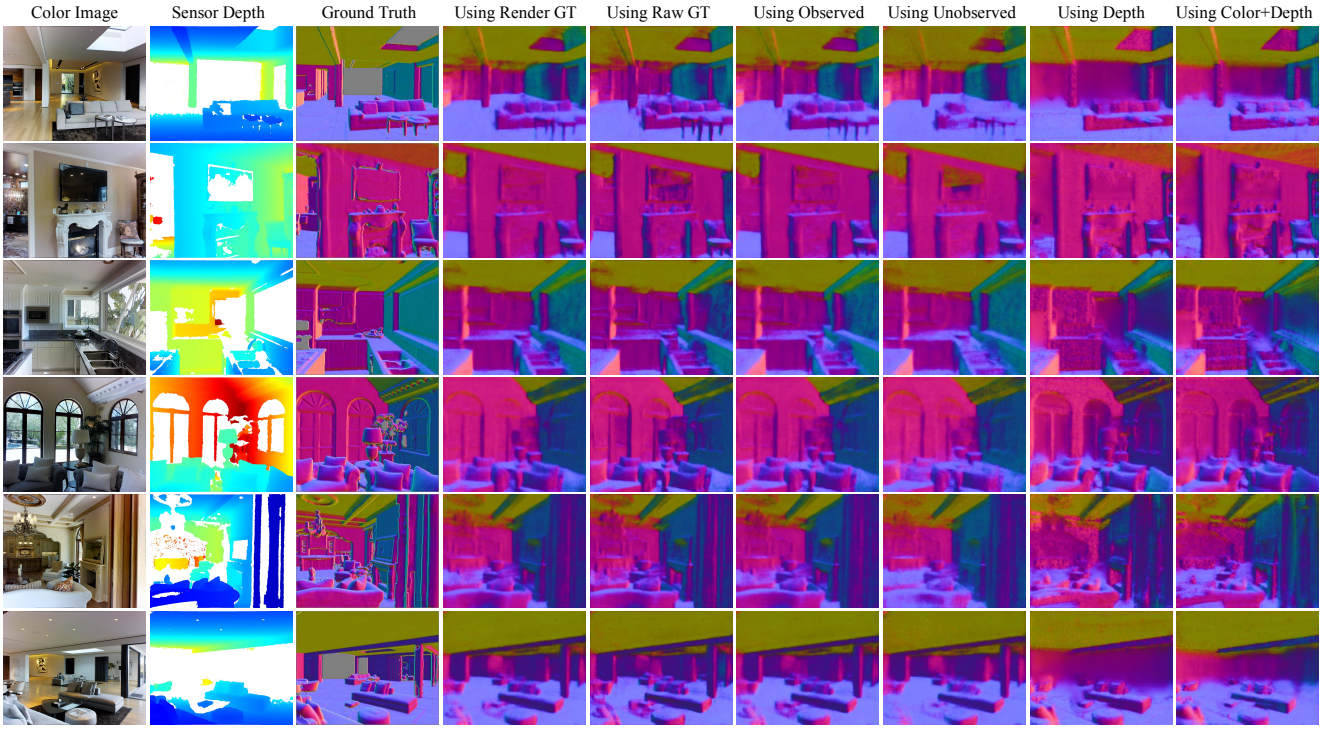

Figure 2. **Comparison of normal estimation with different training settings.** The 4-th column shows the output of the model trained using only color as input and the rendered depth from all pixels as supervision, which is the setting we chose for our system. Comparatively, it generates better surface normal than other alternative training settings.

Comparison	Setting			Depth Completion							Surface Normal Estimation				
	Input	Target	Pixel	Rel↓	RMSE↓	1.05↑	1.10↑	1.25↑	1.25^2 ↑	1.25^3 ↑	Mean↓	Median↓	11.25↑	22.5↑	30↑
Target	Color	*Raw*	Observed	0.094	0.123	39.84	50.40	64.68	76.38	82.80	32.87	18.70	34.2	55.7	64.3
	Color	*Render*	Both	0.089	0.116	40.63	51.21	65.35	76.64	82.98	31.13	17.28	37.7	58.3	67.1
Pixel	Color	Render	*Observed*	0.091	0.121	40.31	50.88	64.92	76.50	82.91	32.16	18.44	34.7	56.4	65.5
	Color	Render	*Unobserved*	0.090	0.119	40.71	51.22	65.21	76.59	83.04	31.52	17.70	35.4	57.7	66.6
	Color	Render	*Both*	0.089	0.116	40.63	51.21	65.35	76.64	82.98	31.13	17.28	37.7	58.3	67.1

Table 2. Ablation studies. Evaluations of estimated surface normals and solved depths using different training inputs and losses.

[3]. We train models to estimate disparity directly from color and raw depth respectively. The results can be seen in Table 1. We find that estimating disparity results in performance that is not better than estimating depth when given either color or depth as input for our depth completion application.

2.2. Cross-Dataset Experiments

This test investigates whether it is possible to train our method on one dataset and use it effectively for another.

Matterport3D and ScanNet We first conduct experiments between Matterport3D and ScanNet datasets. Both have 3D surface reconstructions for large sets of environments (∼1000 rooms each) and thus provide suitable training data for training and test our method with rendered meshes. We train a surface normal estimation model separately on each dataset, and then use it without fine tuning to perform depth completion for the test set of the other. The quantitative results are shown in Table 3. As expected, the models work best on the test dataset matching the source of the training data. Actually, the model trained from Matterport3D has a better generalization capability compared to the model trained from ScanNet, which is presumably because the Matterport3D dataset has a more diverse range of camera viewpoints. However, interestingly, both models work still reasonably well when run on the other dataset, even though they were not fine-tuned at all. We conjecture this is because our surface normal prediction model is trained only on color inputs, which are relatively similar between the two datasets. Alternative methods using depth as input would probably not generalize as well due to the significant differences between the depth images of the two datasets.

Intel RealSense Depth Sensor The depth map from Intel RealSense has better quality in short range but contains more missing area compared to Structure Sensor [2] and

Train	Test	Rel	RMSE	1.05	1.10	1.25	1.25^2	1.25^3
Matterport3D	Matterport3D	0.089	0.116	40.63	51.21	65.35	76.74	82.98
ScanNet	Matterport3D	0.098	0.128	37.96	49.79	64.01	76.04	82.64
Matterport3D	Scannet	0.042	0.065	52.91	65.83	81.20	90.99	94.94
ScanNet	ScanNet	0.041	0.064	53.33	66.02	81.14	90.92	94.92

Table 3. **Cross-dataset performance.** We trained surface normal estimation models on each dataset, Matterport3D and ScanNet, respectively and test on both. Models work the best on the dataset where it is trained from. Model trained from Matterport3D shows better generalization capability than the one from ScanNet.

Kinect [1]. The depth signal can be totally lost or extremely sparse for distant area and surface with special materials, e.g. shinny, dark. We train a surface normal estimation model from ScanNet dataset [4] and directly evaluate on the RGBD images captured by Intel RealSense from SUN-RGBD dataset [11] without any finetuning. The results are shown in Figure 3. From left to right, we show the input color image, input depth image, completed depth image using our method, the point cloud visualization of the input and completed depth map, and the surface normal converted from the completed depth. As can be seen, the depth from RealSense contains more missing area than Matterport3D and ScanNet, yet our model still generates decent results. This again shows that our method can effectively run on RGBD images captured from various of depth sensors with significantly different depth patterns.

2.3. Comparisons to Depth Inpainting Methods

Section 4.2 of the paper provides comparisons to alternative methods for depth inpainting. This section provides further results of that type. In this additional study, we compare with the following methods:

- **DCT [6]:** fill in missing values by solving the penalized least squares of a linear system using discrete cosine transform using the code from Matlab Central [4].
- **FCN [8]:** train an FCN with symmetric shortcut connection to take raw depth as input and generate completed depth as the output using the code from Zhang et.al [13].
- **CE [9]:** train the context encoder of Pathak et.al to inpaint depth images using the code from Github [5].

The results of DCT [6] are similar to other inpainting comparisons provided in the paper. They mostly interpolate holes. The results of FCN and CE show that methods designed for inpainting color are not very effective at inpainting depth. As already described in the paper, methods that learn depth from depth using an FCN can be lazy and only learn to reproduce and interpolate provided depth. However, the problems are more subtle than that, as depth data has many characteristics different from color. For starters, the context encoder has a more shallow generator and lower resolution than our network, and thus generates blurrier depth images than ours. More significantly, the fact that ground-truth depth data can have missing values complicates the training of the discriminator network in the context encoder (CE) – in a naive implementation, the generator would be trained to predict missing values in order to fool the discriminator. We tried multiple approaches to circumvent this problem, including propagating gradients on only unobserved pixels, filling a mean depth value in the missing area. We find that none of them work as well as our method.

More results of our method and comparison to other inpainting methods can be found in Figure 5,6,7 in the end of this paper. Each two rows shows an example, where the 2nd row shows the completed depth of different methods, and 1st row shows their corresponding surface normal for purpose of highlighting details and 3D geometry. For each example, we show the input, ground truth, our result, followed by the results of FCN [8], joint bilateral filter [10], discrete cosine transform [6], optimization with only smoothness, and PDE [5]. As can be seen, our method generates better large scale planar geometry and sharper object boundary.

Method	Rel↓	RMSE↓	1.05↑	1.10↑	1.25↑	1.25^2↑	1.25^3↑
Garcia et.al [6]	0.115	0.144	36.78	47.13	61.48	74.89	81.67
FCN [13]	0.167	0.241	16.43	31.13	57.62	75.63	84.01
Ours	**0.089**	**0.116**	**40.63**	**51.21**	**65.35**	**76.74**	**82.98**

Table 4. **Comparison to baseline inpainting methods.** Our method significantly outperforms baseline inpainting methods.

We also convert the completed depth maps into 3D point clouds for visualization and comparison, which are shown in Figure 4. The camera intrinsics provided in Matterport3D dataset is used to project each pixel on the depth map into a 3D point, and the color intensity are copied from the color image. Each row shows one example, with the color image and point clouds converted from ground truth, input depth (i.e. the raw depth from sensor that contains a lot of missing area), and results of our method, FCN [8], joint bilateral filter [10], and smooth inpainting. Compared to other methods, our method maintains better 3D geometry and less bleeding on the boundary.

[4] https://www.mathworks.com/matlabcentral/fileexchange/27994-inpaint-over-missing-data-in-1-d–2-d–3-d–nd-arrays

[5] https://github.com/pathak22/context-encoder

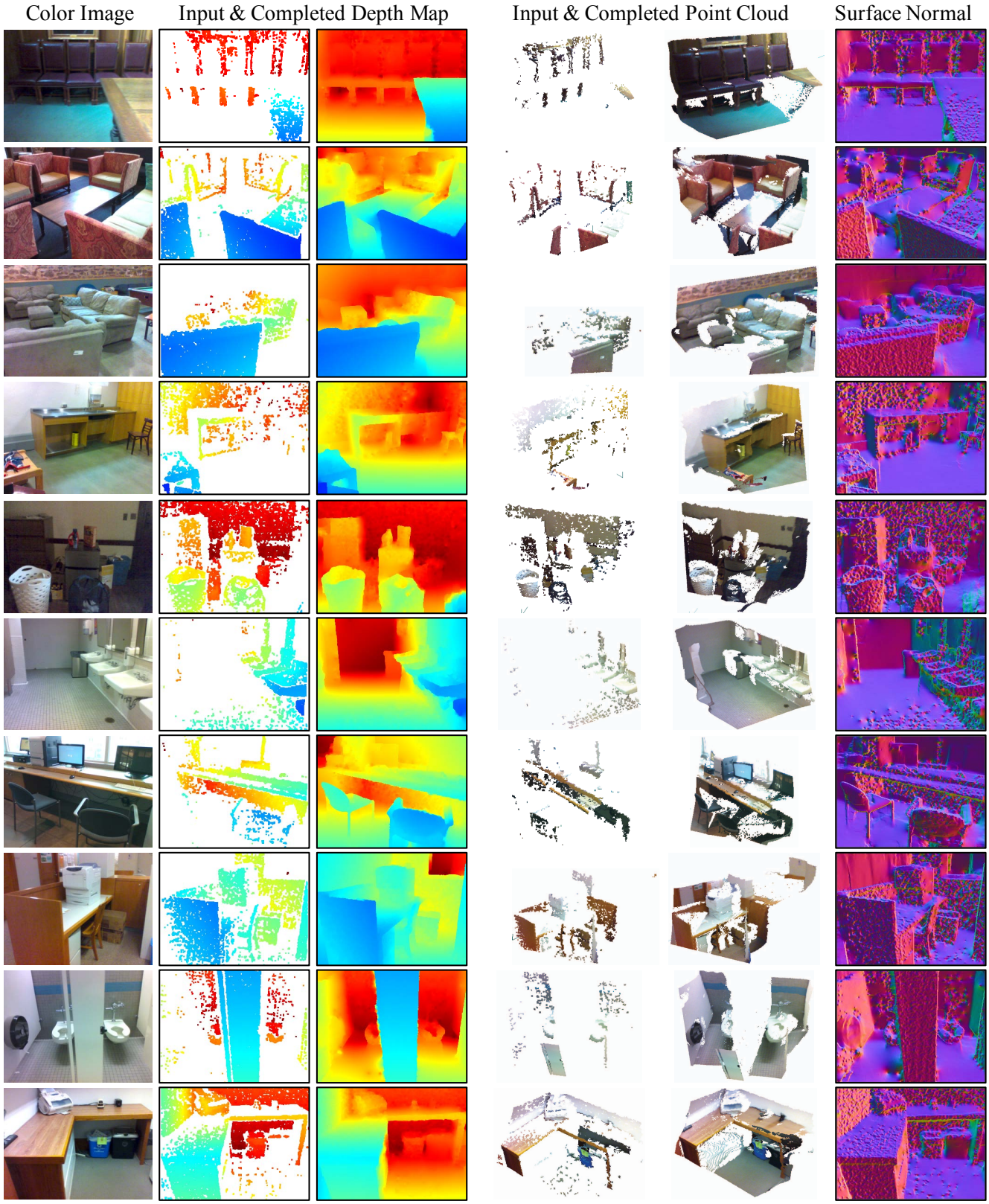

Figure 3. **Our results on RealSense data.** We run a model trained from ScanNet dataset and test on RGBD images captured by Intel RealSense without finetune. From left to right, we show the input color image, input depth image, completed depth image using our method, the point cloud visualization of the input and completed depth map, and the surface normal converted from the completed depth. Our method generates good results for depth completion.

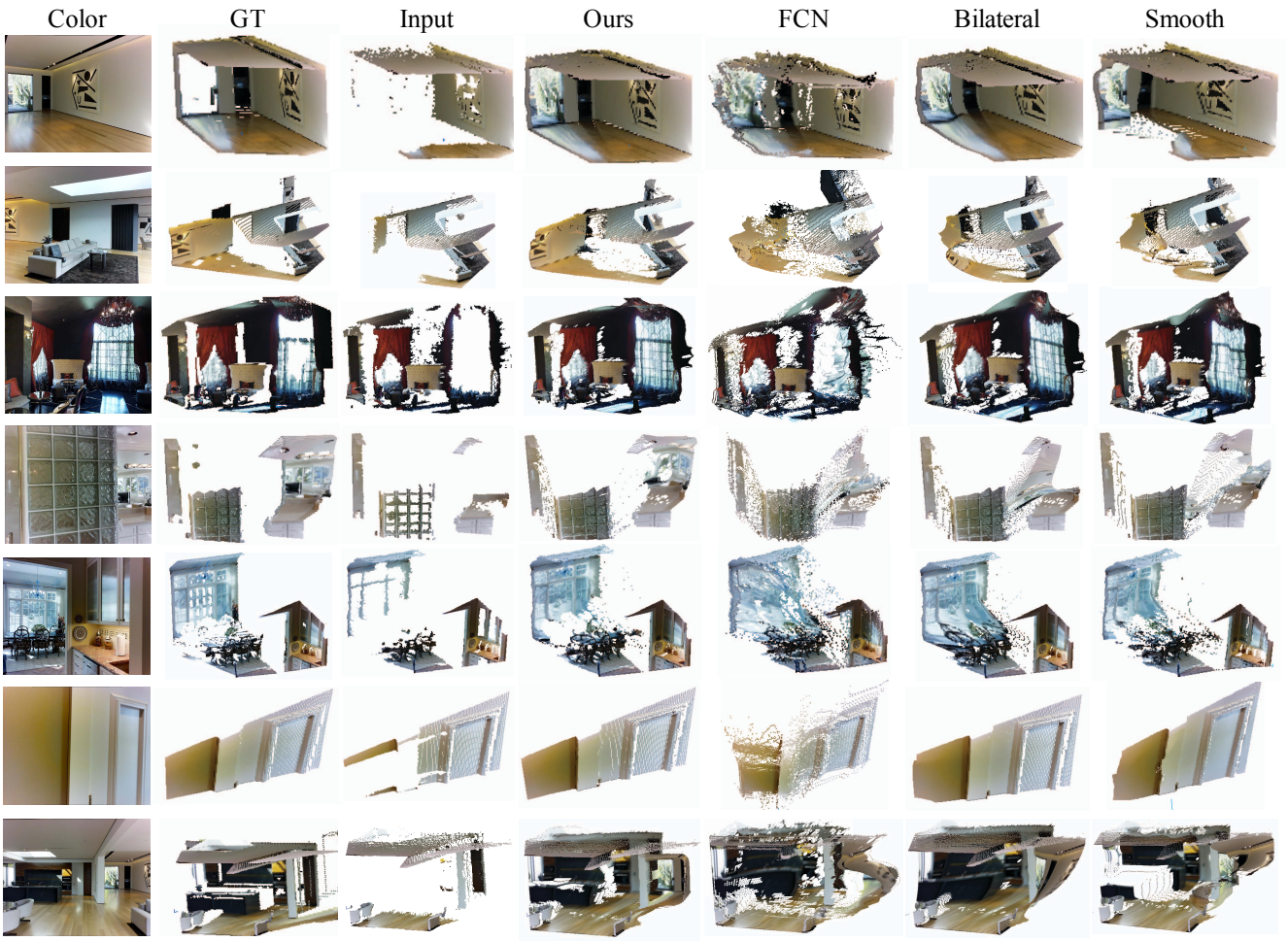

Figure 4. **Point cloud visualization of our method and other comparisons.** We convert the completed depth into point cloud. Our model produces better 3D geometry with fewer bleeding issue at the boundary.

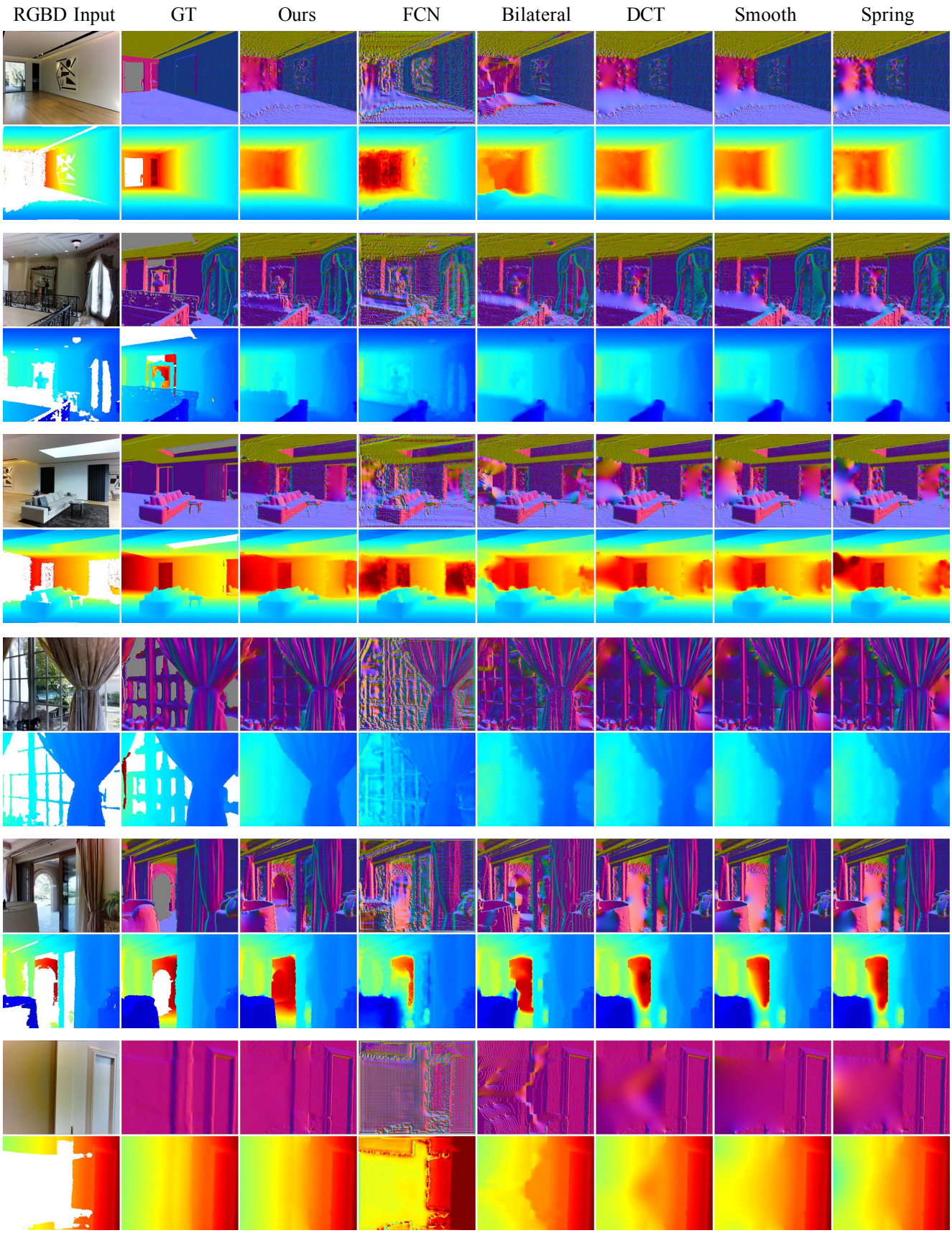

Figure 5. **More results and comparison to inpainting methods.** Each example is shown in two rows, where the second row shows the input, ground truth, and completed depth, whereas the first row shows the surface normal of each corresponding depth map on the second row to highlight details. Our method in general works better than other inpainting methods.

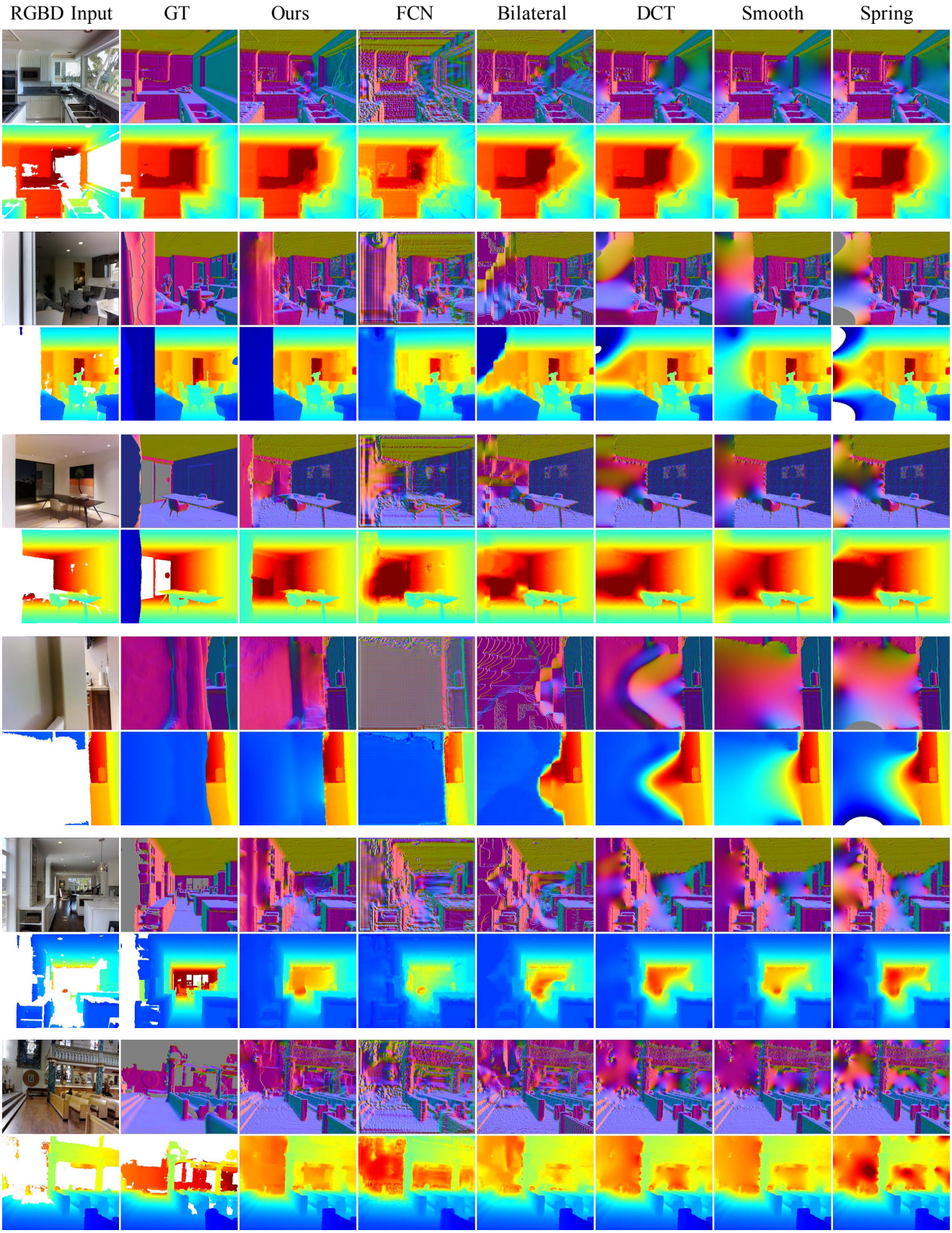

Figure 6. **More results and comparison to inpainting methods.** Each example is shown in two rows, where the second row shows the input, ground truth, and completed depth, whereas the first row shows the surface normal of each corresponding depth map on the second row to highlight details. Our method in general works better than other inpainting methods.

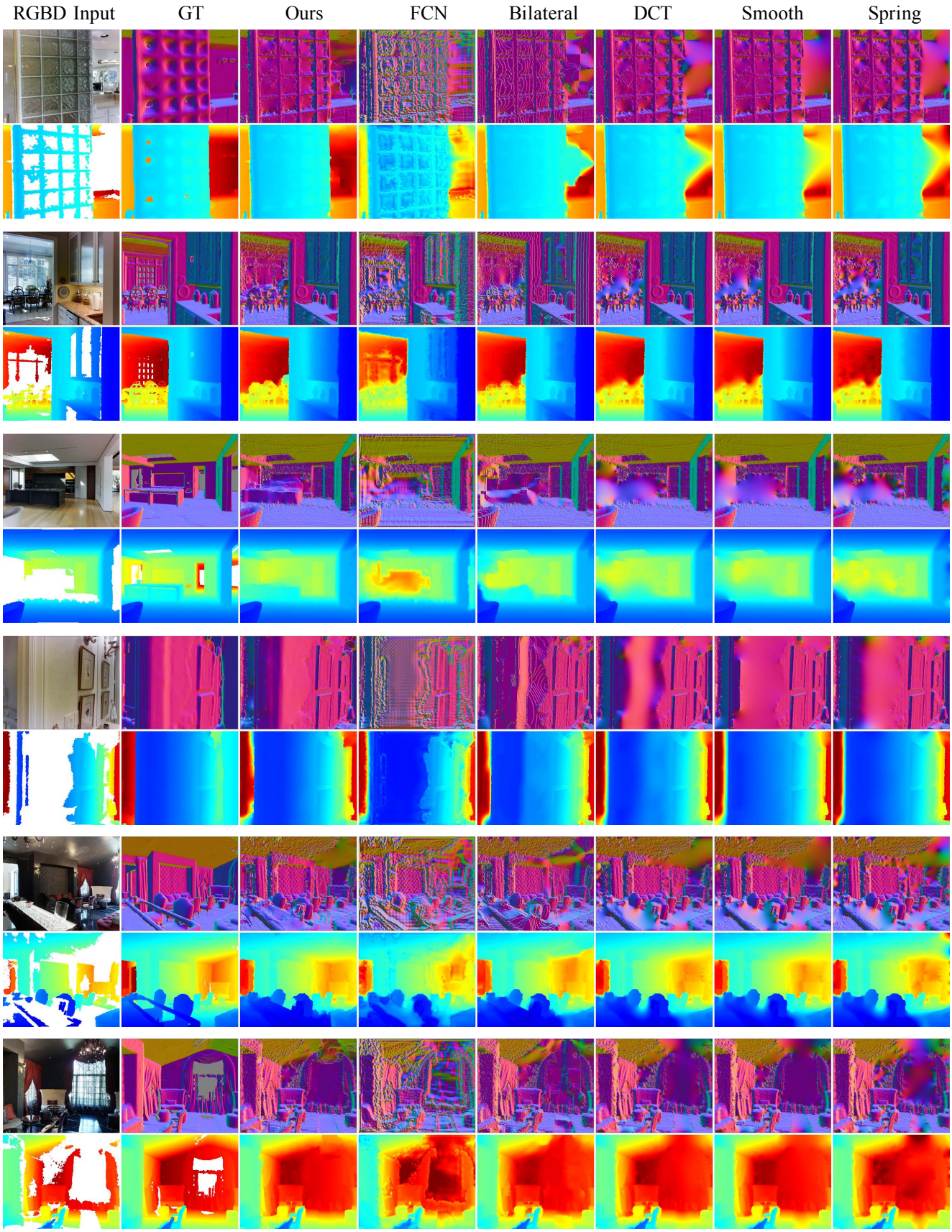

Figure 7. **More results and comparison to inpainting methods.** Each example is shown in two rows, where the second row shows the input, ground truth, and completed depth, whereas the first row shows the surface normal of each corresponding depth map on the second row to highlight details. Our method in general works better than other inpainting methods.

References

[1] Kinect for windows. https://developer.microsoft.com/en-us/windows/kinect. 4

[2] Structure sensor. https://structure.io/. 3

[3] A. Chakrabarti, J. Shao, and G. Shakhnarovich. Depth from a single image by harmonizing overcomplete local network predictions. In *Advances in Neural Information Processing Systems*, pages 2658–2666, 2016. 3

[4] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 4

[5] J. D'Errico. Inpaint nans, 2017. www.mathworks.com/matlabcentral/fileexchange/4551-inpaint-nans. 4

[6] D. Garcia. Robust smoothing of gridded data in one and higher dimensions with missing values. *Computational statistics & data analysis*, 54(4):1167–1178, 2010. 4

[7] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(3):29, 2013. 1

[8] X. Mao, C. Shen, and Y.-B. Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In *Advances in Neural Information Processing Systems*, pages 2802–2810, 2016. 4

[9] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016. 4

[10] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. *Computer Vision–ECCV 2012*, pages 746–760, 2012. 4

[11] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015. 4

[12] R. Zhang, J.-Y. Zhu, P. Isola, X. Geng, A. S. Lin, T. Yu, and A. A. Efros. Real-time user-guided image colorization with learned deep priors. *ACM Transactions on Graphics (TOG)*, 9(4), 2017. 1

[13] Y. Zhang, S. Song, E. Yumer, M. Savva, J.-Y. Lee, H. Jin, and T. Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 1, 4