Supplementary Material for GigaDepth: Learning Depth from Structured Light with Branching Neural Networks

Simon Schreiberhuber[1][0000−0002−1138−7576], Jean-Baptiste Weibel[1][0000−0003−0201−4740], Timothy Patten[2][0000−0003−1139−9451], and Markus Vincze[1][0000−0002−2799−491X]

[1] ACIN, TU Wien, Gusshausstrasse 27-29 / E376, 1040 Vienna, Austria
lastname@acin.tuwien.ac.at
[2] UTS Robotics Institute, 81 Broadway, Building 11, NSW 2007 Sydney, Australia
firstname.lastname@uts.edu.au

1 Simulating the Structure Core

We simulate the Structure Core sensor by utilizing the Unity3D game engine and its High Definition Rendering Pipeline (HDRP), a deferred renderer that is capable of supporting most modern visual effects. However we only use a moderate complement of the rendering capabilities since we care for only a few effects: lighting by different point-light sources, shadowing, reflection by reflection probes, texture, physically-based surfaces shading, bloom and auto exposure. We do not rely on any ray tracing, screen space reflection or any form of ambient occlusion.

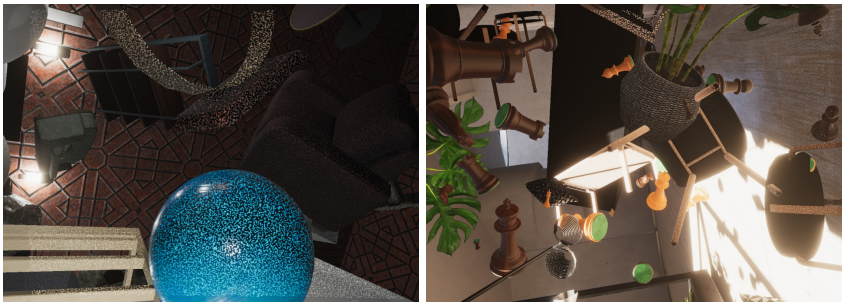
With this setup we render 15k scenes with 4 frames each for training and 10k single frames for testing, each of which have hundreds of randomly selected textured and positioned objects spawned in the camera's vicinity. Both training as well as test data use a completely separate set of assets and parameters including textures, models and density of clutter. See fig. 1 for examples.

1.1 Pattern extraction

The dot-pattern for the simulated projector is captured by physically pointing a real Structure Core at a flat surface and reprojecting the points to a virtual (rectified) camera at the projector position. To capture the peripheral speckles that cannot be captured by this method, we employ a secondary sensor (Intel PrimeSense D435), map these points and align them with the central part of the pattern. The final pattern is reconstructed by drawing a Gaussian intensity distribution around the center points. See fig. 2 for some of the captured data and results.

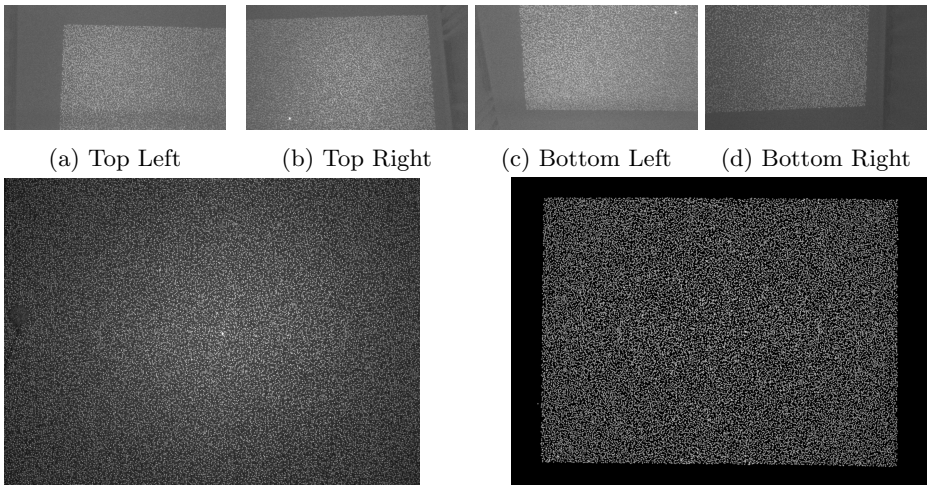
1.2 Masks

To generate a mask for surfaces that are influenced by the dot-pattern projector, we render two additional images for each camera not utilizing any of the typical post-processing image effects (e.g. auto exposure, bloom) of Unity3D: one



(a) training image (b) test image

Fig. 1: Images for training as well as testing use different objects, textures and lighting conditions.



(a) Top Left (b) Top Right (c) Bottom Left (d) Bottom Right

(e) Center (Structure Core) (f) Result

Fig. 2: While the center of the dot-pattern is captured via the Structure Core, the peripheral (a to d) are captured with a RealSense (D415) camera. The resulting pattern is reconstructed from the aligned, accumulated and clustered center points of 18 captured frames.

with the projector disabled as well as one with the pattern replaced by a white rectangle. Wherever these two images differ sufficiently enough, we can expect a pattern to be visible (fig. 3). The resulting mask is used to attenuate the loss where a dot-pattern is not visible and the training signal is ambiguous.

To increase sharpness on object boundaries, we increase the weight on the loss applied in their region. For this purpose we extract a mask from the depth-map as seen in fig. 4. This is done by Sobel filtering, thresholding and dilating the ground-truth depthmap.

(a) Projector off (b) Projector on (rectangle) (c) Resulting mask

Fig. 3: We mask all pixel that show enough of the pattern to reasonably attempt depth estimation. This is done by comparing two images: one taken with active IR illuminator and one without thresholding.

(a) simulated IR (b) depth (c) edge mask

Fig. 4: Starting with the ground-truth depth we use a Sobel filter followed by thresholding and dilation to extract masks for depth discontinuities. We use those to emphasize the loss on the edges, which increases sharpness at object boundaries and depth discontinuities.

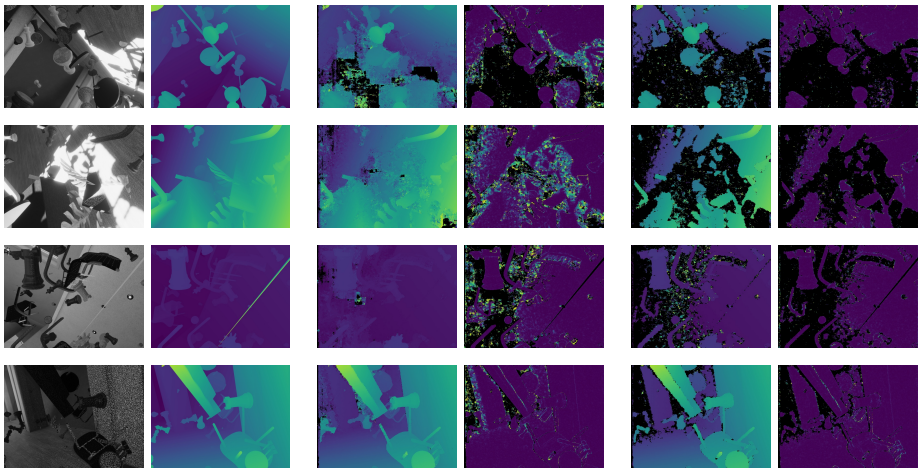
2 Ablation study

The ablation study demonstrates the influence of increased class count on our model's performance. In this supplementary we provide the reader with a more complete account of the exact composition of the MLPs at each tree-level. Similarly, the use of weight-sharing strategies are further elaborated. See table 1 for a precise account of these hyperparameters used in the experiments.

The experiment also features the usage of an UNet as backbone demonstrating the influence of this much heavier feature extractor on the overall performance (see fig. 5 and fig. 6 for qualitative results on real and rendered data). While the combination of UNet backbone and our regressor does consume orders of magnitude more time to compute ($1s$ vs. $60ms$), it does not produce better outlier ratios for thresholds lower than 0.5 pixels. It shows however, visibly and measurably, more complete results when higher thresholds for disparity are acceptable. This is either by being more sensitive to the attenuated and fragmented pattern or by better interpolation in challenging situations.

Table 1: Different configurations of our architecture tested on synthetic data. We test two backbones, ours as well as a full UNet network. The information that was omitted in the original paper is the number of classes (c_1, c_2, c_3) at each stage of the classification tree and the overlap (o_1, o_2, o_3) between the raw classification output of neighbouring nodes. We also share the weights of the hidden layers of consecutive classes in our regressor MLPs (not in the final output layer). The depth of each node (not leaf) MLP is stated in $^{o}/_{l}$ and does not describe the width hidden layers of the used MLPs however. For $l = 1$ we do not have hidden layers, $l = 2$ is described in our main document and actually uses a three staged classifier at the root nodes. For $l = 3$, the stage one would work with [64, 64, 32, 32, $^{c}/_{o}$] in the root node and [64, 32, 32, $^{c}/_{o}$] in the remaining nodes ([*in*, *hidden*, $^{c}/_{o}$] notation). The structure of the regressor MLPs was kept the same in all experiments [32, 32, $^{1}/_{1}$].

B.B.	UNet		Our Backbone									
Reg.	none	lines	$^{1920}/_{2}$	$^{288}/_{2}$	$^{384}/_{2}$	$^{640}/_{1}$	$^{640}/_{2}$	$^{640}/_{3}$	$^{1280}/_{2}$	$^{1920^a}/_{2}$	$^{1920}/_{2}$	$^{2688}/_{2}$
c_1			16	8	16	16	16	16	16	16	16	16
c_2, o_2			12, 2	6, 1	6, 1	10, 3	10, 3	10, 3	10, 3	12, 2	12, 2	14, 1
c_3, o_3			10, 3	6, 1	4, 2	4, 2	4, 2	4, 2	8, 4	10, 3	10, 3	12, 2
w. sharing			4	1	2	2	2	2	3	4	4	6

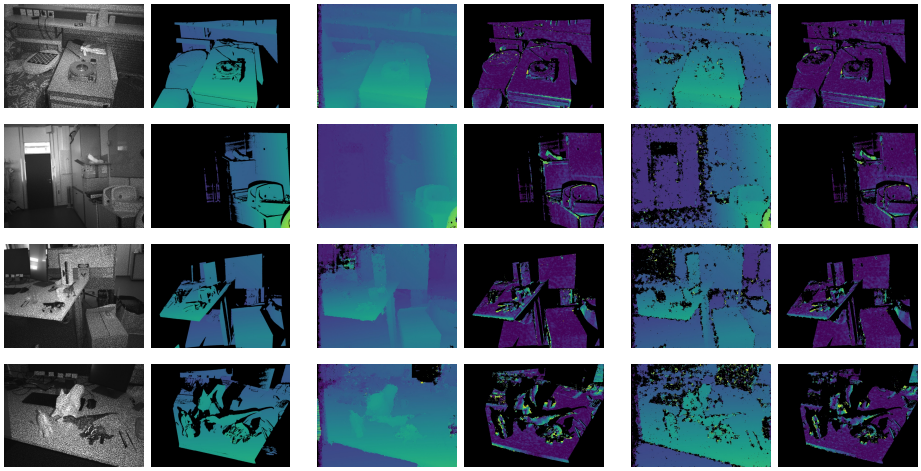


(a) IR + GT (b) GD + UNet (c) GD (ours)

Fig. 5: Disparities (left b, c) of our algorithm applied on scenes rendered with Unity3D. Color coding is applied for disparity errors (right b, c) of 0-5 pixels with outliers (> 5 pixels) being black. While our algorithm with a heavier UNet backbone (b) produces more complete results, it requires $1s$ for computation. Our backbone (c) achieves a better trade-off with an execution time of $60ms$.

3 Local Contrast Normalization

To accentuate the features of the projected pattern and attenuate the influences of ambient lighting and surface characteristics, we employ LCN, as seen in fig. 7,



(a) IR + GT (b) GD + UNet (c) GD (ours)

Fig. 6: Disparities (left b, c) of our algorithm applied on scenes captured with the Occipital Structure Core. Color coding is applied for disparity errors (right b, c) of 0-5 pixels with outliers (> 5 pixels) being black. While our algorithm with a heavier UNet backbone (b) produces more complete results, it requires $1s$ for computation. Our backbone (c) achieves a better trade-off with an execution time of $60ms$.

as a pre-processing step. The output and the default infrared input image are concatenated to form the input data for the backbone. The same filter has been employed in the works of Riegler et al. [5] and Zhang et al. [6] to either guide the training or help detection at runtime.

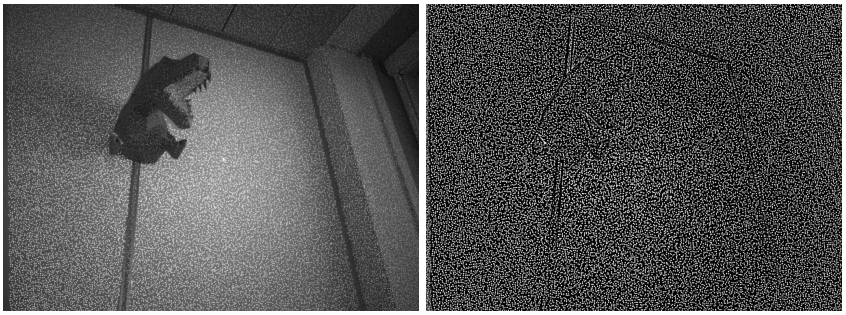
The fundamental idea is to remove low-frequency components of the image and scale up the high-frequency signal with two operations. First, the intensity I of the intput image is adjusted around the mean $\mu_I(\boldsymbol{x})$ at each pixel position $\boldsymbol{x}$. In the second step, we scale/normalize the output by the standard deviation $\sigma_I(\boldsymbol{x})$,

$$LCN(I, \boldsymbol{x}) = \frac{I(\boldsymbol{x}) - \mu_I(\boldsymbol{x})}{\sigma_I(\boldsymbol{x}) + \epsilon} \tag{1}$$

The mean and standard deviation ($\mu_I(\boldsymbol{x})$, $\sigma_I(\boldsymbol{x})$) are calculated in a small neighbourhood of 11 pixels around the center $\boldsymbol{x}$ and thus represent local measures that reflect the diverse lighting situations that appear within frames.

4 Isolated Comparisons

While the datasets used in [5, 4] are perfectly suitable to demonstrate the respective abilities for semi-supervision, they are not well suited to prepare our



(a) IR input (b) LCN

Fig. 7: Local Contrast Normalization (LCN) is used to accentuate the pattern of the IR projector.

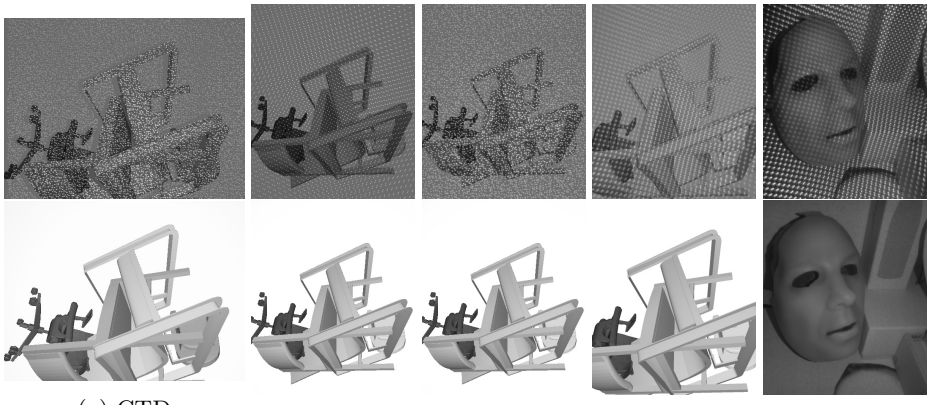
(a) IR image (b) Ambient image

Fig. 8: For 241 of 967 images our dataset provides an "ambient" image. Images taken with the same camera at the same pose but with deactivated IR projector.

fully-supervised method for the simulation-to-real domain gap. To prepare our algorithm for real-world application we rely on cluttered renderings, with textures and diverse lighting conditions that seem to be too challenging for the training strategies of [5, 4]. This is despite our attempts of adapting the cost function, fully supervised pre-training or capturing the ambient images (fig. 8) required by [5, 4]. We thus also train and test our method on the datasets native to [5, 4].

4.1 Connecting The Dots

Riegler et al. [5] use a synthetic as well as a captured dataset to train and test their method. The synthetic dataset is comprised of 9216 scenes featuring 4 frames each (640×480) and objects of the "chair" class taken from ShapeNet [1]. These objects are put in front of a single untextured plane and are lit by environment light and a projector that emits a crop of a pattern that is used by a



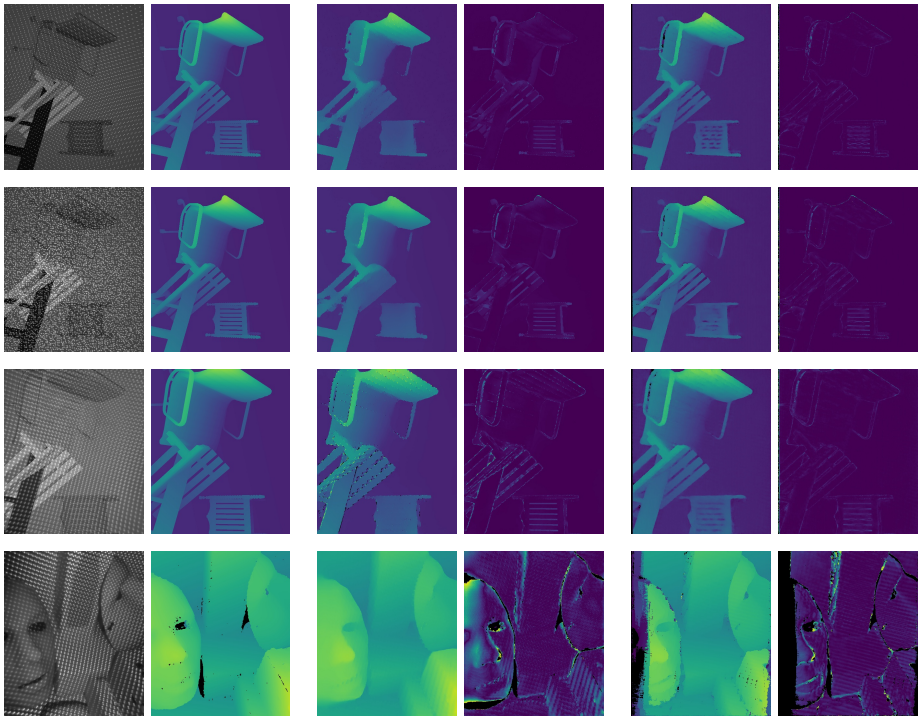
(a) CTD (b) DIS default (c) DIS Kinect (d) DIS real (e) DIS

Fig. 9: The training datasets of Connecting The Dots (CTD) compared to the one of Depth In Space (DIS). The rendered data (a - d) completely relies on textureless models drawn from the "chair" class of the ShapeNet [1]. The used Kinect pattern is a crop of the original pattern (a, d) to achieve a speckles per pixel ratio similar to the original sensor. The completely artificial pattern (b), as well as the captured real pattern (d, e) in comparison show a lower density of speckles. Sequences captured with the real hardware (e) only capture few sequences, and do not strictly split between training, validation and test set. See the last row fig. 10 for a frame of the test set.

Microsoft Kinect v1 or PrimeSense Carmine (see fig. 9a). In the original publication, these renders are used to train (index 1024 to 9216) as well as validate/test (index 0 to 1024) the algorithm.

Instead of the full (Kinect/PrimeSense) pattern with a resolution of 1280×1024, a 640×480 pixel crop is used. This preserves the pixel density compared to data captured by the original sensor, as the raw frames are captured at 1280×1024 while the renderings happen at 640×480. For the real-world experiments it is not entirely clear which training data is used as the training regime requires some captures with deactivated projector to train the edge-detection network. This makes recreating the results in [5] challenging. The testing however has been conducted on a PrimeSense based dataset provided in [2]. Training seems to happen on this dataset as well as additional captured, unpublished data. It is noteworthy that these images are downsampled from 1280×1024 to 640×480 leading to an input with different pattern density compared to the cropping approach used during rendering.

While we cannot train our algorithm for the real-world data provided in [2], we do provide a comparison on synthetic data in our main paper. However, we deviate from [5] by rendering a test set consisting of 1024 scenes with objects of the "airplane", "car" and "watercraft" to reduce the similarity to the data the



(a) IR + GT (b) DepthInSpace (c) GigaDepth (ours)

Fig. 10: Disparity (left b, c) estimates by DepthInSpace and GigaDepth applied on the different test datasets native to DepthInSpace. Color coding is applied for disparity errors (right b, c) of 0-5 pixels with outliers (> 5 pixels) being black. The rows from top to bottom show: rendered with artificial pattern (default), rendered with kinect pattern, rendered with captured pattern and captured data.

algorithm is trained on. We also modify our backbone CNN for this experiment to not downsample but to operate at the full input resolution.

4.2 DepthInSpace

The synthetic datasets used in [4] uses the same methodology of [5] but differs in the aspect ratio of the images (512×432 vs. 640×480) and the used patterns (see fig. 9). In comparison to [5] the authors have been more clear about the data used to train the real-world variant of their network. The public dataset contains 148 sequences of 4 frames each that have been sampled from 4 longer continuous video captures mostly showing close-range masks and simple scenes. Unfortunately the sampling for the validation data is the same as the one for testing and consists of every 8th frame in those sequences. As a result, the training data is very similar to the target/test domain. We further note that the

Table 2: We extended table 1 of [4] with our evaluation. Three of the datasets are rendered with a completely artificial pattern (default), a pattern captured from a Microsoft Kinect v1 and one captured from the intended sensor (captured). The last set of experiments is based on real data. The compared methods are: semi-global matching (SGM), HyperDepth (HD), Connecting The Dots (CTD), Depth In Space for single frames (DIS-SF), Depth In Space on single frames with fine-tuning (DIS-FTSF) by its multi-frame variant, Depth In Space with a spatial network capable of consolidating the results of multiple single-frame results (DIS-MF) and finally our GigaDepth (GD). The best single-frame results are presented in bold.

data	method	$o(0.5)$	$o(1)$	$o(2)$	$o(5)$
Synthetic Kinect	SGM	10.36	9.13	8.76	2.45
	HD	4.38	3.22	2.69	2.39
	CTD	2.74	1.45	0.77	0.24
	DIS-SF	2.11	1.13	0.59	0.16
	DIS-FTSF	1.92	**1.00**	**0.51**	**0.14**
	DIS-MF	1.59	0.72	0.33	0.10
	GD (ours)	**1.88**	1.63	1.39	0.82
Synthetic default	SGM	12.93	11.64	11.22	4.06
	HD	7.35	6.48	6.11	5.86
	CTD	3.38	1.71	0.85	0.28
	DIS-SF	2.31	1.24	0.62	0.19
	DIS-FTSF	1.96	**0.95**	**0.45**	**0.12**
	DIS-MF	1.58	0.71	0.32	0.10
	GD (ours)	**1.75**	1.48	1.20	0.82

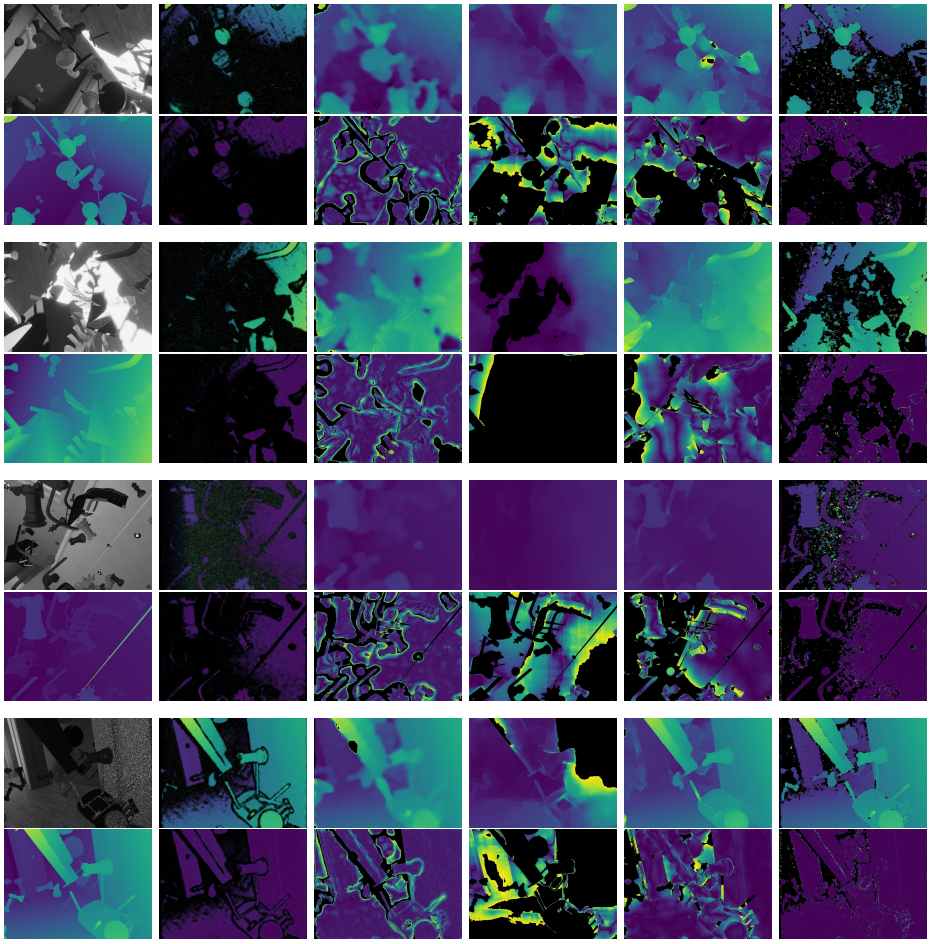
data	method	$o(0.5)$	$o(1)$	$o(2)$	$o(5)$
Synthetic captured	SGM	12.45	10.37	9.55	4.83
	HD	6.13	4.92	4.34	4.00
	CTD	3.76	2.25	1.03	0.37
	DIS-SF	3.66	2.16	1.00	0.23
	DIS-FTSF	2.87	**1.48**	**0.66**	**0.17**
	DIS-MF	2.46	1.24	0.54	0.14
	GD (ours)	**2.41**	2.82	1.24	0.86
real	SGM	25.54	19.23	17.75	16.96
	HD	34.62	25.09	22.49	21.77
	CTD	22.74	9.26	3.79	**1.00**
	DIS-SF	17.95	7.93	3.59	1.14
	DIS-FTSF	**17.06**	**7.48**	**3.47**	1.11
	DIS-MF	16.07	7.14	3.41	1.09
	GD (ours)	18.59	12.06	10.34	8.80

utilized patterns are sparser and more regular than the ones used in our dataset or the one used in [5]. We thus modify our backbone network to not downsample and increase the receptive field to compensate for the lower pattern density. For HyperDepth [3], whose random forests operate on a hardcoded window of 32×32 pixel, this might be a factor contributing to decreased performance.

Despite these reasons we extend table 1 of [4] to demonstrate our algorithm's competitive performance for applications where accuracy better than 0.5 pixels is required and ground-truth training data can be supplied (table 2). Qualitative results in fig. 10 show that GigaDepth performs well on captured data even though we do not perform any fine-tuning. On the left side of the captured images, however, we register a decline in performance. We assume this is because the synthetic training data is barely showing this region of the pattern. The surfaces and objects are unfortunately rendered at a rather high distance.

5 Additional Qualitative Results

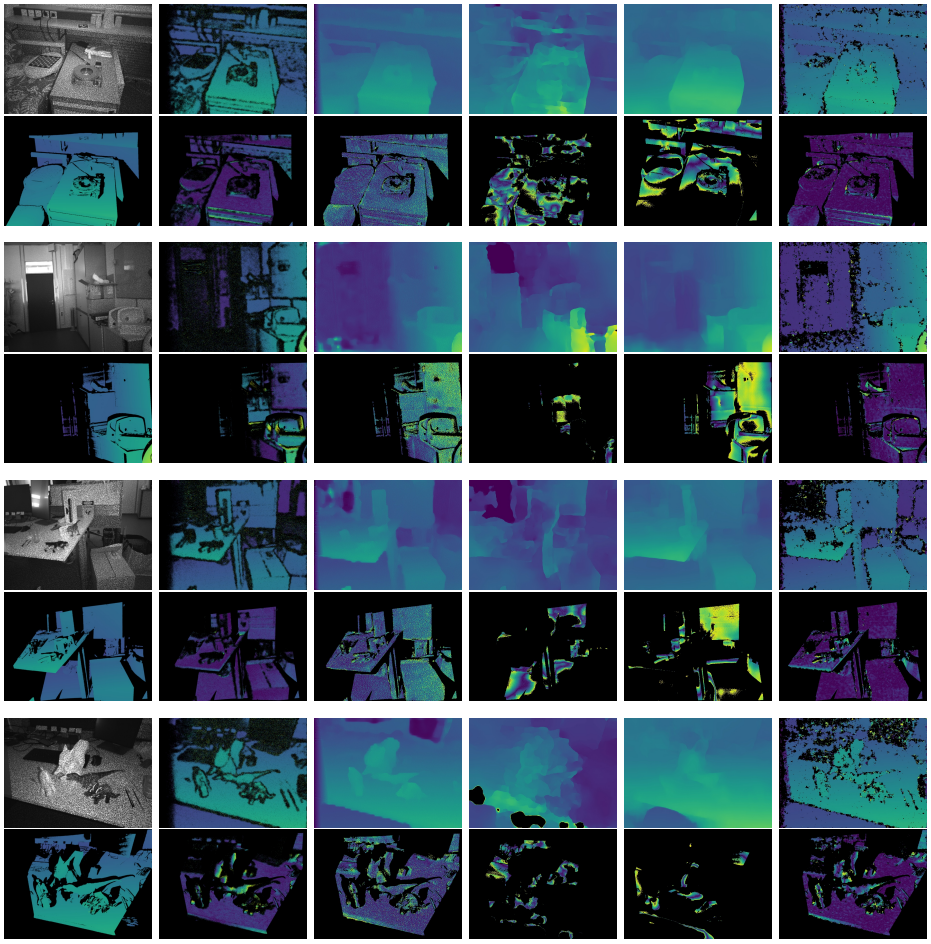
The figs. 11 and 12 provide additional qualitative results on our rendered as well as captured test datasets. These images once more illustrate our algorithm's potential but also that it is, albeit resillient, still susceptible to the domain gap.



(a) IR + GT (b) HD (c) ASN (d) CTD (e) DIS (f) GD (ours)

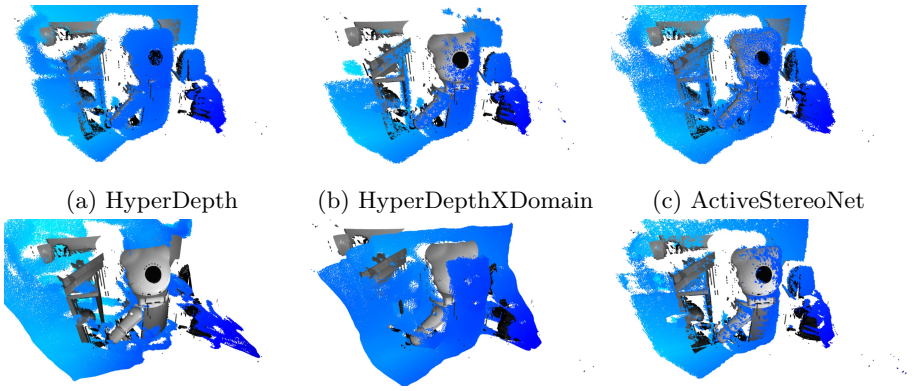
Fig. 11: Disparities (top b - f) of different algorithms applied on scenes rendered with Unity3D. Color coding is applied for disparity errors (bottom b - f) of 0-5 pixels with outliers (> 5 pixels) being black. Algorithms are: HyperDepth (b, HD), ActiveStereoNet (c, ASN), Connecting The Dots (d, CTD), DepthInSpace (e, DIS) and GigaDepth (e, GD).

In fig. 13, we show the ICP alignment of the different methods to the ground-truth pointcloud captured by the Photoneo MotionCam-3D M. Even though each of these pointclouds are aligned individually, some of the tested algorithms produce results that just cannot be properly matched with the ground-truth.



(a) IR + GT (b) HD (c) ASN (d) CTD (e) DIS (f) GD (ours)

Fig. 12: Disparities (top b - f) of different algorithms applied on scenes captured by the Occipital Structure Core. The ground-truth (GT) is captured with the Photoneo MotionCam-3D M and compared to other algorithms. Color coding is applied for disparity errors (bottom b - f) of 0-5 pixels with outliers (> 5 pixels) being black. Algorithms are: HyperDepth (b, HD), ActiveStereoNet (c, ASN), Connecting The Dots (d, CTD), DepthInSpace (e, DIS) and GigaDepth (e, GD).



(a) HyperDepth (b) HyperDepthXDomain (c) ActiveStereoNet

(d) Connecting The Dots (e) DepthInSpace-FTSF (f) GigaDepth (ours)

Fig. 13: The pointcloud captured with the Photoneo Scanner (greyscale) aligned with the cloud captured by the Structure Core (blue). The pointclouds resulting from the algorithms (blue) are aligned with the ground-truth (greyscale) via ICP.

References

1. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An information-rich 3D model repository. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago (2015)
2. Chen, Q., Koltun, V.: Fast MRF optimization with application to depth reconstruction. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3914–3921 (2014)
3. Fanello, S.R., Rhemann, C., Tankovich, V., Kowdle, A., Escolano, S.O., Kim, D., Izadi, S.: HyperDepth: Learning depth from structured light without matching. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5441–5450 (2016)
4. Johari, M., Carta, C., Fleuret, F.: DepthInSpace: Exploitation and fusion of multiple video frames for structured-light depth estimation. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 6019–6028 (2021)
5. Riegler, G., Liao, Y., Donne, S., Koltun, V., Geiger, A.: Connecting the Dots: Learning representations for active monocular depth estimation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7616–7625 (2019)
6. Zhang, Y., Khamis, S., Rhemann, C., Valentin, J.P.C., Kowdle, A., Tankovich, V., Schoenberg, M., Izadi, S., Funkhouser, T.A., Fanello, S.R.: ActiveStereoNet: End-to-end self-supervised learning for active stereo systems. In: Proceedings of the European Conference on Computer Vision. pp. 802–819 (2018)