TurboSL: Dense, Accurate and Fast 3D by Neural Inverse Structured Light: Supplemental Document

Parsa Mirdehghan[1,2] Maxx Wu[1,2] Wenzheng Chen[1,2] David B. Lindell[1,2] Kiriakos N. Kutulakos[1,2]
[1]University of Toronto [2]Vector Institute
{parsa,wumaxx,wenzheng,lindell,kyros}@cs.toronto.edu

A. Supplemental Implementation Details

A.1. Training Details

We begin by describing the details of our network architecture and its optimization.

Network architecture. Our model is built on the existing NeuS implementation [3, 9] which incorporates the NerfAcc [6] backbone for efficient sampling. Our model consists of three Instant-NGP[8] networks (MLPs with multi-resolution hash encodings) for the signed distance function (SDF), reflectance, and residual contributions of indirect or ambient illumination. The SDF network takes as input a 3D point in space and outputs its SDF, while the reflectance and residual networks query 2D points on the image plane and output the points' reflectance and residual contributions. For all three networks, the number of hash feature grids and the feature size for each level are set to 16 and 2, respectively. The resolution of the coarsest grid is 16 and each level has a scale factor of 1.45. The base MLPs for all three networks have 2 hidden layers with width 64 and a ReLU activation. The reflectance and residual networks employ sigmoid activations at the output to constrain the values to between zero and one. We omit the activation function at the output of the SDF network so that the output range is unconstrained.

Occupancy grid. The NerfAcc backbone maintains an occupancy grid to omit empty regions for efficient sampling. The grid's resolution is set to 128 in each dimension, and the density threshold for pruning is set to 10^{-3}. We used 1024 samples per ray to render each pixel, where some of these samples are skipped if they fall within empty regions of the occupancy grid. All the scenes are scaled and optimized within a bounding box of radius 1.5.

Loss function. The loss function used to train the model consists of five main components: (1) the camera loss that compares the rendered image with the input structured light (SL) images; (2) the projector loss that compares the rendered pattern with the input projection patterns; (3) the sparsity loss that encourages sparse reconstruction leading to fewer regions of spurious density; (4) the mask loss that encourages the model to converge to an opaque surface for any ray inside the mask and empty space for rays outside the mask; and (5) the Eikonal loss to regularize the SDF [2]. The camera and projector losses are both weighted sums of L1 and L2 penalty functions, where we set the weights to 1.0 and 10, respectively. We find that adding the L1 loss improves the reconstruction in low reflectance regions. For the mask loss, we estimate the mask by thresholding the difference between the minimum and maximum pixel intensities across all the captured images. If the difference is higher than a certain threshold, we consider the pixel to be visible to the projector and set the mask value to one (and otherwise to zero). The weights of the camera loss, projector loss, sparsity loss, Eikonal loss, and mask loss are set to 1.0, 1.0, 0.01, 0.1, and 0.1, respectively.

Optimization and runtime. Our models usually converge in 10k iterations which takes around 12 minutes on a single Quadro RTX 6000 GPU. Initially, in each iteration we randomly sample 256 rays from the camera and projector planes, and render their respective images. As the optimization progresses, the number of effective samples per ray decreases from the original 1024 samples because pruned-out regions in the occupancy grid are no longer sampled. Therefore, we dynamically increase the number of rays during each training iteration to keep the total number of samples constant over the course of training (without exceeding a maximum of 8192 rays).

A.2. Projector Loss

Here, we describe the challenges of incorporating the projector loss and our proposed solutions.

Accounting for noise when rendering the projector pattern. While the input projector patterns are noise-free, the camera images include measurement noise. Rendering the projector pattern with the reverse rendering procedure (Equation T5) thus results in a noisy pattern. Further, this noise is divided by the product of the reflectance and a cosine similarity term. Therefore, we multiply the per-pixel mean squared error loss of the rendered projector rays by the square of this scale factor. The per-pixel L1 loss of the rendered projector rays is multiplied by the the absolute value of this scale factor. Scaling the loss functions in this way puts more weight on rays corresponding to higher albedo (defined as reflectance multiplied by the cosine similarity term) and reduces the contribution of noisy pixels to the reconstruction.

Numerical instability of the gradients. Rendering the projector rays using Equation T5 involves a division by a scale factor that includes the reflectance and the cosine similarity term. This division, particularly in the initial stage of optimization, leads to exploding gradients and inhibits convergence during optimization. To address this, we used two solutions. First, we clamped the denominator to a minimum of 0.1 and maximum of 1. This procedure zeros the gradient when the denominator is smaller or greater than the threshold. Second, we begin the optimization using only the camera loss for 1k iterations before switching to use both the camera and projector losses for the remaining 9k iterations.

Lack of mask for projector image plane. Unlike the camera image, there is no convenient way to estimate a mask for the projector. Naively sampling from the projector plane and evaluating the loss between the input projector pattern and the rendered one leads to creation of density in empty space along each projector ray. To avoid this, we composite the rendered image onto a background that is identical to the projector pattern. This ensures that zero density regions are not penalized, leading to non-zero density only in regions that affect the camera image.

A.3. Blur Kernel Optimization

In this section, we describe the details of optimizing the projector blur kernel.

In theory, the projector can be described using a pinhole model. However, in practice, the mapping from the projector input to output is a manufacturer-dependent black-box function and involves a series of operations such as blurring or sharpening. We carefully adjust the projector to make it in focus and calibrate the projector to reduce the lens distortion. On top of that, we optimize a blur kernel to account for any additional black-box processing. We define the blur kernel B as a $K \times K$ matrix and apply it to the whole projector pattern, assuming it is spatially invariant. We use $K = 11$ in all experiments. To reduce the number of parameters, we optimize two separable one-dimensional filters B_x, $\mathrm{B}_y \in \mathbb{R}^K$ and use their outer product as the blur kernel B:

$$\mathrm{B} = \mathrm{B}_x \mathrm{B}_y^T \quad . \tag{1}$$

The kernel has extra constraints: first, the center of mass should be located in the center of the kernel; and second, the sum of the kernel should be equal to one to conserve the energy. Therefore, for each one-dimensional filter, we apply re-weighting and normalization to satisfy the constraints. Specifically, we break the filter into 3 parts: left half vector, center element, and right half vector:

$$\begin{aligned} K &= 2M+1 \\ \mathrm{B}_x &= [\mathrm{B}_x^l, \mathrm{B}_x^m, \mathrm{B}_x^r] \\ \mathrm{B}_x^l &= [b_{-m}, b_{-m+1}, ..., b_{-1}] \\ \mathrm{B}_x^m &= [b_0] \\ \mathrm{B}_x^r &= [b_1, b_2, ..., b_m] \quad , \end{aligned} \tag{2}$$

where K is an odd number and b_i is the i-th element of filter B_x for $i \in \{-m, ..., -1, 0, 1, ..., m\}$. We learn only the right and left parts and adjust the center element to satisfy the summation constraint. More specifically, we rescale B_x^r to match B_x^l to keep the kernel centered each time it is computed. We further define B_x^m to be equal to one minus the sum of B_x^l and B_x^r to

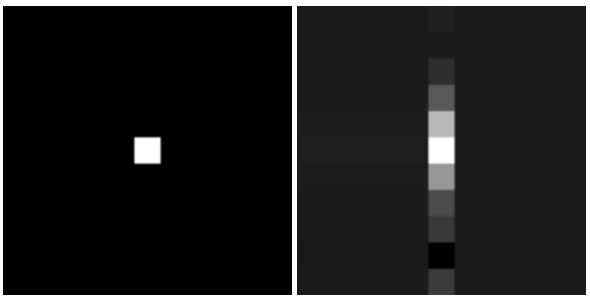

Figure S1: Blur kernel optimization. **Left:** Initial blur kernel. **Right:** The blur kernel learned after joint optimization with the SDF, reflectance, and the residual contributions of indirect and ambient illumination.

constrain the filter sum to be equal to one. These additional update rules are given as:

$$
\begin{aligned}
r &= -\frac{\sum_{i=-m}^{-1} i \cdot b_i}{\sum_{i=1}^{m} i \cdot b_i} \\
\hat{\mathrm{B}}_x^l &= \mathrm{B}_x^l \\
\hat{\mathrm{B}}_x^r &= r \cdot \mathrm{B}_x^r \\
\hat{\mathrm{B}}_x^m &= 1 - \sum \hat{\mathrm{B}}_x^l - \sum \hat{\mathrm{B}}_x^r \ ,
\end{aligned}
\tag{3}
$$

where $\{\hat{\mathrm{B}}_x^l, \hat{\mathrm{B}}_x^m, \hat{\mathrm{B}}_x^r\}$ are the normalized left, middle and right parts of B_x. We compute B_y in the same fashion. Fig. S1 shows the kernel at initialization and after convergence. Since we are projecting the patterns as horizontal stripes the learned kernel mainly learns a blur across rows.

A.4. Mesh Extraction

In the following, we describe how to extract a mesh from (1) depth predictions from SL methods based on pixelwise decoding and (2) the SDF predicted using TurboSL.

Meshing the pixelwise decoder output. We first generate a depth map based on the pixelwise decoder outputs. Then, we backproject the depth map to calculate point cloud points based on the calibrated camera intrinsics. The point cloud points are converted to a triangle mesh by adding edges to connect neighboring points (based on their positions in the depth map). We omit from the meshing procedure any point cloud points that fall outside the mask used to compute the mask loss.

Meshing the TurboSL output. To mesh the SDF predicted by TurboSL, we first estimate per-pixel depth using the expected ray termination distance [1, 6]. Similar to the previous case, we backproject the depth map points to recover point cloud points using the calibrated camera intrinsics, and we add edges to connect neighboring points based on the point positions in the depth map. Again, we do not mesh any points falling outside the mask used for the mask loss.

Meshing the TurboSL SDF using marching cubes. We use the above depth-based meshing procedure and the Blender renderer to create the surface visualizations used for most results in the main paper and supplement. We use marching cubes to create the meshes used for surface visualizations in Figure 1, Figures S3, S4, S12, and the dynamic scene results shown in the supplemental video from 0:07–0:34. Results shown using this procedure demonstrate that the SDF predicted by TurboSL can be meshed directly without the depth map intermediary.

Specifically, we use marching cubes with a resolution of 512^3 to extract the zero level set of the predicted SDF. This procedure extracts some spurious surfaces that are in occluded regions not visible to the camera or projector. To remove these surfaces, we iterate over all vertices in the extracted mesh, shoot a ray from the vertex to the camera center of projection, then check for intersections with the extracted mesh. If the ray intersects the extracted mesh, it indicates that the vertex is occluded and is invisible. We remove all such occluded vertices and all triangle faces that include at least one of the removed vertices. We further remove vertices and faces whose projections onto the camera image plane fall outside the mask used for the mask loss.

Figure S2 shows two examples of this process. The left image shows the original mesh extracted with marching cubes, and

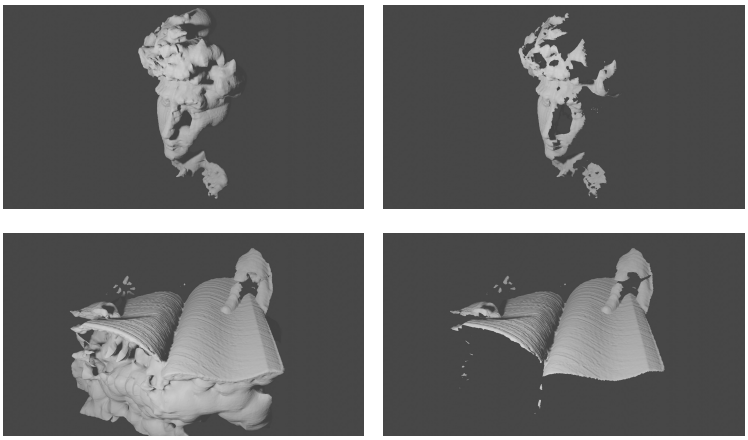

Figure S2: The original surface reconstructed with marching cubes (left) and the visible surface (right) extracted by removing occluded vertices and faces.

the right image shows the processed mesh. Row one corresponds to the scene in Figure 3 of the paper, and row 2 corresponds to the scene in Figure S3.

A.5. Hardware Implementation Details

For our experiments, we use two different SL systems. In both systems, we use a DMD-based LED mini projector (LG PH550) with 720×1280 resolution and 60 Hz refresh rate to project SL patterns onto the scene. However, we choose two different cameras to capture the static and dynamic scenes. For static scenes, the images are captured using a CMOS camera (IDS UI-3240CP, 1024×1280) with 1/60s or 1/30s exposure times.

For dynamic scenes, a quad-tap CCD scientific camera (AVT Prosilica GT1920c, 1024×1280) is used because it offers more precise control over the synchronization. This camera has a maximum frame rate of 30 fps, but the projector has a refresh rate of 60 Hz. To account for this, we project every pattern for two projector frames, synchronizing the system at an effective 30 Hz frame rate. To trigger the camera, we extract the V-sync signal from the projector using a VGA splitter.

For both systems, the camera-projector radiometric calibration is conducted by projecting grayscale images (i.e., pixel values from 0 to 255) onto a diffuse white plane. A polynomial response curve is fitted using images captured at the different projector intensities.

To evaluate the robustness of our framework, we capture the same scene using both SL systems and compare the results. Figures S3 and S4 show the reconstruction results for the book and David scenes. Please note that the captured scenes are not exactly identical; however, the results show that our framework achieves qualitatively similar performance when used with either SL system.

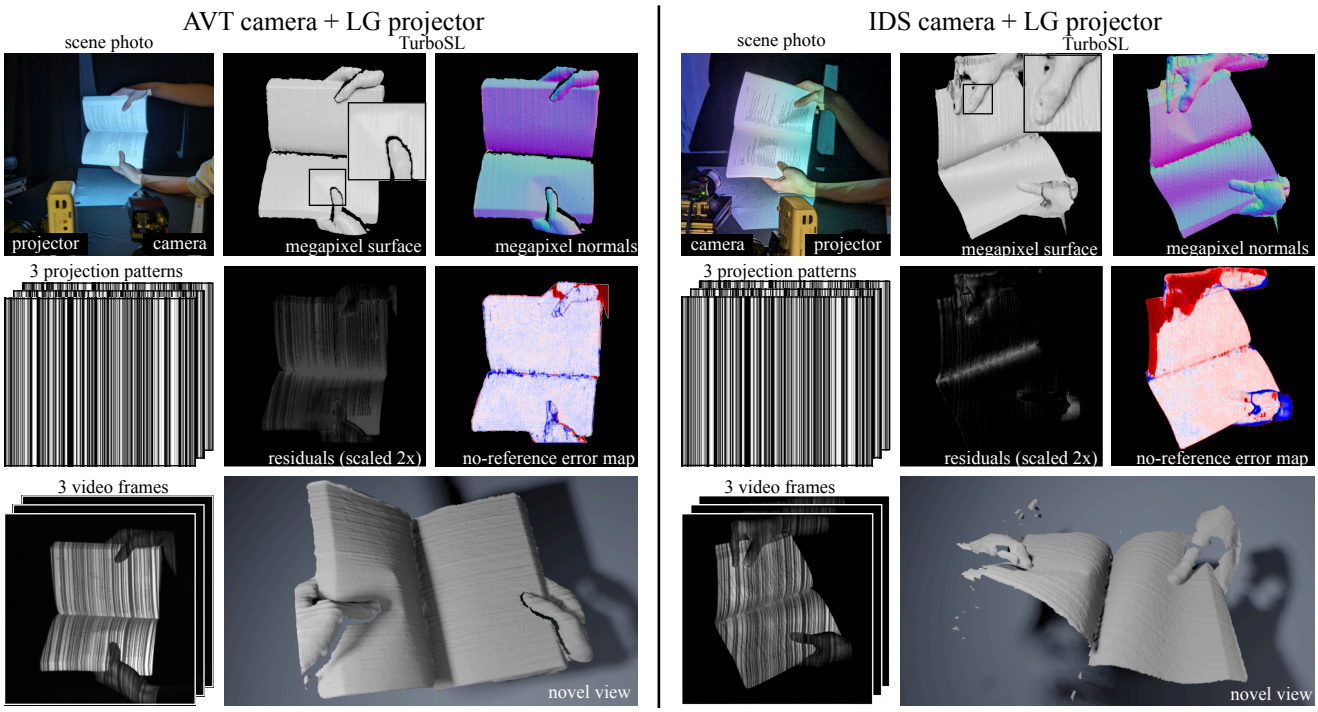

Figure S3: A book captured using two different imaging systems. The left and right panels show the results using the AVT and IDS cameras, respectively.

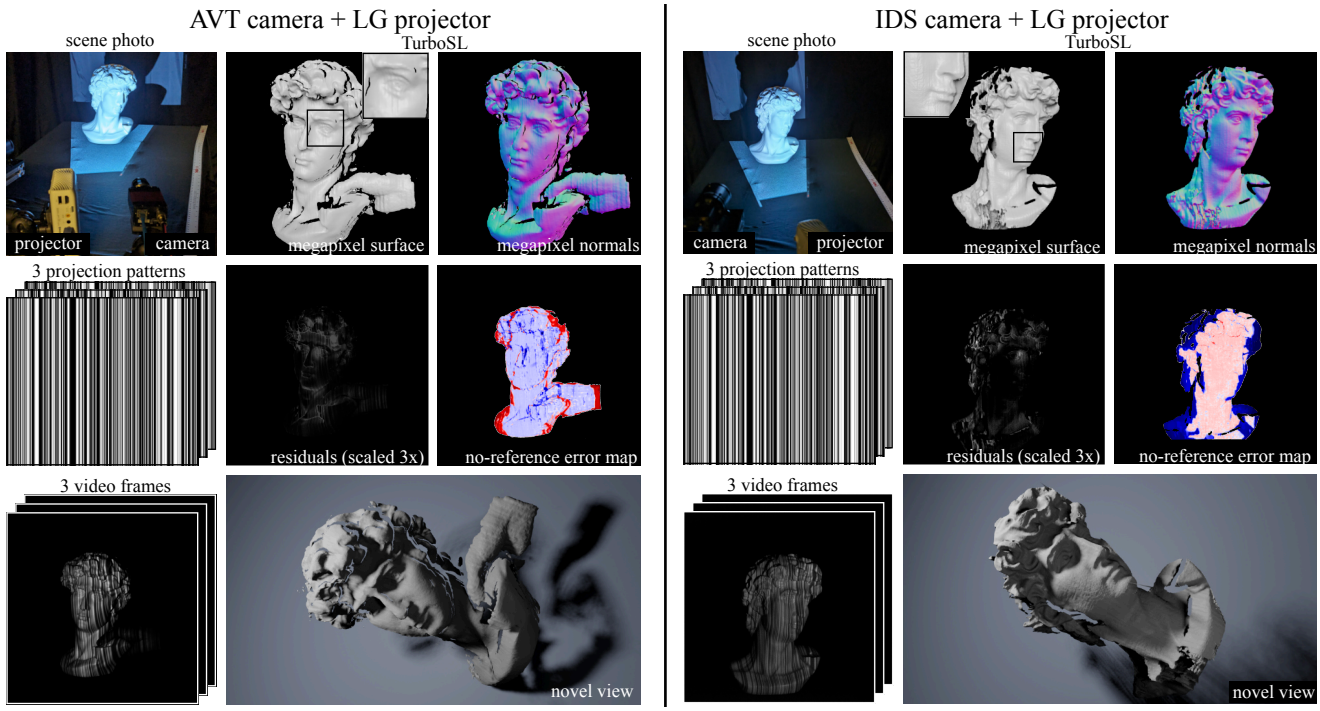

Figure S4: The David scene captured using two different imaging systems. The left and right panels show the results using the AVT and IDS cameras, respectively.

B. Additional Experimental Results

B.1. Captured Dataset

Here we provide the experimental details for captured scenes used throughout the main paper.

Table S1: *Experimental details for captured scenes.*

scene name	figure	camera	exposure (s)	SNR (dB)	pattern family	max frequency	num. of patterns
book	1	AVT	1/30	N/A	a la carte	128	3
David	3, 4	IDS	1/60	23.89	multiple	multiple	multiple
wedge	6	IDS	1/60	27.37	a la carte	128	3
statue	7	IDS	1/30	25.10	a la carte	32	3
plush toy	7	IDS	1/30	28.99	a la carte	32	3
fabric animal	7	IDS	1/30	30.02	a la carte	64	4
fruit bowl	7	IDS	1/30	28.50	a la carte	64	3

For the David scene, we used 4 different sets of patterns as indicated in Figure 3 of the main paper. For Figure 4, we used 3 a la carte patterns with maximum frequency 128.

B.2. Using the TurboSL Decoder with Different SL Patterns

In this section, we explore the performance of the TurboSL decoder across four distinct dimensions: pattern family, number of patterns, maximum frequency of the patterns, and image SNR. Figures S5, S6, and S7 show results for the David scene across these dimensions. Figures S8, S9, and S10 show similar results for the Fruit scene. We used 1/60s exposure time for the David scene (SNR of 23.89 dB) and 1/30s exposure time for the Fruit scene (SNR of 28.50 dB) unless otherwise stated.

Figures S5 and S8 compare the TurboSL decoder with a state-of-the-art pixelwise decoder that uses zero-mean normalized cross-correlation (ZNCC) between the observation vector and the set of code vectors [7]. Three different families of encoding patterns are considered: micro-phase shifting (MPS), which employs narrow-band high frequency sinusoidal patterns [5]; Hamiltonian patterns which are constructed to maximize the coding curve and minimize the mean depth error [4]; and a la carte patterns that are optimized to maximize the number of pixels with zero disparity error [7]. We use the original set of Hamiltonian patterns that do not have any frequency constraints, but for a la carte and MPS, we show the frequency that exhibits the best performance.

Tables S2 to S7 provide additional quantitative comparisons between the pixelwise decoder and the TurboSL decoder on three main error metrics: the mean disparity error (measured in pixels); top-0 percentage, defined as the percentage of pixels reconstructed with sub-pixel accuracy; and the percentage of outliers in the reconstruction as defined by the percentage of pixels that have disparity error of greater than 10 pixels. These error metrics are all computed on the depth map pixels falling within the inlier mask, which is computed using bidirectional rendering and the no-reference error proxy. Refer to Section C.2 for more details on how the inlier masks are estimated.

TurboSL decoding vs. pixelwise decoding. The TurboSL decoder outperforms pixelwise decoding across different pattern families and different pattern-agnostic decoders (Figures S5 and S8). This suggests that the TurboSL decoder is not limited to being used with any specific pattern, but is a generic, pattern-agnostic decoder. The TurboSL decoder is particularly effective at reconstructing highly accurate surfaces when combined with a la carte patterns. This can be attributed to the zero-disparity-error nature of a la carte patterns, and the denoising bias of neural networks. From here on, we refer to TurboSL as the TurboSL decoder paired with a la carte patterns.

TurboSL decoder performance with additional patterns. In contrast to pixelwise decoders whose performance noticeably improves by increasing the number of patterns, the improvement in TurboSL's performance with an increasing number of patterns is small. This suggests that TurboSL is very effective at decoding SL images without requiring a large number of patterns, and it enables surface reconstruction with as few as 3 patterns.

Effect of frequency and SNR. We evaluate the effect of pattern frequency on TurboSL performance in Figures S6 and S9. We observe that high-frequency a la carte patterns lead to highly detailed and accurate surface reconstructions. Using low frequency patterns still results in reasonable surface reconstructions, though some details are smoothed out.

We evaluate the effect of SNR in Figures S7 and S10. Lowering the SNR of captured images (by reducing the exposure time) does not lead to a significant reduction in accuracy, but it does result in outliers in the reconstruction (highlighted by the dark colors in the disparity error maps). These regions, can be identified and removed from the output surface reconstructions using the error proxy. Section C provides more details about the error proxy.

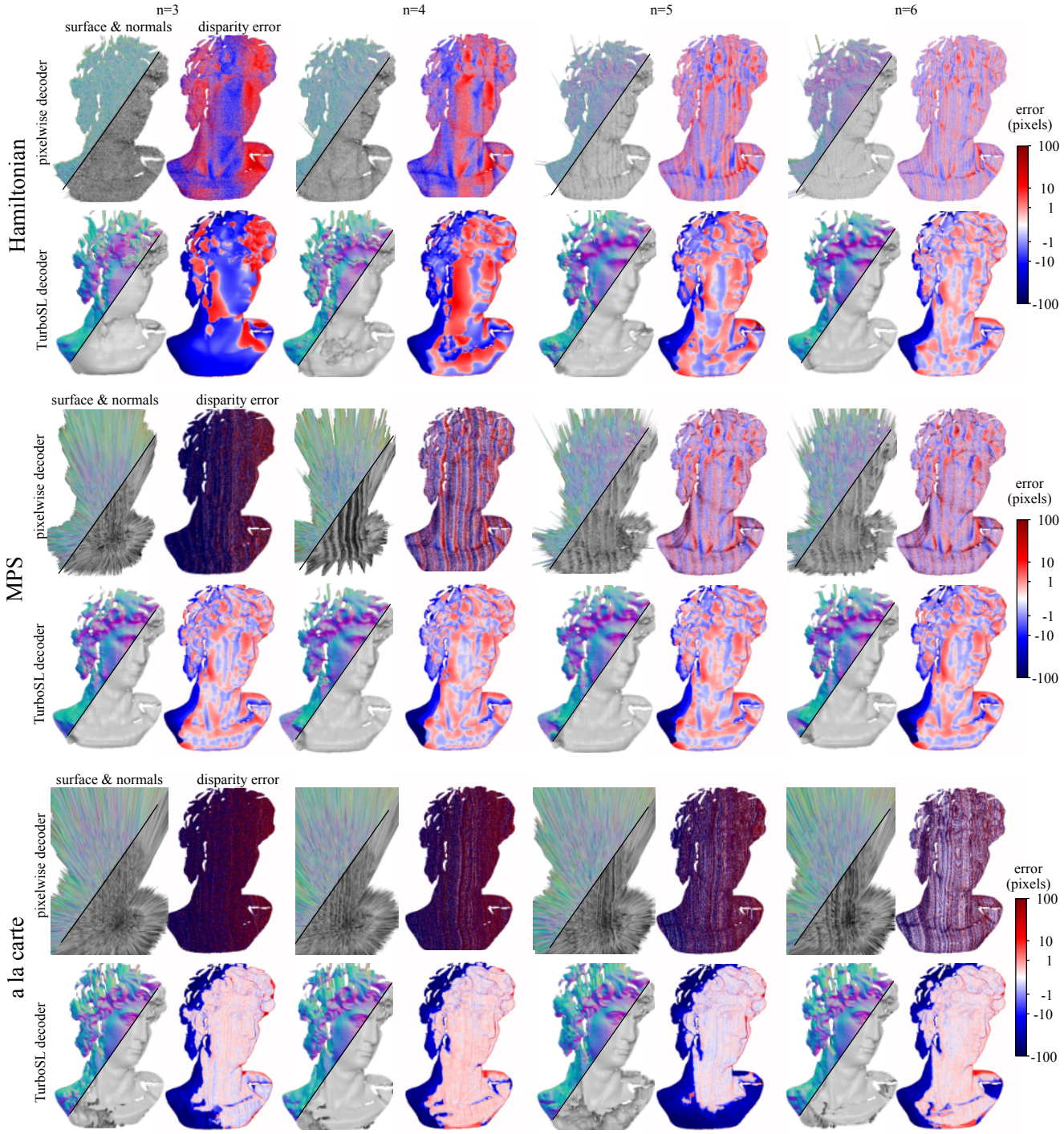

Figure S5: Pixelwise decoding vs. TurboSL decoding for different families of patterns and different numbers of patterns. **Top:** A la carte patterns with maximum frequency of 8. **Middle:** MPS patterns with maximum frequency of 128. **Bottom:** Hamiltonian patterns. The TurboSL decoder consistently outperforms the pixelwise decoder in reconstructing the surfaces.

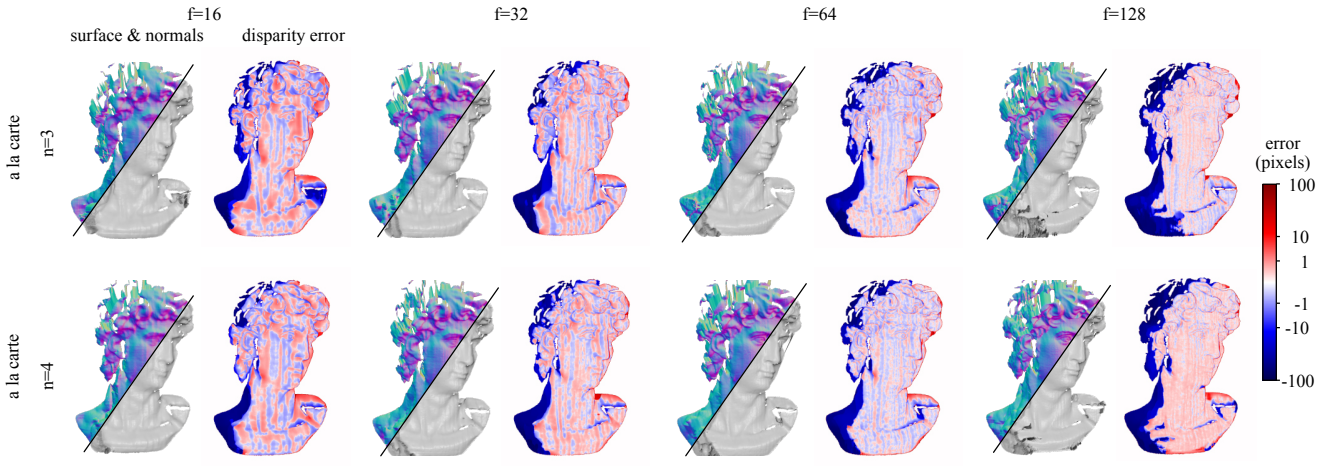

Figure S6: TurboSL performance using a la carte patterns with different maximum frequencies. Reconstruction using 3 patterns (top) and 4 patterns (bottom).

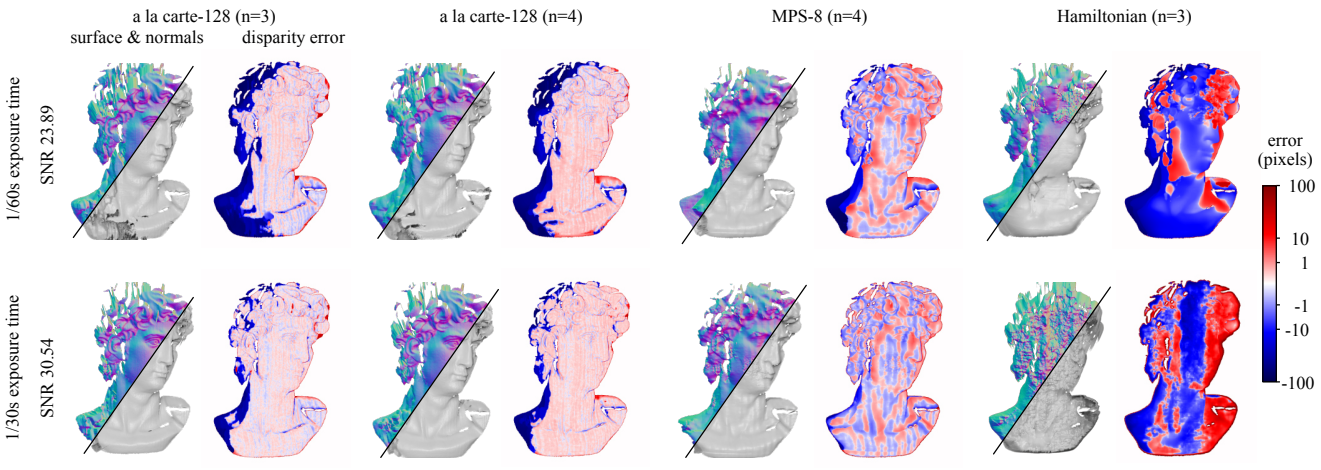

Figure S7: TurboSL performance with different image SNRs. **Top:** Scene images are captured with 1/60s exposure time leading to an image SNR of 23.89 dB. **Bottom:** Scene images are captured with 1/30s exposure time leading to an image SNR of 30.54 dB.

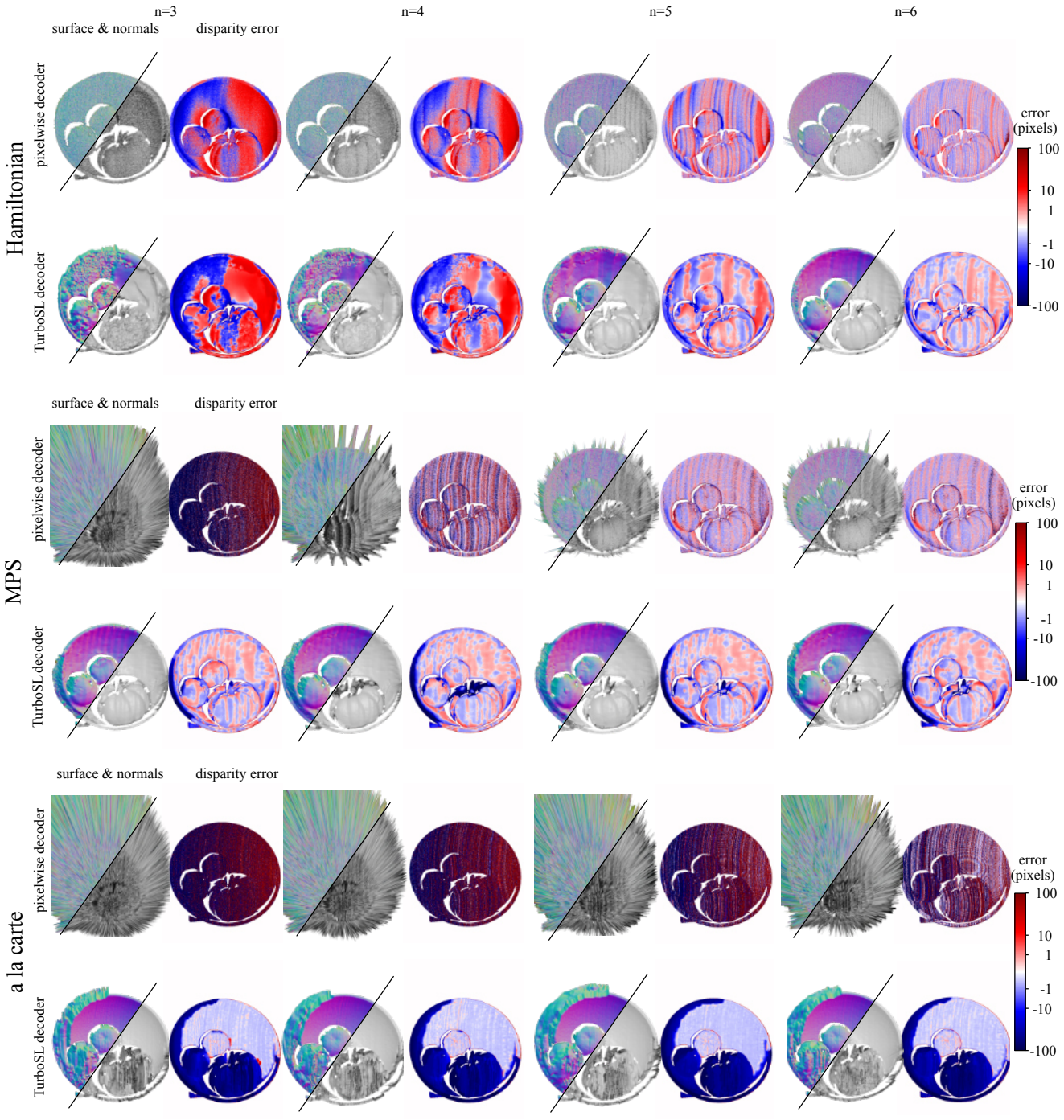

Figure S8: Pixelwise decoding vs. TurboSL decoding for different families of patterns and different numbers of patterns. **Top:** A la carte patterns with maximum frequency of 128; **Middle:** MPS patterns with maximum frequency of 8; **Bottom:** Hamiltonian patterns. The TurbosSL decoder consistently outperforms the pixelwise decoder in reconstructing the surfaces.

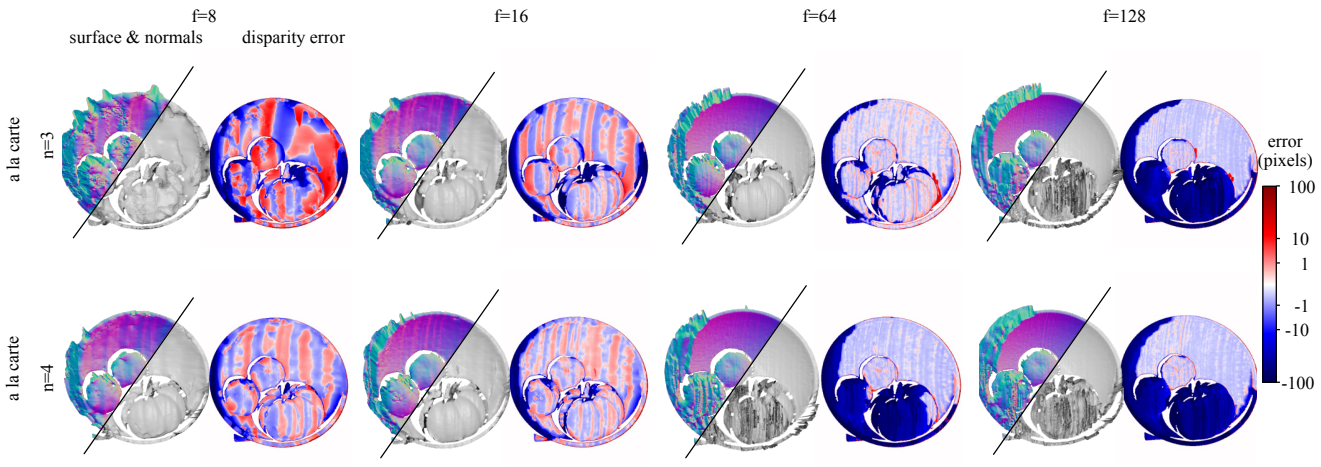

Figure S9: TurboSL performance across a la carte patterns with different maximum frequencies. Reconstruction using 3 patterns (top) and 4 patterns (bottom).

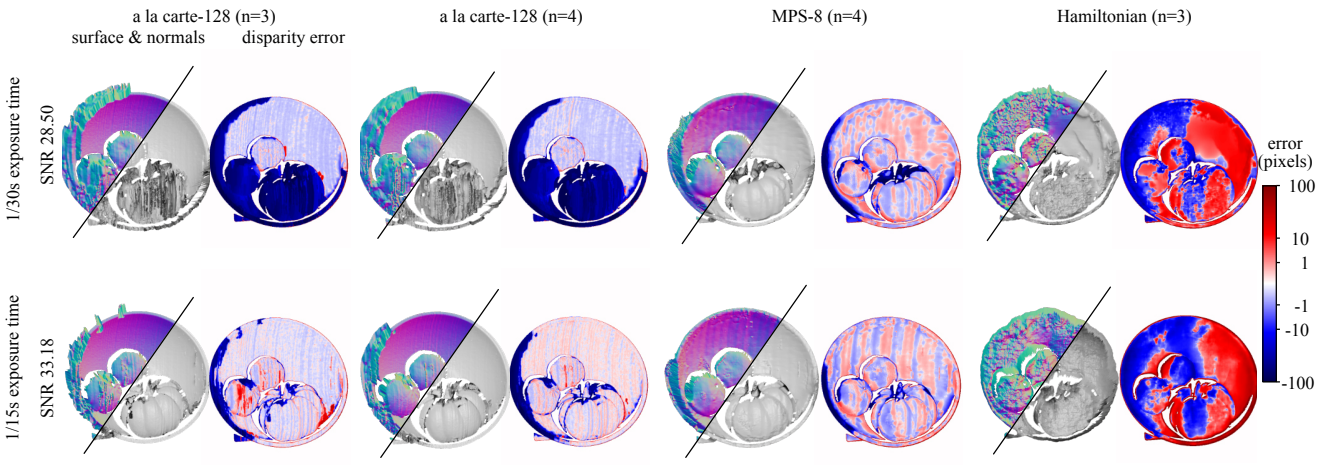

Figure S10: TurboSL performance on two different image SNRs. **Top:** Scene images are captured with 1/60s exposure time leading to an image SNR of 28.50 dB. **Bottom:** Scene images are captured with 1/30s exposure time leading to an image SNR of 33.18 dB.

Table S2: *Quantitative comparison of the pixelwise decoder and the TurboSL decoder for the David scene with a la carte-128 patterns*

Method	mean disparity error (pixel)↓				top-0 percentage↑				outlier percentage ↓			
	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6
pixelwise	200.98	175.05	149.68	95.53	1.18	7.66	19.18	50.03	96.53	89.82	78.57	47.86
TurboSL	0.17	0.18	0.25	0.20	98.24	98.61	97.40	97.99	0.02	0.01	0.04	0.02

Table S3: *Quantitative comparison of the pixelwise decoder and the TurboSL decoder for the David scene with MPS-8 patterns*

Method	mean disparity error (pixel)↓				top-0 percentage↑				outlier percentage ↓			
	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6
pixelwise	188.65	71.20	7.72	6.86	7.77	35.00	51.12	54.12	84.14	37.80	6.62	5.82
TurboSL	0.72	0.69	0.78	0.84	75.32	78.19	74.66	71.13	0.00	0.00	0.01	0.01

Table S4: *Quantitative comparison of the pixelwise decoder and the TurboSL decoder for the David scene with Hamiltonian patterns*

Method	mean disparity error (pixel)↓				top-0 percentage↑				outlier percentage ↓			
	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6
pixelwise	6.52	3.77	1.75	1.02	10.08	18.46	40.13	63.55	21.09	5.25	0.54	0.04
TurboSL	5.66	3.74	1.33	0.65	8.94	21.10	54.36	80.97	10.57	5.77	0.07	0.01

Table S5: *Quantitative comparison of the pixelwise decoder and the TurboSL decoder for the fruit bowl scene with a la carte-128 patterns*

Method	mean disparity error (pixel)↓				top-0 percentage↑				outlier percentage ↓			
	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6
pixelwise	214.25	187.45	171.78	112.63	1.64	5.72	14.59	41.20	95.57	90.98	83.40	56.43
TurboSL	0.15	0.23	0.20	0.17	99.32	99.49	99.42	99.70	0.04	0.04	0.01	0.08

Table S6: *Quantitative comparison of the pixelwise decoder and the TurboSL decoder for the fruit bowl scene with MPS-8 patterns*

Method	mean disparity error (pixel)↓				top-0 percentage↑				outlier percentage ↓			
	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6
pixelwise	193.32	60.85	5.73	4.51	9.76	47.19	66.40	68.47	84.45	32.12	4.97	3.66
TurboSL	0.51	0.48	0.56	0.61	89.14	90.82	88.40	87.57	0.00	0.01	0.02	0.02

Table S7: *Quantitative comparison of the pixelwise decoder and the TurboSL decoder for the fruit bowl scene with Hamiltonian patterns*

Method	mean disparity error (pixel)↓				top-0 percentage↑				outlier percentage ↓			
	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6	n = 3	n = 4	n = 5	n = 6
pixelwise	9.69	4.81	1.77	0.93	7.53	18.23	43.81	69.05	34.84	12.97	0.79	0.06
TurboSL	9.70	4.37	1.26	0.49	6.93	23.18	59.55	91.26	38.36	12.89	0.18	0.01

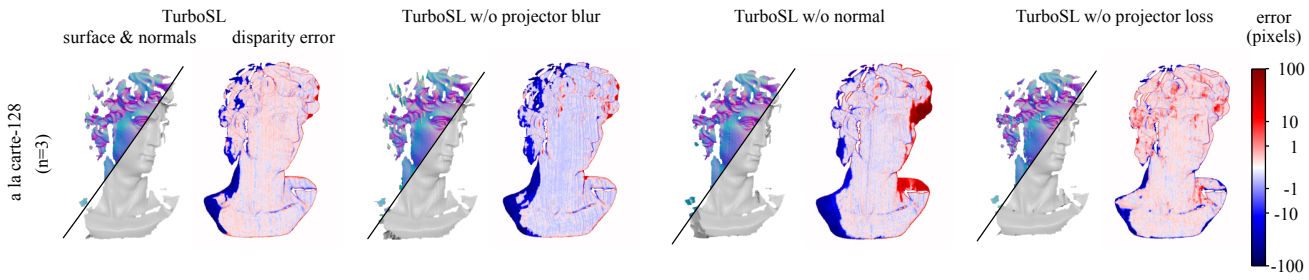

Figure S11: TurboSL ablation study. **Column 1:** TurboSL reconstruction and disparity error; **Column 2:** TurboSL without optimizing for the projector blur kernel; **Column 3:** TurboSL without modelling the normal in the image formation model; **Column 4:** TurboSL without using the projector loss. Omitting any of these components leads to higher disparity error in the reconstructions.

B.3. Ablation Study

We investigate the role of each component in the TurboSL decoder's performance through a set of ablation experiments. Figure 7 of the main paper shows results that progressively add each of the method's main components to the base model. Here, we compare the results of TurboSL with and without each of its main components. Figure S11 shows the reconstructed surface and its disparity error compared to the ground truth. As with all figures in the main paper and the supplemental document, we use a non-linear color bar to highlight very small disparity errors. Comparing the error maps demonstrates the role of each component in the final performance of TurboSL.

B.4. Dynamic Scene Reconstruction

Here we describe TurboSL reconstructions of two dynamic scenes captured at 30 fps. Figure S12 depicts a rigid, handheld object being rotated during the capture (row 1), and a (non-rigid) book being opened (row 2, also appeared in Figure 1 of the main paper). The projector operates at 30 fps and cyclically projects 3 a la carte patterns with maximum frequency 128. During reconstruction, a sliding window of 3 consecutive video frames (each illuminated by a distinct pattern) are grouped as input to TurboSL, resulting in an output rate of 30 fps. The underlying assumption is that across 3 consecutive frames of a video capture, the geometry in the scene remains sufficiently stationary to enable accurate reconstruction. Refer to the supplemental video for more visualizations.

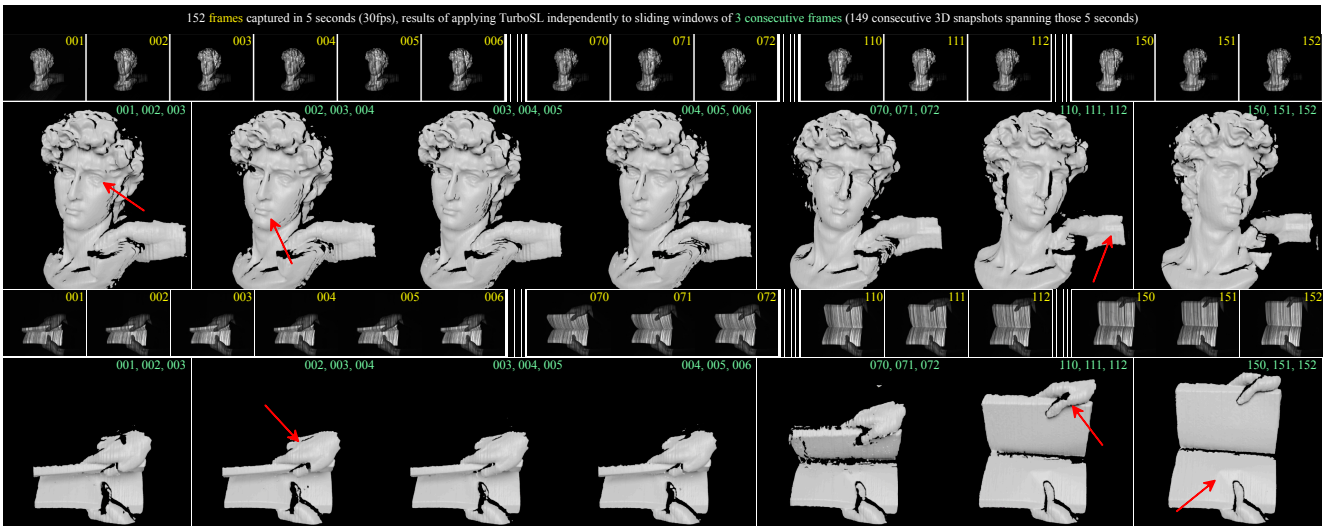

Figure S12: Dynamic reconstruction using TurboSL. **Row 1:** Handheld rigid object rotated during capture. **Row 2:** Opening of a non-rigid book. All 152 video frames are captured over 5 seconds at 30 fps. Three different a la carte patterns (frequency 128) are projected in a cyclic fashion. A sliding window of 3 video frames (e.g. 001, 002, 003) is used as input to TurboSL for reconstruction. Quality is consistent with static reconstructions; note the details highlighted by the red arrows (e.g., eyes, lips, wrist veins, book creases).

C. No-Reference Error Proxy via Back-to-Front Rendering

The back-to-front rendering scheme provides a second reconstruction that can be used to compute an error proxy for the front-to-back reconstruction. We show additional results demonstrating the correlation between the actual disparity error and the error proxy, and we show how the error proxy can be used to remove low-confidence regions in the final reconstruction.

C.1. Correlation Between the Error Proxy and Reconstruction Error

Figures S13 and S14 show the disparity errors for back-to-front (column 2) and front-to-back (column 4) renderings for two different scenes: David and Fruit. We define the error proxy as the difference between the front-to-back and back-to-front reconstructed disparities. Intuitively, we expect that for accurately predicted pixels, the outputs of both rendering schemes should agree (i.e., they predict the same disparity). We show that this intuition holds, especially for a la carte patterns for which the error proxy and the actual disparity error are highly correlated.

Correlation between the error proxy and the actual disparity error. In Figures S13 and S14 we show scatter plots of the error proxy versus the disparity error (calculated using the ground truth acquisition). Both front-to-back and back-to-front renderings show a strong correlation between the reconstruction error and the error proxy, and this correlation is particularly strong for a la carte patterns, which produce the most accurate results. The error proxy can be used to reliably identify erroneous pixels by setting an error threshold. For MPS and Hamiltonian patterns, the correlation between the error proxy and actual disparity error is noticeably weaker compared to using a la carte patterns; however, the disparity predictions themselves are also significantly worse. Here, the error proxy may still be useful for outlier rejection, i.e., by identifying regions in the error proxy with the largest magnitude.

Horizontal and vertical lines in a la carte correlations. We notice horizontal and vertical structures in scatter plots of the error proxy versus disparity error shown in Figures S13 and S14. The vertical structures correspond to pixels whose actual disparity error is close to zero; in this case the disparity error is small but the front-to-back and back-to-front disparity predictions disagree. Hence, the vertical structures correspond to false negatives when using the error proxy to identify accurately predicted disparities. The horizontal structures comprise a very sparse set of pixels that are actually reconstructed incorrectly, but the back-to-front and front-to-back disparity predictions show close agreement. Thus, these points correspond to false positives when using the error proxy to identify accurately predicted disparities. Overall, the horizontal and vertical structures contain a much sparser set of pixels compared to the dominant diagonal structures, which indicate the strong correlation between the error proxy and the actual disparity error.

Bias in the back-to-front and front-to-back disparity predictions. Comparing the correlation plots and the signed disparity errors across different patterns shows that each reconstruction tends to be biased towards one side of the ground truth surface. The back-to-front predictions are biased towards the back of the ground truth surface (since the disparity error is mostly positive), and the front-to-back predictions are biased towards the front side of the ground truth surface. *This inherent bias distinguishes these two rendering schemes from two random trials of the same optimization.*

C.2. Filtering Disparity Predictions Using the Error Proxy

As described previously, the error proxy is a strong indicator of the actual reconstruction errors. Therefore, it can be used to identify pixels with accurate disparity predictions and to remove uncertain regions from the reconstructed surface. Figure S15 shows one example where the error proxy is used to define an inlier mask for the *confident pixels*, and that mask is used to remove the uncertain regions.

We compute the inlier mask by evaluating the median error proxy value m within the mask used for the mask loss. The mask is given by thresholding the error proxy values to those that are less than $100m$ for a la carte patterns and less than $20m$ for other patterns. The threshold is calibrated to keep inlier predictions with a disparity error of less than 10 pixels.

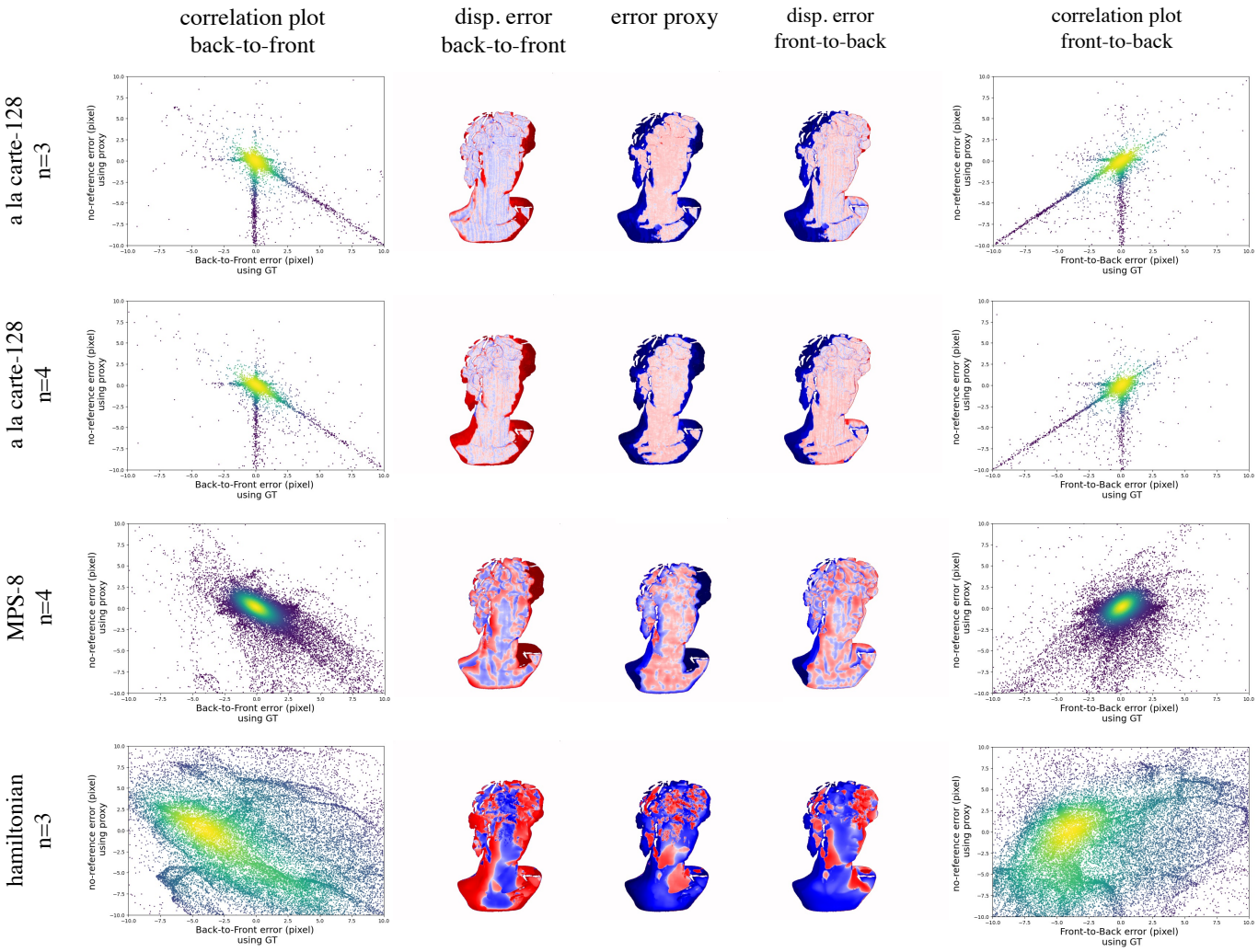

Figure S13: Front-to-back and back-to-front reconstruction error for the David scene. **Columns 1 and 5:** The scatter plots show the relationship between the error proxy (i.e., the difference between disparities predicted with front-to-back and back-to-front rendering) and the ground truth disparity error. **Columns 2 and 4:** Ground truth disparity errors using front-to-back and back-to-front rendering. **Column 3:** The error proxy. The error maps use the same non-linear color scale as all the figures in the main paper and the supplement.

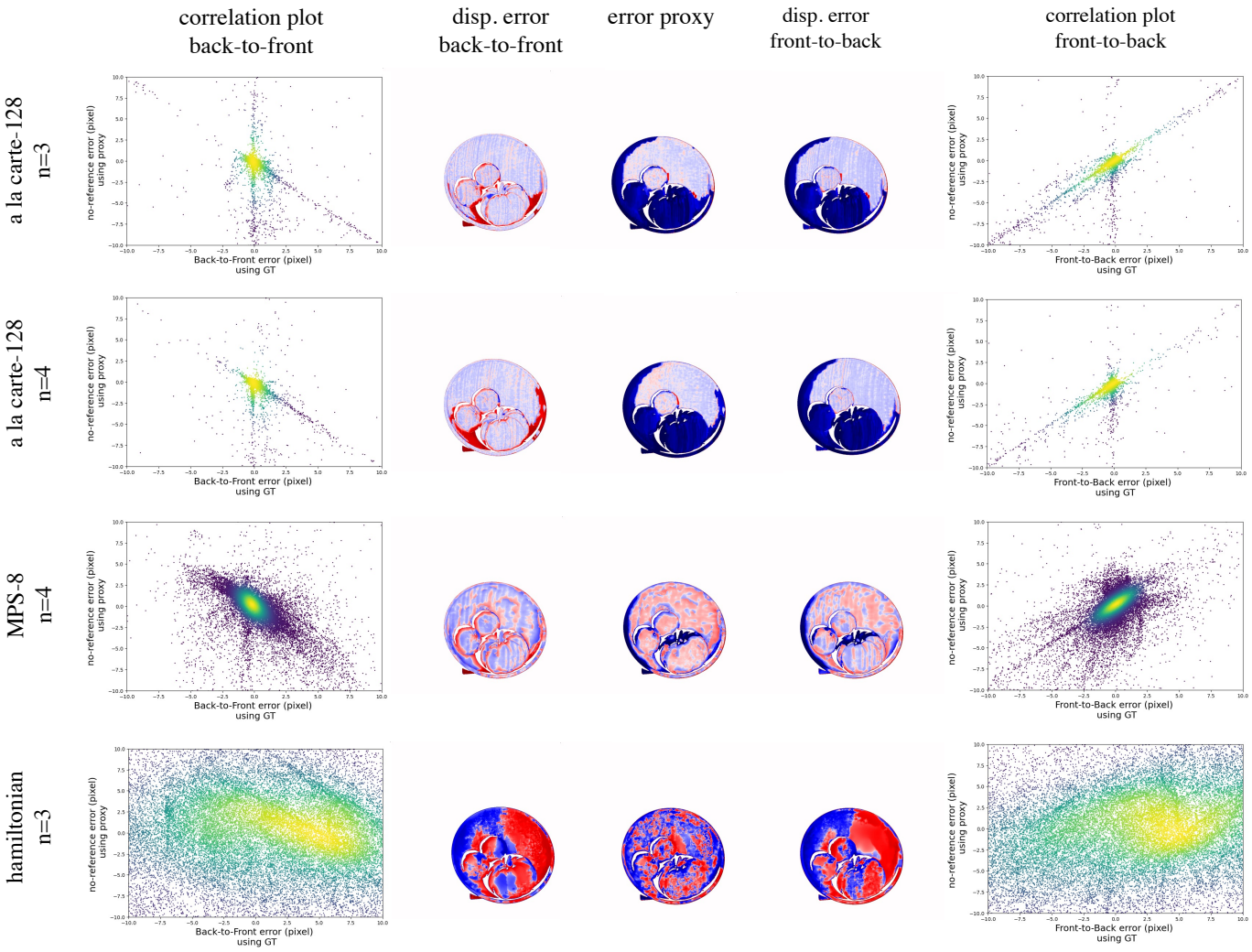

Figure S14: Front-to-back and back-to-front reconstruction error for the Fruit scene. **Columns 1 and 5:** **Columns 1 and 5:** The scatter plots show the relationship between the error proxy (i.e., the difference between disparities predicted with front-to-back and back-to-front rendering) and the ground truth disparity error. **Columns 2 and 4:** Ground truth disparity errors using front-to-back and back-to-front rendering. **Column 3:** The error proxy. The error maps use the same non-linear color scale as all the figures in the main paper and the supplement.

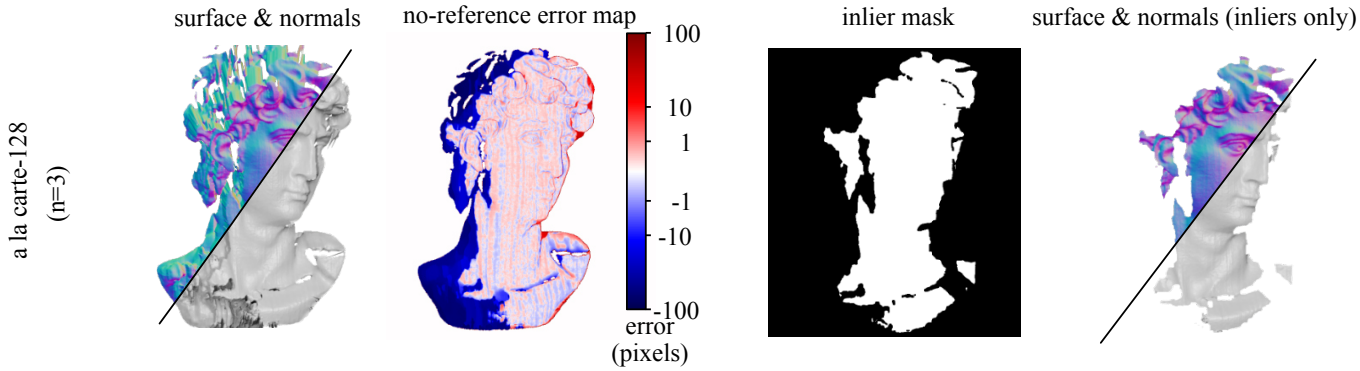

Figure S15: Using the error proxy to filter out reconstruction errors. **Column 1:** The surface reconstruction from front-to-back rendering. **Column 2:** The error proxy estimated by computing the signed difference between the front-to-back and back-to-front disparity predictions. **Column 3:** The inlier mask estimated from the error proxy by setting a threshold based on the median error proxy value. **Column 4:** The filtered reconstructed surface after removing outliers identified with the error proxy.

References

[1] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. In *CVPR*, 2022.

[2] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *Proceedings of the 37th International Conference on Machine Learning*, pages 3789–3799, 2020.

[3] Yuan-Chen Guo. Instant neural surface reconstruction, 2022. https://github.com/bennyguo/instant-nsr-pl.

[4] Mohit Gupta and Nikhil Nakhate. A geometric perspective on structured light coding. In *ECCV*, 2018.

[5] Mohit Gupta and Shree K Nayar. Micro phase shifting. In *CVPR*, 2012.

[6] Ruilong Li, Matthew Tancik, and Angjoo Kanazawa. NerfAcc: A general NeRF acceleration toolbox. *arXiv preprint arXiv:2210.04847*, 2022.

[7] Parsa Mirdehghan, Wenzheng Chen, and Kiriakos N Kutulakos. Optimal structured light a la carte. In *CVPR*, 2018.

[8] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):1–15, 2022.

[9] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *NeurIPS*, 2021.