Shape, Albedo, and Illumination from a Single Image of an Unknown Object Supplementary Material

Jonathan T. Barron and Jitendra Malik
UC Berkeley
{barron, malik}@eecs.berkeley.edu

1. Gradient Norm

We will detail how to calculate $\|\nabla A\|$ and its analytical derivative efficiently. The gradient norm of log-albedo is:

$$\|\nabla A\| = \sqrt{\frac{\partial A}{\partial x}^2 + \frac{\partial A}{\partial y}^2} \quad (1)$$

We will discretely approximate this using filter output. We would like to use 3×3 filters so that gradient estimates are located on pixels, rather than between pixels. Naively we would do this using sobel filters, but that formulation has the property that $\|\nabla A\|_{x,y}$ is not a function of $A_{x,y}$, because of the zeros in the filters being used. This often leads to undesirable artifacts in the output, where single isolated pixels of A are wrong. We will therefore use this formulation:

$$\|\nabla A\| = \sqrt{(A * h_2^x)^2 + (A * h_2^y)^2} * h_2^b \quad (2)$$

$$h_2^x = \frac{1}{2}\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}, \quad h_2^y = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \quad h_2^b = \frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad (3)$$

This operator is still a 3×3 operation on A, but responds more evenly to variation in A.

When computing $\|\nabla A\|$, we compute some intermediate "images", which are held onto until after the loss functions on $\|\nabla A\|$ have been calculated, at which point they are used to backpropagate the gradient of the loss function using the chain rule.

$$G^x = A * h_2^x, \qquad G^y = A * h_2^y \quad (4)$$

$$G^n = \sqrt{(G^x)^2 + (G^y)^2} \quad (5)$$

$$\|\nabla A\| = G^n * h_2^b \quad (6)$$

Given $f(\|\nabla A\|)$ and $\nabla_{\|\nabla A\|} f$, a loss function and the gradient of that loss function with respect to $\|\nabla A\|$, here is how calculate $\nabla_A f$, the gradient of the loss with respect to A:

$$D^n = \frac{(\nabla_{\|\nabla A\|} f) \star h_2^b}{G^n} \quad (7)$$

$$\nabla_A f = (G^x \times D^n) \star h_2^x + (G^y \times D^n) \star h_2^y \quad (8)$$

$\times$ is component-wise multiplication of two images, $/$ is component-wise division, $*$ is convolution and $\star$ is cross-correlation.

2. Quadratic Entropy

Here we will detail a novel method for calculating $V(\mathbf{x}, \sigma)$, the information potential of the elements of $\mathbf{x}$ under the quadratic entropy measure introduced in [6] and popularized in [3]. The entropy of $\mathbf{x}$ is $-\log(V(\mathbf{x}, \sigma)$, which is the quantity we actually minimize because it's in the same "units" as our priors on likelihood.

Let $\mathbf{x}$ be a vector, N is the length of $\mathbf{x}$, and σ is the bandwidth parameter (the width of the Gaussian bump around each element of $\mathbf{x}$). Then the quadratic entropy of $\mathbf{x}$ under the Parzen window defined by $\mathbf{x}$ and σ is defined as:

$$V(\mathbf{x}, \sigma) = \frac{1}{Z}\sum_{i=1}^{N}\sum_{j=1}^{N} \exp\left(-\frac{(\mathbf{x}_i - \mathbf{x}_j)^2}{4\sigma^2}\right) \quad (9)$$

$$Z = N^2\sqrt{4\pi\sigma^2} \quad (10)$$

Our first insight is that this can be re-expressed as a function on a histogram of $\mathbf{x}$. Let B = bin width, K = number of bins, and $\mathbf{n}_i$ = count of $\mathbf{x}$ in bin i. Then:

$$V(\mathbf{n}, \sigma) = \sum_{i=1}^{K} \mathbf{n}_i \sum_{j=1}^{K} \frac{\mathbf{n}_j}{Z} \exp\left(-\frac{B^2(i-j)^2}{4\sigma^2}\right) \quad (11)$$

Provided many elements of $\mathbf{x}$ fall into the same bin, this may be easier to compute than the naive representation, but it's still quadratic in its worst-case computational complexity. Our second insight is that this can be expressed as a convolution of $\mathbf{n}$ with a small Gaussian filter. Let $\mathbf{g}$ be a Gaussian filter of length $2J+1$ centered at $J+1$:

$$\mathbf{g}_j = \frac{1}{Z}\exp\left(-\frac{B^2 j^2}{4\sigma^2}\right) \quad (12)$$

Where j is distance from the center. With this, we can rewrite $V(\mathbf{n})$ as follows:

$$V(\mathbf{n}, \sigma) = \mathbf{n}^{\mathrm{T}}(\mathbf{n} * \mathbf{g}) \quad (13)$$

Where $*$ is convolution. This quantity is extremely efficient to compute, provided that the lengths of $\mathbf{n}$ and $\mathbf{g}$ are small, which is true provided that the range of $\mathbf{x}$ is not much larger than σ, which is generally true in practice.

Crucially, this formulation also allows us to easily compute the gradient of $V(\mathbf{n}, \sigma)$ with respect to $\mathbf{n}$:

$$\nabla V(\mathbf{n}, \sigma) = 2(\mathbf{n} * \mathbf{g}) \tag{14}$$

Simple histogramming is a non-smooth operation, making it impossible to backpropagate this gradient signal onto $\mathbf{x}$. However, if instead of standard histogramming with use a smooth approximation like linear interpolation, then the gradient with respect to $\mathbf{x}$ can indeed be calculated analytically. Intuitively, this is done by having each x vote for its two nearest bins, rather than simply vote for its one nearest bin. With this approximation, the "bins" in our histogram can be thought of as fenceposts, where x is assigned to the closest fenceposts above and below it.

More concretely: Let R_L and R_U define the bounds on the range of the bins, and $R_L - R_U = BK$, and $R_U = \min(\mathbf{x})$ and $R_L = \max(\mathbf{x})$. Consider datapoint x. The fencepost it will be assigned to are b_L and b_U, where b_L is the largest fencepost below it, and b_U is the smallest fencepost above it:

$$b_L = \lfloor (x - R_L)/B \rfloor, \quad b_U = b_L + 1 \tag{15}$$

x will be assigned to those bins according to these weights:

$$w_L = (x - b_L)/B, \quad w_U = 1 - w_L \tag{16}$$

When adding x to the histogram, we just add these two weights to the appropriate bins:

$$\mathbf{n}_L = \mathbf{n}_L + w_L, \quad \mathbf{n}_U = \mathbf{n}_U + w_U \tag{17}$$

The partial derivatives of the histogram with respect to x are simple:

$$\frac{\partial \mathbf{n}_L}{\partial x} = -\frac{1}{B}, \quad \frac{\partial \mathbf{n}_U}{\partial x} = \frac{1}{B} \tag{18}$$

With this, we can construct the Jacobian $J_{\mathbf{n}}$ of $\mathbf{n}$ with respect to $\mathbf{x}$, which is a K by N sparse matrix. With this, we can calculate the gradient of V with respect to $\mathbf{x}$:

$$\nabla V(\mathbf{x}) \approx J_{\mathbf{n}}^{\mathrm{T}} \nabla V(\mathbf{n}) \tag{19}$$

This approximation to quadratic entropy is, in practice, extremely efficient and extremely accurate. Other techniques exist for computing approximations to this quantity, most notably the fast Gauss transform and the improved fast Gauss transform. Also, $V(\mathbf{x}, \sigma)$ could be computed exactly using the naive formulation in Equation 9. The naive formulation is completely intractable, as the computation complexity if $O(N^2)$. Our algorithm, and the FGT-based algorithms are both $O(N)$. However, there is no efficient way to compute $\nabla V(\mathbf{x}, \sigma)$ using the FGT-based algorithms, which makes those algorithms impossible to use in our gradient-based optimization scheme. Our approximation is usually within 0.01% of the true entropy, which is similar to the accuracy obtained using the fast Gauss transform or the improved fast Gauss transform. In addition to allowing for $\nabla V(\mathbf{x}, \sigma)$ to be approximated extremely efficiently, our algorithm is 10 or 100 times faster than the FGT-based algorithms.

3. Mean Curvature

We will detail how to calculate $H(Z)$ and its analytical derivative efficiently. Mean curvature on a surface is a function of the first and second partial derivatives of that surface.

$$H(Z) = \frac{\left(1 + Z_x^2\right) Z_{yy} - 2 Z_x Z_y Z_{xy} + \left(1 + Z_y^2\right) Z_{xx}}{2\left(1 + Z_x^2 + Z_y^2\right)^{3/2}} \tag{20}$$

To calculate this discretely, we will first approximate the partial derivatives using filter convolutions.

$$Z_x = Z * h_3^x, \quad Z_y = Z * h_3^y \tag{21}$$

$$Z_{xx} = Z * h_3^{xx}, \quad Z_{yy} = Z * h_3^{yy}, \quad Z_{xy} = Z * h_3^{xy} \tag{22}$$

$$h_3^x = \tfrac{1}{8}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad h_3^y = \tfrac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$h_3^{xy} = \tfrac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}, \quad h_3^{yy} = \tfrac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{bmatrix}, \quad h_3^{xx} = \tfrac{1}{4}\begin{bmatrix} 1 & -2 & 1 \\ 2 & -4 & 2 \\ 1 & -2 & 1 \end{bmatrix}$$

We then compute the following intermediate "images", and use them to compute $H(Z)$.

$$\begin{aligned} M &= \sqrt{1 + Z_x^2 + Z_y^2} \\ N &= (1 + Z_x^2) Z_{yy} - 2 Z_x Z_y Z_{xy} + (1 + Zy^2) Z_{xx} \\ D &= 2M^3 \\ H(Z) &= N/D \end{aligned} \tag{23}$$

When computing $H(Z)$, we also compute the following, which are held onto until after the loss functions on $H(Z)$ have been calculated, at which point they will be used to backpropagate the gradient of the loss function using the chain rule.

$$\begin{aligned} F_x &= 2(Z_x Z_{yy} - Z_{xy} Z_y) - \frac{3 Z_x N}{M^2} \\ F_y &= 2(Z_{xx} Z_y - Z_x Z_{xy}) - \frac{3 Z_y N}{M^2} \\ F_{xx} &= 1 + Z_y^2 \\ F_{yy} &= 1 + Z_x^2 \\ F_{xy} &= -2 Z_x Z_y \end{aligned} \tag{24}$$

Given $f(H(Z))$ and $\nabla_{H(Z)} f$, a loss function and the gradient of that loss function with respect to $H(Z)$, here is how calculate $\nabla_Z f$, the gradient of the loss with respect to Z.

$$B = \frac{\nabla_{H(Z)} f}{D} \tag{25}$$

$$\begin{aligned} \nabla_Z f &= (BF_x) \star h_3^x + (BF_y) \star h_3^y \\ &+ (BF_{xx}) \star h_3^{xx} + (BF_{yy}) \star h_3^{yy} + (BF_{xy}) \star h_3^{xy} \end{aligned} \tag{26}$$

Adjacent variables are component-wise multiplication of two images, / is component-wise division, $*$ is convolution and $\star$ is cross-correlation.

4. Linearization, Rendering, and Albedo

We will detail how to calculate $S(Z, L)$ and its analytical derivative efficiently, for the purpose of calculating A and backpropagating losses on A back onto Z . First, we convert Z into a set of surface normals:

$$N^x = \frac{Z * h_3^x}{B}, \quad N^y = \frac{Z * h_3^y}{B}, \quad N^z = \frac{1}{B} \tag{27}$$

$$B = \sqrt{1 + (Z * h_3^x)^2 + (Z * h_3^y)^2} \tag{28}$$

We also compute the following:

$$F_{11} = (1 - N^x \times N^x) \times N^z \tag{29}$$

$$F_{22} = (1 - N^y \times N^y) \times N^z \tag{30}$$

$$F_{13} = -(N^x \times N^z \times N^z) \tag{31}$$

$$F_{23} = -(N^y \times N^z \times N^z) \tag{32}$$

$$F_{12} = -(N^x \times N^y \times N^z) \tag{33}$$

Let us look at the surface normal at one pixel: $\mathbf{n}_i = [N_i^x, N_i^y, N_i^z]^{\mathrm{T}}$. Rendering that point with spherical harmonics is:

$$S(\mathbf{n}_i, L) = [\mathbf{n}_i; 1]^{\mathrm{T}} \mathrm{M} [\mathbf{n}_i; 1] \tag{34}$$

$$\mathrm{M} = \begin{bmatrix} c1L_9 & c1L_5 & c1L_8 & c2L_4 \\ c1L_5 & -c1L_9 & c1L_6 & c2L_2 \\ c1L_8 & c1L_6 & c3L_7 & c2L_3 \\ c2L_4 & c2L_2 & c2L_3 & c4L_1 - c5L_7 \end{bmatrix}$$

$$c1 = 0.429043 \quad c2 = 0.511664$$

$$c3 = 0.743125 \quad c4 = 0.886227 \quad c5 = 0.247708$$

Note that $S(\mathbf{n}_i, L)$ is the *log*-shading at pixel i, not the shading. This is different from the traditional usage of spherical harmonic illumination. Directly modeling log-shading makes optimization easier by guaranteeing that shading is greater than 0 without needing to clamp shading at 0, as is normally done. The gradient of the log-shading at this point with respect to the surface normal is:

$$B_i = \nabla_{\mathbf{n}_i} S(\mathbf{n}_i, L) = 2\mathbf{n}_i^{\mathrm{T}} \mathrm{M}[:, 1:3]$$

Where B is a three-channel image, where B^x is the gradient of S with respect to N^x, etc. Given the log-shading, we can infer what the log-albedo at this point must be:

$$A_i = I_i - S(\mathbf{n}_i, L) \tag{35}$$

After calculating $g(A)$ and $\nabla_A g(A)$, we can backpropagate the gradient onto Z as follows:

$$\begin{aligned} D_S &= -\nabla_A g(A) \\ D_x &= B^x \times F_{11} + B^y \times F_{12} + B^z \times F_{13} \\ D_y &= B^x \times F_{12} + B^y \times F_{22} + B^z \times F_{23} \\ \nabla_Z g(A) &= (D_S \times D_x) \star h_3^x + (D_S \times D_y) \star h_3^y \end{aligned} \tag{36}$$

$\times$ is component-wise multiplication of two images, / is component-wise division, $*$ is convolution and $\star$ is cross-correlation.

5. Unknown Illumination

When illumination is unknown, the optimization problem we must solve is:

$$\underset{Z}{\text{maximize}} \quad \log \left(P(Z) \sum_L P(A|Z, L) P(L) \right) \tag{37}$$

Rather than optimizing this marginal log-likelihood using Expectation-Maximization (which is intractably expensive due to the cost of doing a complete M-step), or directly optimize the complete log-likelihood with respect to Z and some L using gradient techniques (which works poorly due to local minima), we use the technique presented in [7] to directly optimize the expected complete log-likelihood using gradient-based techniques, in a manner that approximates doing EM:

$$\underset{Z}{\text{maximize}} \quad \log\left(P(Z)\right) + \sum_L P(L|A, Z) \log\left(P(A|Z, L)\right)$$

Or, in terms of our loss functions:

$$\underset{Z}{\text{minimize}} \quad f(Z) + \sum_{L_i} P(L_i|A, Z) g(I - S(Z, L_i))$$

First we must clarify what L means in these equations. Rather than building a sophisticated model of illuminations for $P(L)$, we simply assume that L is one of the illumination conditions in our training set. That is, $P(L)$ assigns all a uniform probability to all illumination conditions within our training set and a probability of 0 to any other illumination. This is accomplished by using the set $\{L_i\}$ (where each L_i is a training illumination) in the marginalization in our optimization. Illumination conditions in which the z component is negative are removed when constructing $\{L_i\}$, as they are assumed to be incorrectly estimated by our

photometric stereo algorithm, as the lighting comes from the front in all of our images.

For one of these illumination conditions, by Bayes rule, we have:

$$P(L_i|A,Z) \propto P(A|L_i,Z)P(L_i) \quad (38)$$
$$\propto P(A|L_i,Z) \quad (39)$$

Which we calculate as follows:

$$P(L_i|A,Z) = \frac{\exp\left(-\gamma g(I - S(Z,L_i))\right)}{\sum_j \exp\left(-\gamma g(I - S(Z,L_j))\right)} \quad (40)$$

Where γ is a parameter that determines how smoothed this distribution is. A too-peaked distribution will cause optimization to quickly select a single illumination as the most probable one, and a too-smooth distribution will cause optimization to never focus in on any particular illumination. We fit γ as a new hyper parameter, but it is actually just a scaling applied to the two pre-existing hyperparameters that govern our albedo priors: λ_s and λ_e.

With $P(L_i|A,Z)$, we can compute the expected complete log-likelihood and its gradient using a weighted combination of the losses and gradient of the losses for each illumination. We then simply optimize the expected complete log-likelihood using the coarse-to-fine L-BFGS optimization technique detailed in this paper. Once optimization is complete, we use our recovered shape $\hat{Z}$ to compute the expected illumination:

$$\hat{L} = \sum_{L_i} P(L_i|A,\hat{Z})L_i \quad (41)$$

And then use $\hat{Z}$ and $\hat{L}$ to compute an albedo:

$$\hat{A} = I - S(\hat{Z},\hat{L}) \quad (42)$$

Note that because $\hat{L}$ is an expectation over L, that our algorithm is capable of producing any illumination within the convex hull of the illuminations in our training set, despite $P(L)$ being a discrete model.

Optimization can be sped up through the observation that, after optimization has progressed for some time, the vast majority of illumination conditions are assigned a posterior very near zero ($P(L_i|A,Z) \approx 0$). To take advantage of this, at each evaluation of our loss function we fit a normal distribution to all L_i, weighted by $P(L_i|A,Z)$, and then flag all L_i within a Mahalanobis distance of 6 (all L_i within six standard deviations) as being "active". At the next evaluation of the loss function, only "active" illuminations are considered. This means that once $P(L_i|A,Z)$ starts getting peaked, optimization becomes much faster. It's important to note that an illumination condition that has been removed from the active set can be revived if the $P(L|A,Z)$ distribution shifts towards it, at which point it will be within the normal distribution that we fit on each iteration. In practice, this approximation does not seem to effect the output of optimization.

6. Error

Our error metric I-MSE is an approximation of the expectation of the mean-squared error between renderings of our recovered shape and albedo against the true shape and albedo, over all illumination conditions. Formally:

$$I\text{-MSE} \approx \frac{1}{n}\sum_{x,y} \mathrm{E}_L\left[\left(\hat{\rho}_{x,y}\exp(S(\hat{Z},L)_{x,y}) - \rho^*_{x,y}\exp(S(Z^*,L)_{x,y})\right)^2\right]$$

This error metric was introduced in [1], but with some mistakes in their derivation of the efficient closed-form approximation of this error metric. Here we will present a corrected version of I-MSE and its derivation.

First, we will define $\hat{\rho}$ and $\hat{\mathbf{n}}$ our estimated albedo and normal at some pixel, and ρ^* and $\mathbf{n}^*$, the true normal and albedo at that pixel. Ignoring the $\max(0,\cdot)$ term in Lambertian reflectance, the squared error in the renderings of our recovered albedo and normal and the true albedo and normal given some illumination $\mathbf{L}$ is:

$$Err(L_x,L_y,L_z) = (\hat{\rho}\times(L_x\hat{n}_x + L_y\hat{n}_y + L_z\hat{n}_z) - \rho^*\times(L_xn^*_x + L_yn^*_y + L_zn^*_z))^2$$

Let:

$$d_x = \hat{\rho}\hat{n}_x - \rho^*n^*_x \quad (43)$$
$$d_y = \hat{\rho}\hat{n}_y - \rho^*n^*_y \quad (44)$$
$$d_z = \hat{\rho}\hat{n}_z - \rho^*n^*_z \quad (45)$$

Then we can rewrite the error as:

$$Err(L_x,L_y,L_z) = (d_xL_x + d_yL_y + d_zL_z)^2 \quad (46)$$

We would like to integrate over Err over the unit sphere of illuminations. This is equivalent to the following:

$$L_x = \cos(\theta)\sin(\phi) \quad (47)$$
$$L_y = \sin(\theta)\sin(\phi) \quad (48)$$
$$L_z = \cos(\phi) \quad (49)$$
$$\mathrm{E}[Err] = \frac{3}{4\pi}\int_0^{2\pi}\int_0^{\pi}\sin(\phi)Err(L_x,L_y,L_z)\,d\theta d\phi = d_x^2 + d_y^2 + d_z^2 \quad (50)$$

We define I-MSE as the mean of $\mathrm{E}[Err]$ over all pixels in the image for which we have ground-truth shape and albedo.

$$I\text{-MSE} = \frac{1}{n}\sum_{x,y}\left\|\hat{\rho}_{x,y}\hat{N}_{x,y} - \rho^*_{x,y}N^*_{x,y}\right\|_2^2 \quad (51)$$

7. Dataset

Here we will detail how we recover "ground-truth" shape and spherical harmonic illumination for each image of each object in our dataset. This is a simple photometric stereo algorithm, in which we optimize over shapes and illuminations to minimize the absolute error between renderings of our dataset and the actual images in our dataset. Absolute error is used to give us robustness to errors due to shadows and specularities, which our rendering engine (and therefore, our dataset) do not consider or address properly. Recovered shapes and illuminations were then cleaned up by hand to address bas-relief ambiguity issues[2].

We treat each channel of each image as a separate image, so our dataset is in color. In the experiments in this paper, we use only the grayscale information we recover, by taking the average of the SH illuminations we recover over each channel, and taking the mean of the color reflectance images.

First, to account for varying albedo, we compute a "shading" image for each image on our dataset.

$$s^*_{i,j} = \exp(I_{i,j} - A_i) \tag{52}$$

We will now detail each step in the inner loop of our photometric stereo algorithm. We first take each current shape estimate Z, and linearize it to get a set of fixed surface normals. For each image j

$$L_j \leftarrow \arg\min_L \sum_i |\exp(S(\mathbf{n}_i, L)) - s^*_{i,j}| \tag{53}$$

This is done using Iteratively Reweighted Least-Squares. We then fix each image's illumination L_j, and optimize over each object's normals $\mathbf{n}_i$.

$$\mathbf{n}_i \leftarrow \arg\min_{\mathbf{n}} \sum_j |\exp(S(\mathbf{n}, L_j)) - s^*_{i,j}| \tag{54}$$

This optimization is done with L-BFGS. In this step, the normals are decoupled, and so surface integrability is not enforced. Given this estimate of surface normals, we can compute a integrable surface Z which approximates this normal field using least-squares:

$$Z \leftarrow \arg\min_Z \sum_i \left(Z * h^x - \frac{\mathbf{n}_i^x}{\mathbf{n}_i^z}\right)^2 + \left(Z * h^y - \frac{\mathbf{n}_i^y}{\mathbf{n}_i^z}\right)^2$$

These three optimization steps are repeated until convergence (30 iterations). For the first 10 iterations, we constrain all of the illuminations belonging to the same "scene" (light1, light2, diffuse, etc) to be scaled and shifted versions of each other, and then allow each illumination for every image to vary freely for the next 20 iterations. The result of this algorithm is an estimate of Z for each object and an estimate of L for each image.

This photometric stereo algorithm still suffers from Bas-Relief ambiguity[2] issues, despite the abundance of data. We therefore manually adjust each recovered Z over the three parameters of the Bas-Relief ambiguity by hand. Also, some regions of Z are clearly incorrect due to shadows. These regions are manually removed (and are not included in the evaluation of our error metrics which concern Z). After these manual tweaks to each shape, we update the set of illuminations to minimize absolute error once again. The two "cup" and "teabag" images did not have discriminative enough shape features for photometric stereo to recover reasonable second-order spherical harmonic illuminations, so for those objects we instead recover only first-order spherical harmonic illumination parameters (equivalent to point-light + ambient illumination).

The MIT Intrinsic Images dataset was not acquired with the goal of having the product of the "shading" and "reflectance" images be exactly equal to the diffuse image, which our model (and our baseline models) assume. That is, a lambertian rendering of our recovered shape and illumination resembles a scaled version of the original "shading" image. We correct for this by adjusting the brightness of the "shading" image such that it matches our rendering in a least-squares sense, and we use this "corrected" shading image in all of our experiments.

Note that the optimization tools we use for our photometric stereo algorithm are completely disjoint from the optimization techniques used by algorithm in our paper, despite the fact that those techniques could have been adapted to do photometric stereo. This was done intentionally to dispel any concerns that our results might be good simply because they were obtained using a similar toolset as our photometric stereo algorithm.

Examples of our recovered shapes and illuminations, as well as the shading and reflectance images already contained in the MIT Intrinsic Images dataset, can be seen in Figures 5 and 6. Images contributed by the MIT Intrinsic Images dataset are shown in blue. The "ground truth" shape and illumination generated by our photometric stereo algorithm are shown in red. The yellow images are renderings of our shape and illuminations, which look nearly identical to the reference images, thereby demonstrating that our recovered shapes and illuminations are reasonable.

8. Shape From Shading

Our model for recovering shape and albedo given a single image and illumination can easily be reduced to a model for doing classic shape-from-shading (recovering shape given a single image and illumination). Our optimization problem becomes:

$$\underset{Z}{\text{minimize}} \qquad \lambda|I - S(Z, L)| + f(Z) \tag{55}$$

Where I is the input log-image, and λ is a multiplier that trades off the importance of the reconstruction terms against the regularizer on Z. $f(Z)$ and $S(Z, L)$ are the same as defined in the paper. Optimization is done using our coarse-to-fine algorithm. This SFS algorithm is similar to past algorithms which optimize over a linearized representation of a depth map, with the primary difference being our choice of $f(Z)$.

This SFS algorithm is run on the shading images produced by the "intrinsic image" algorithms we benchmark against. This is a very generous comparison on our part, as we are effectively giving these other algorithms one-half of the model we present here. However, the performance of the comparison algorithms using less-sophisticated SFS algorithms, in terms of Z-MSE and I-MSE (the only error metrics that depend on recovered shape), was so poor that we felt it necessary to aggressively help these other algorithms so as to not appear biased. This means, however, that our improvement over these algorithms is not as much a reflection of the effectiveness of $f(Z)$ *in isolation*, but is instead a demonstration of the effectiveness of optimizing over $f(Z)$ and $g(A)$ to jointly recover shape and albedo, rather than generating a shading image without considering the underlying shape, and then attempting to recover Z from that shading image.

References

[1] J. T. Barron and J. Malik. High-frequency shape and albedo from shading using natural image statistics. *CVPR*, 2011. 4, 10

[2] P. Belhumeur, D. Kriegman, and A. Yuille. The Bas-Relief Ambiguity. *IJCV*, 1999. 5

[3] G. D. Finlayson, M. S. Drew, and C. Lu. Entropy minimization for shadow removal. *IJCV*, 2009. 1

[4] R. Grosse, M. K. Johnson, E. H. Adelson, and W. T. Freeman. Ground-truth dataset and baseline evaluations for intrinsic image algorithms. *ICCV*, 2009. 10, 11, 12

[5] B. K. P. Horn. Determining lightness from an image. *Computer Graphics and Image Processing*, 1974. 10

[6] J. C. Principe and D. Xu. Learning from examples with quadratic mutual information. *Workshop on Neural Networks for Signal Processing*, 1998. 1

[7] R. Salakhutdinov, S. Roweis, and Z. Ghahramani. Optimization with em and expectation-conjugate-gradient. *ICML*, 2003. 3

[8] J. Shen, X. Yang, Y. Jia, and X. Li. Intrinsic images using optimization. *CVPR*, 2011. 10

[9] L. Shen and C. Yeo. Intrinsic images decomposition using a local and global sparse representation of reflectance. *CVPR*, 2011. 10

[10] M. F. Tappen, W. T. Freeman, and E. H. Adelson. Recovering intrinsic images from a single image. *TPAMI*, 2005. 10

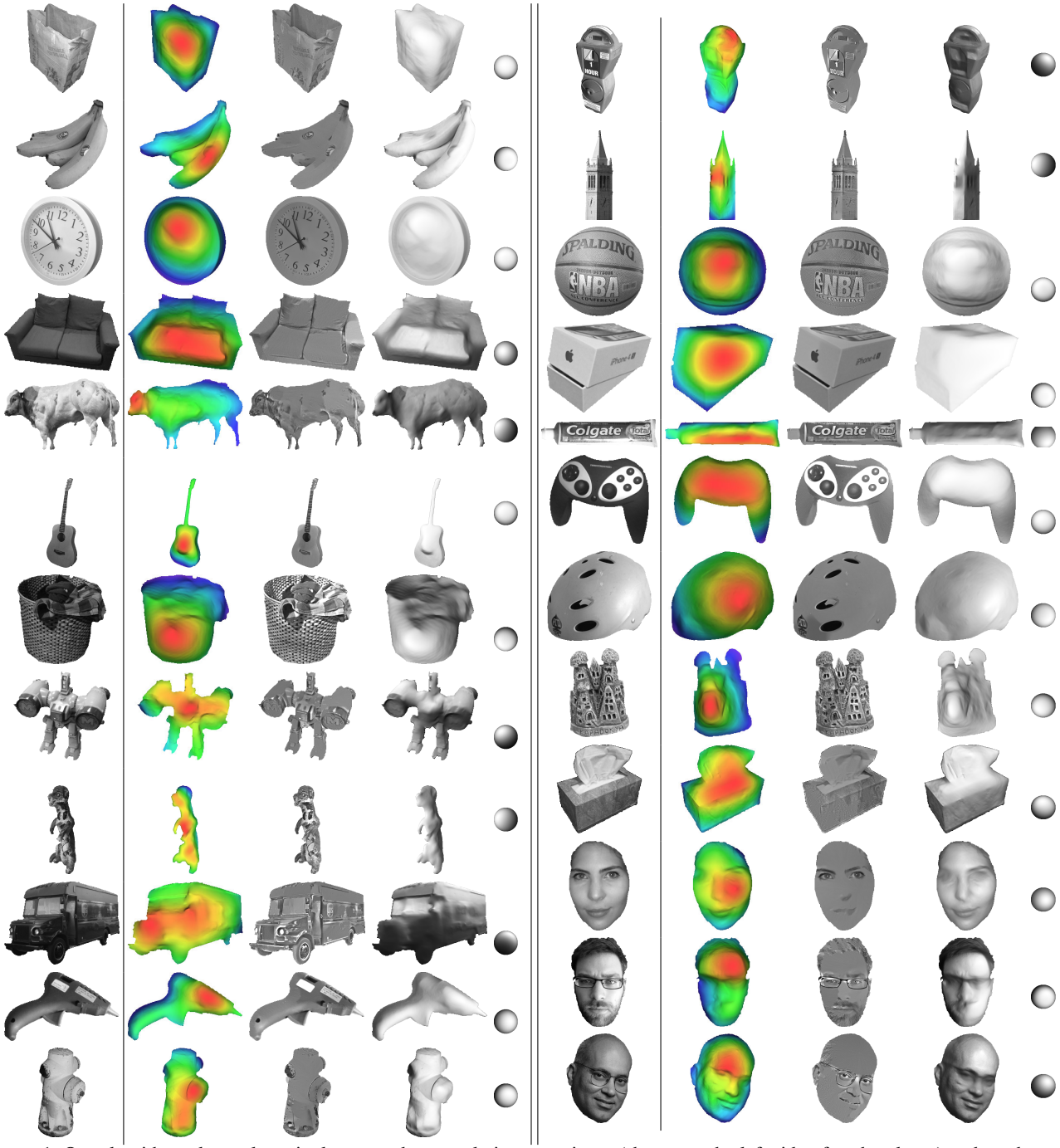


Figure 1. Our algorithm takes only a single cropped grayscale image as input (shown on the left side of each column) and produces as output a depth map, albedo map, shading image, and spherical harmonic illumination model. All images here were taken by the authors with a cellphone camera in uncontrolled indoor and outdoor illumination conditions, or downloaded from the internet. The "shading" image is a rendering of the recovered depth under the recovered illumination, and depth is shown with a psuedo-color visualization (red is near, blue is far). Our algorithm produces reasonable results, with most errors due to misattribution of image edges to albedo rather than shape, or vice versa. Other sources of error are: specularities in the image, which often cause raised shapes at the specular highlight, cast shadows, which are arbitrarily attributed to shape or albedo, or general low-frequency errors in shape, which are due the nature of shading with regard to providing low-frequency shape information, or equivalently to shortcomings in our current shape priors.

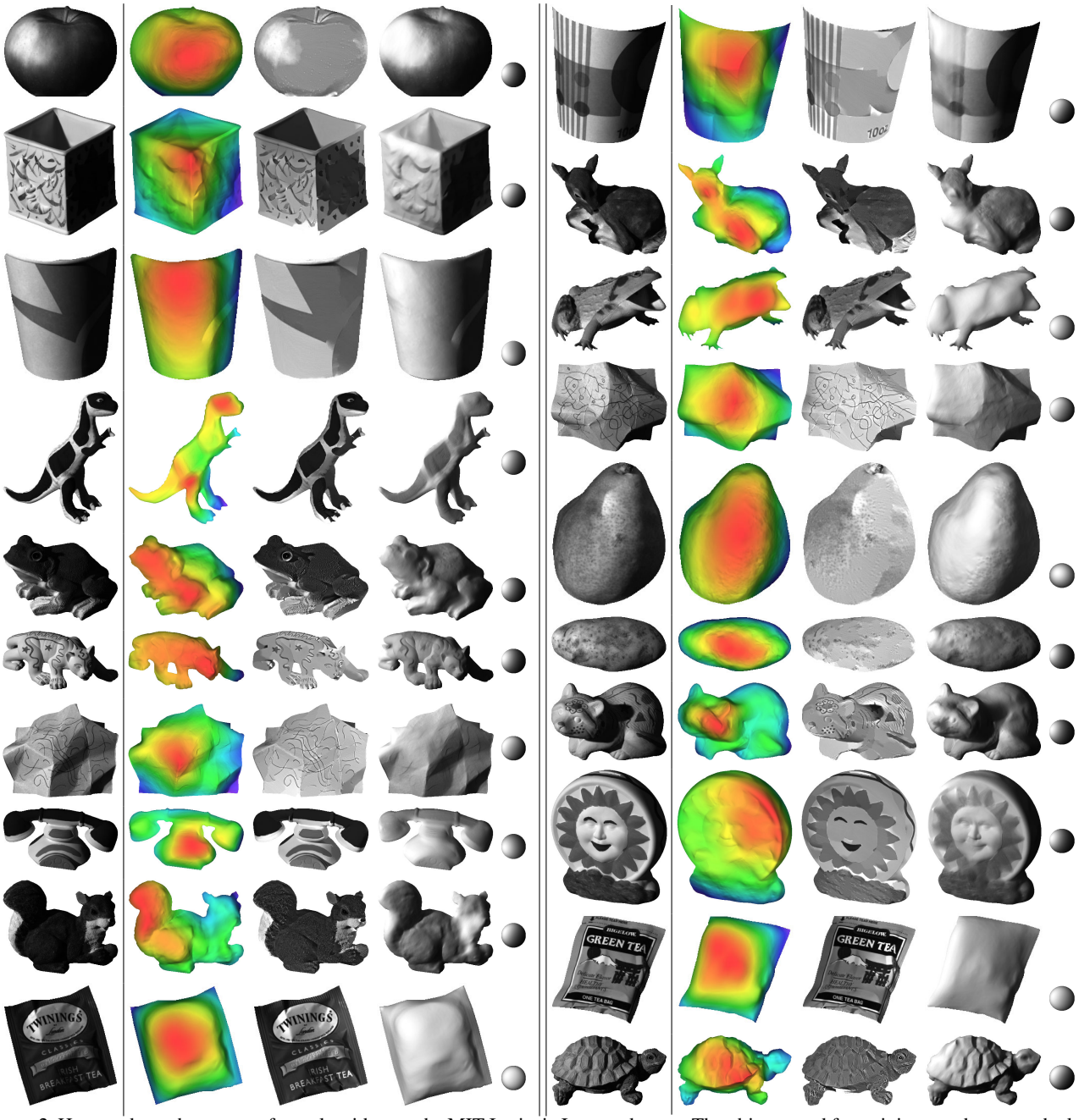
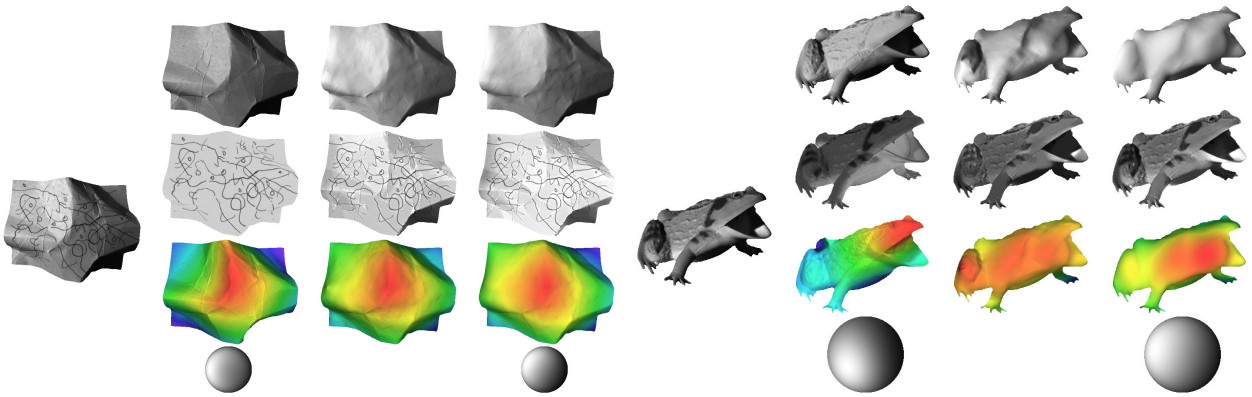
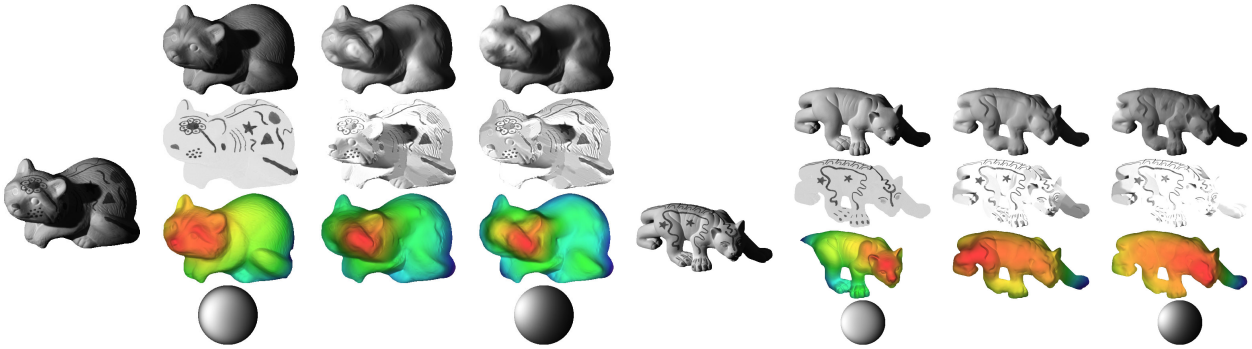


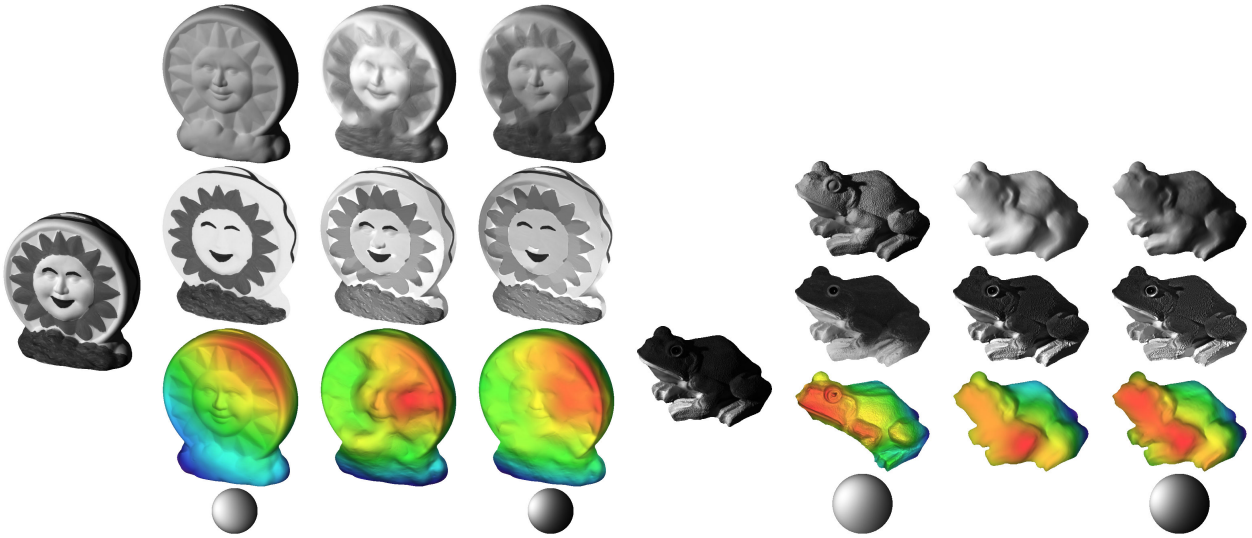
Figure 2. Here we have the output of our algorithm on the MIT Intrinsic Images dataset. The objects used for training are shown on the left, and the objects used for testing are shown in the right. The problem, and the presentation of the output, is exactly the same as in Figure 1



(a) Input Image (b) Ground Truth (c) Known Illumination (d) Unknown Illumination (e) Input Image (f) Ground Truth (g) Known Illumination (h) Unknown Illumination

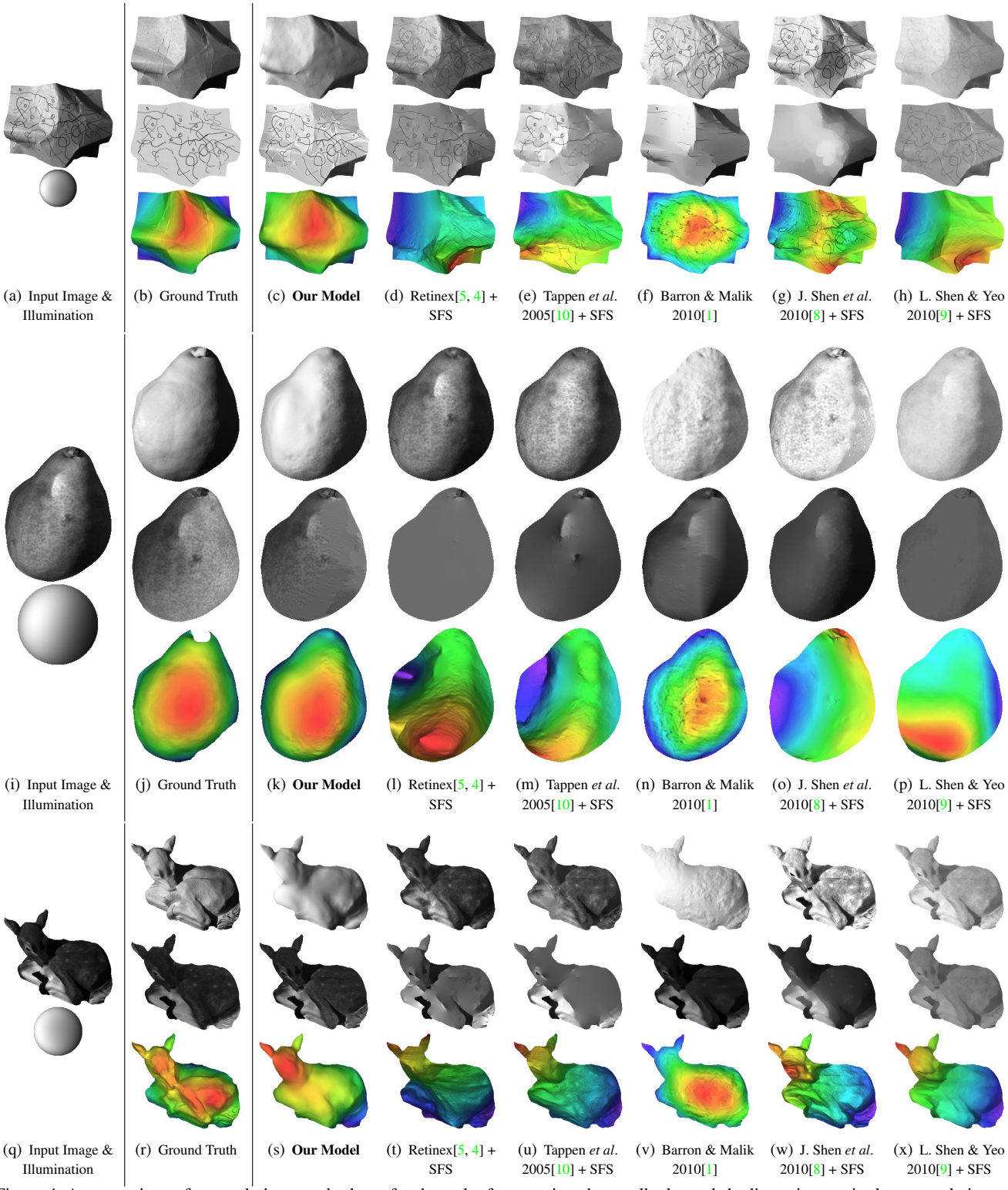


(i) Input Image (j) Ground Truth (k) Known Illumination (l) Unknown Illumination (m) Input Image (n) Ground Truth (o) Known Illumination (p) Unknown Illumination



(q) Input Image (r) Ground Truth (s) Known Illumination (t) Unknown Illumination (u) Input Image (v) Ground Truth (w) Known Illumination (x) Unknown Illumination

Figure 3. The output of our model, relative to ground-truth, when illumination is known and unknown, on objects from the MIT Intrinsic Images dataset. Though performance varies across the two cases, in both our model produces reasonable output.



(a) Input Image & Illumination | (b) Ground Truth | (c) **Our Model** | (d) Retinex[5, 4] + SFS | (e) Tappen *et al.* 2005[10] + SFS | (f) Barron & Malik 2010[1] | (g) J. Shen *et al.* 2010[8] + SFS | (h) L. Shen & Yeo 2010[9] + SFS

(i) Input Image & Illumination | (j) Ground Truth | (k) **Our Model** | (l) Retinex[5, 4] + SFS | (m) Tappen *et al.* 2005[10] + SFS | (n) Barron & Malik 2010[1] | (o) J. Shen *et al.* 2010[8] + SFS | (p) L. Shen & Yeo 2010[9] + SFS

(q) Input Image & Illumination | (r) Ground Truth | (s) **Our Model** | (t) Retinex[5, 4] + SFS | (u) Tappen *et al.* 2005[10] + SFS | (v) Barron & Malik 2010[1] | (w) J. Shen *et al.* 2010[8] + SFS | (x) L. Shen & Yeo 2010[9] + SFS

Figure 4. A comparison of our technique, and others, for the task of recovering shape, albedo, and shading, given a single grayscale image and a known illumination.

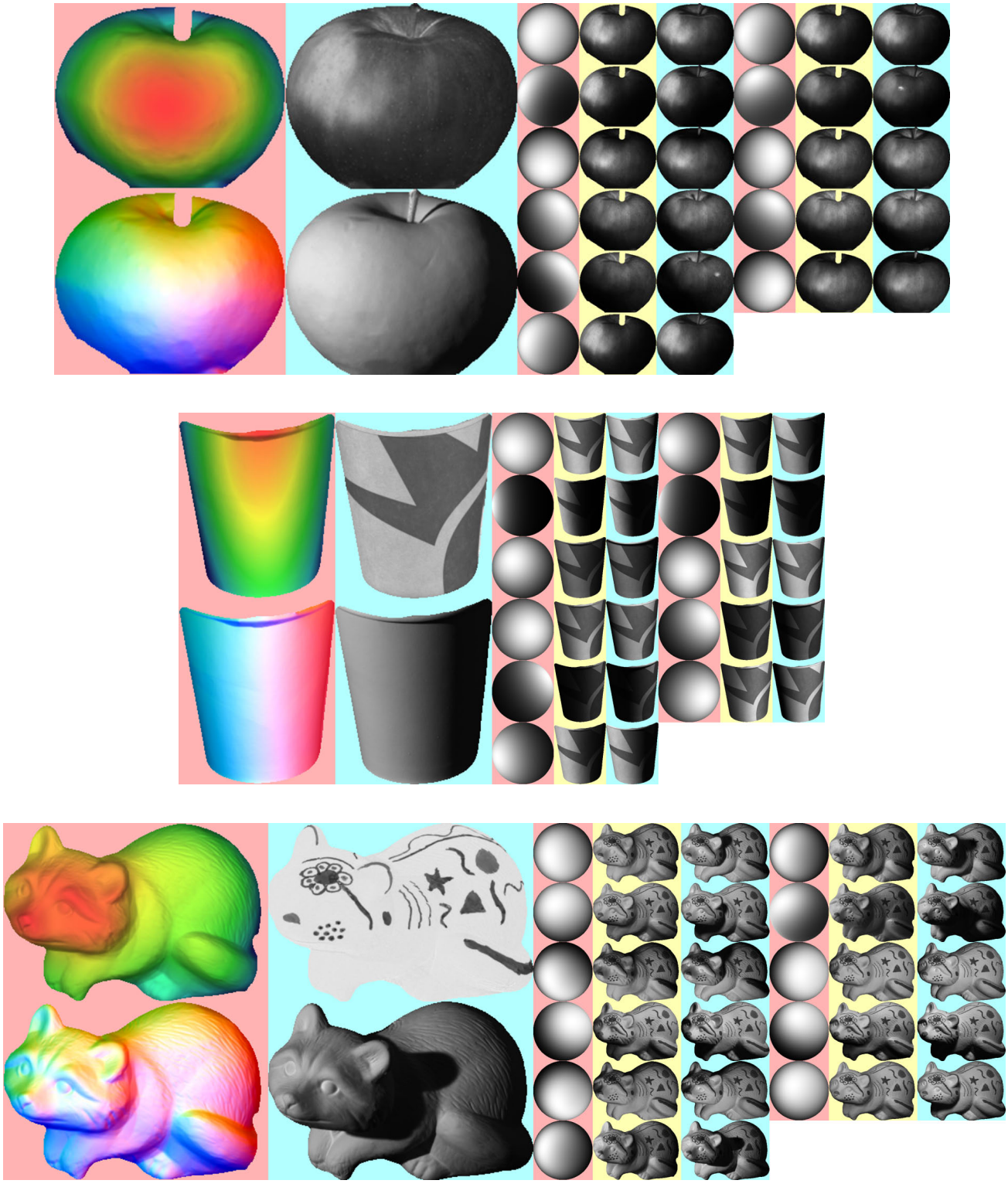


Figure 5. Some examples from our expanded version of the MIT Intrinsic Images dataset. Our contributions (shape, illumination) are shown in red, and the MIT Intrinsic Images dataset's[4] (albedo, shading, images) are shown in blue.

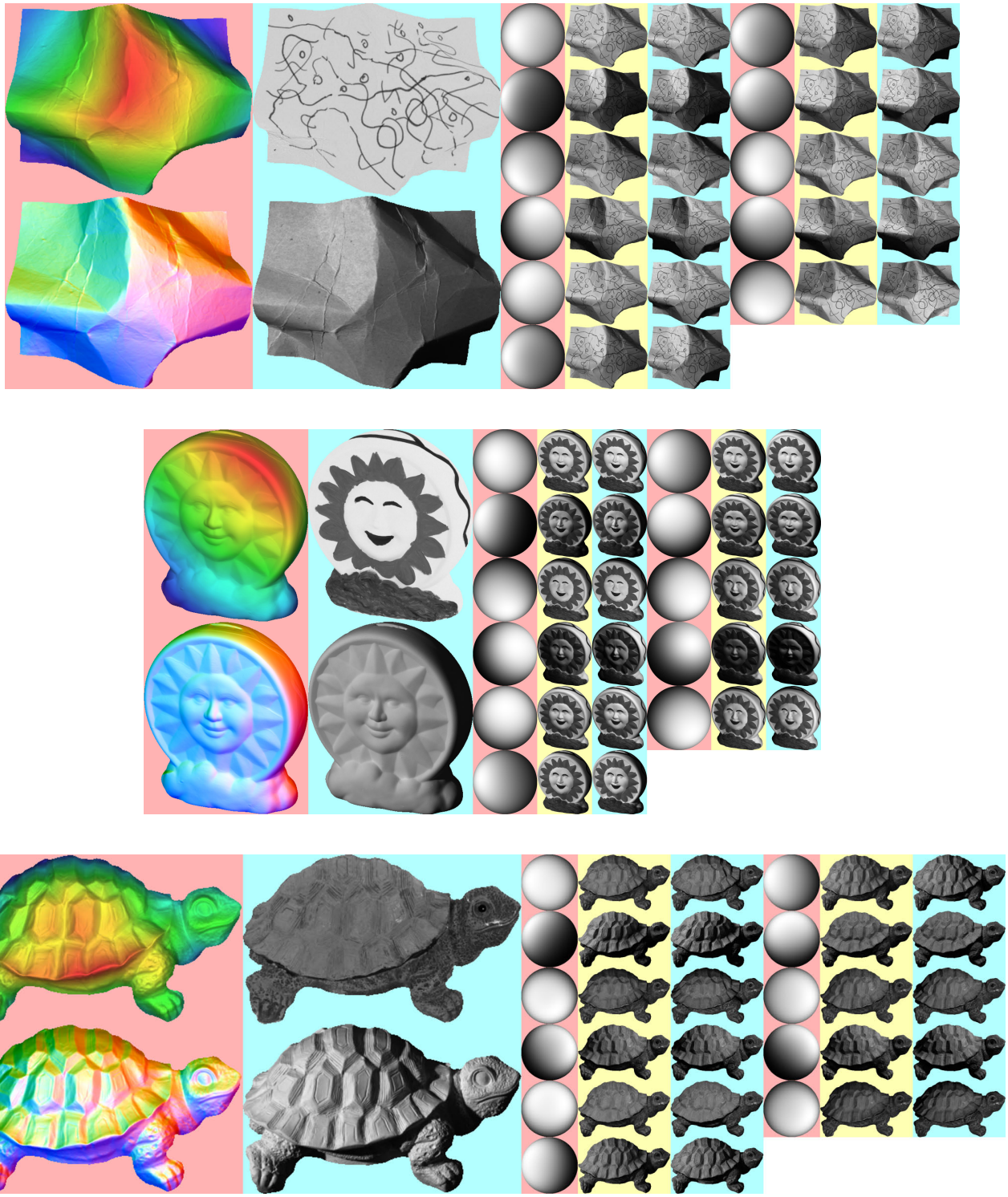


Figure 6. Some examples from our expanded version of the MIT Intrinsic Images dataset. Our contributions (shape, illumination) are shown in red, and the MIT Intrinsic Images dataset's[4] (albedo, shading, images) are shown in blue.

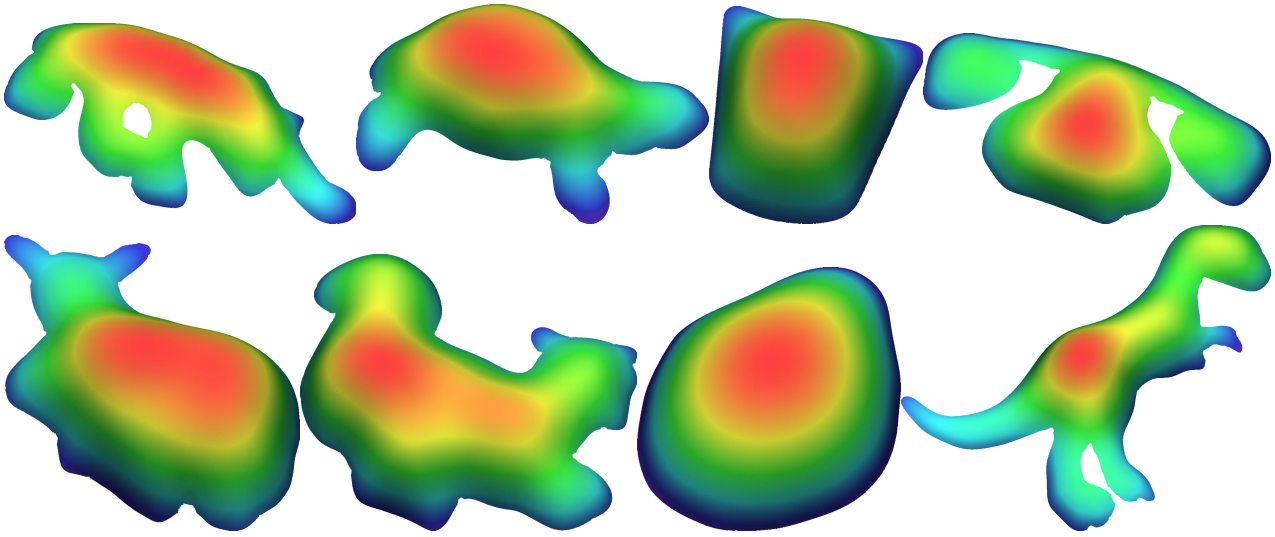


Figure 7. Some examples of the output of our “shape from contour” baseline.