Supplementary material to: Linear depth estimation from an uncalibrated, monocular polarisation image

William A. P. Smith[1], Ravi Ramamoorthi[2] and Silvia Tozza[3]

[1]University of York william.smith@york.ac.uk [2]UC San Diego ravir@cs.ucsd.edu [3]Sapienza - Università di Roma tozza@mat.uniroma1.it

In this supplementary document, we provide additional implementation details and experimental results. Specifically, we provide additional details regarding the implementation of our method (Sections 1, 2, 3 and 4), details on how we implemented the two comparison methods (Sections 5 and 6) and additional comparative experimental results (Section 7).

1 Specular sinusoid fitting

Estimating degree of specular polarisation requires an additional pre-processing step. Prior to fitting the model for specular degree of polarisation, we must first subtract the assumed diffuse component from the overall intensity. Our assumption is that for specular pixels the diffuse polarisation is insignificant compared to the specular polarisation. So, to correctly estimate the degree of polarisation, it should be the height of the sinusoid as a fraction of the unpolarised *specular* intensity rather than the combined diffuse/specular intensity. Specifically, we assume that

$$i^{\text{spec}}_{\vartheta_j}(\mathbf{u}) = i^{\text{diff}}(\mathbf{u}) + \frac{I_{\max}(\mathbf{u}) + I_{\min}(\mathbf{u})}{2} + \frac{I_{\max}(\mathbf{u}) - I_{\min}(\mathbf{u})}{2}\cos[2\vartheta_j - 2\phi(\mathbf{u})]. \tag{1}$$

Since we assume that the surface normal for specular pixels points approximately in the halfway direction, i.e.

$$\mathbf{u} \in \mathcal{S} \Rightarrow \mathbf{n}(\mathbf{u}) \approx \mathbf{h}, \tag{2}$$

we can compute an assumed diffuse intensity as:

$$i^{\text{diff}}(\mathbf{u}) = \mathbf{h} \cdot \mathbf{s}, \tag{3}$$

using the estimated or known light source vector $\mathbf{s}$. This quantity is subtracted from the captured polarimetric images prior to sinusoid fitting for specular pixels only.

2 Surface Priors

For robust performance on real world data, we find it advantageous (though not essential) to include two priors on the surface height: 1. a Laplacian smoothness term, 2. a convexity prior. Both are expressed as linear equations in the surface height.

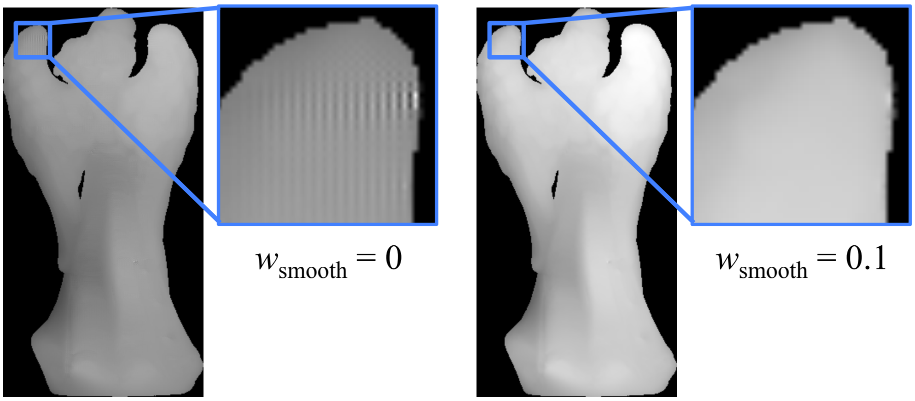


Fig. 1: Estimated depth maps without (left) and with (right) Laplacian smoothness prior. The inset zoomed region shows the "checkerboard" effect that occurs with no smoothing. Sensitivity to noise is also reduced.

2.1 Laplacian smoothness

The first prior is a Laplacian smoothness term. This adds a smoothness penalty, $\varepsilon_{\text{smooth}}$, to the overall objective function:

$$\varepsilon_{\text{smooth}} = w_{\text{smooth}} \sum_{\mathbf{u} \in \mathcal{F}} \left(z(\mathbf{u}) * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \right)^2, \tag{4}$$

where w_{smooth} weights the influence of the prior. This prior encourages a pixel to have a depth close to the average of its neighbours. It is minimised by locally planar regions, so can lead to oversmoothing of curved regions, but has the advantage of being linear in the surface gradient. We use a value of $w_{\text{smooth}} = 0.1$ in all of our experiments. The main benefit of this prior is to reduce noise caused by the independence of the gradient estimates between adjacent pixels caused by the smoothed central difference approximation. This effect is shown in Figure 1.

2.2 Convexity

The second prior (applicable only to objects with a foreground mask) is a convexity prior that encourages the azimuth angle of the surface normal to align with the azimuth of outward facing boundary normals. This is helpful for data that is noisy close to the occluding boundary, for example when some background is included in the image due to an inaccurate foreground mask.

We compute unit vectors in the image plane that are normal to the boundary and outward facing. To propagate these vectors into the interior, we erode the foreground mask and repeat this process until all pixels have been assigned an outward facing vector. Next, we smooth this vector field using a 9 by 9 window

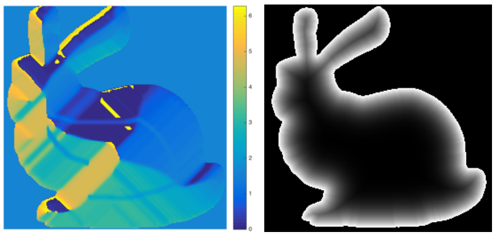

(a) Azimuth from boundary (b) Per-pixel weight

Fig. 2: Azimuth angles and weights for convexity prior.

approximating a Gaussian kernel with a standard deviation of 2. Finally, we convert these vectors to azimuth angles, $\alpha_b(\mathbf{u})$.

Now, to measure the deviation between $\alpha_b(\mathbf{u})$ and the azimuth angle of the estimated surface normal, we construct a surface normal vector $\mathbf{n}_b(\mathbf{u})$ using $\alpha_b(\mathbf{u})$ and the zenith angle estimated by polarisation, $\theta(\mathbf{u})$ (using Equation 6 or 9 in the paper for diffuse/specular pixels respectively):

$$\mathbf{n}_b(\mathbf{u}) = [\sin\alpha_b(\mathbf{u})\sin\theta(\mathbf{u}) \quad \cos\alpha_b(\mathbf{u})\sin\theta(\mathbf{u}) \quad \cos\theta(\mathbf{u})]^T. \tag{5}$$

We wish to encourage the boundary-propagated surface normal $\mathbf{n}_b(\mathbf{u})$ and the estimated surface normal $\mathbf{n}(\mathbf{u})$ to point in the same direction. To express this in terms of the surface gradient, we wish the gradient according to $\mathbf{n}_b(\mathbf{u})$ to be close to that according to $\mathbf{n}(\mathbf{u})$, i.e.

$$p(\mathbf{u}) = \frac{\sin\alpha_b(\mathbf{u})\sin\theta(\mathbf{u})}{\cos\theta(\mathbf{u})} \quad \text{and} \quad q(\mathbf{u}) = \frac{\cos\alpha_b(\mathbf{u})\sin\theta(\mathbf{u})}{\cos\theta(\mathbf{u})}. \tag{6}$$

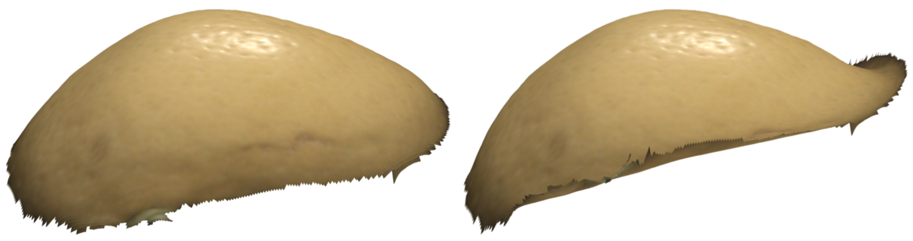
For numerical stability, we multiply both sides of these equations by $\cos\theta(\mathbf{u})$ (this avoids the magnitude of the equation becoming very large when $\theta(\mathbf{u})$ is close to π). Finally, we weight this prior such that it has high influence close to the boundary but the weight falls off rapidly as distance to the boundary increases. The per-pixel weights are defined as follows:

$$w_{\text{convexity}}(\mathbf{u}) = \left(\frac{[\max_{\mathbf{v}\in\mathcal{F}} d_b(\mathbf{v})] - d_b(\mathbf{u})}{\max_{\mathbf{v}\in\mathcal{F}} d_b(\mathbf{v})}\right)^m, \tag{7}$$

where $d_b(\mathbf{u})$ is the Euclidean distance from $\mathbf{u}$ to the boundary pixel closest to $\mathbf{u}$. The parameter m determines how quickly the weight reduces with distance from the boundary. We use $m = 5$ in our experiments. In Figure 2 we show the per-pixel weight and boundary azimuth angle for the Stanford bunny data.

The convexity prior is expressed via the following objective:

$$\varepsilon_{\text{convexity}} = \sum_{\mathbf{u}\in\mathcal{F}} w_{\text{convexity}}(\mathbf{u})[\,(p(\mathbf{u})\cos\theta(\mathbf{u}) - \sin\alpha_b(\mathbf{u})\sin\theta(\mathbf{u}))^2 + (q(\mathbf{u})\cos\theta(\mathbf{u}) - \cos\alpha_b(\mathbf{u})\sin\theta(\mathbf{u}))^2]. \tag{8}$$



(a) Estimated surface with convexity prior (b) Estimated surface without convexity prior

Fig. 3: Effect of the convexity prior.

We show surface reconstruction results in Figure 3 with and without this prior to demonstrate its effect.

Rather than assuming convexity, an alternative that could be explored in future work would be to encourage the normals to be close to the disambiguation chosen by the light source estimation. This would give additional weight to both the shading cue and polarisation measurements and avoid an assumption that is invalid for non-convex regions.

2.3 Running time

The smoothness and convexity priors add up to $3K$ additional equations to the least squares system in total. However, even with this larger system of equations, building and solving the system takes < 2 seconds in unoptimised Matlab code for the results shown in the paper. The example shown in Figure 3 (43,308 foreground pixels, 215,848 linear equations and 1,581,700 non-zero entries in the least squares system matrix) took 1 second to build and solve the linear system.

3 Specular labelling

In order to label specular dominant pixels, we combine two very simple heuristics: 1. rank order position of unpolarised pixel intensity and 2. specular coefficient in dichromatic model. We find that even without any tuning of parameters, this is sufficient to obtain a specular mask that enables good performance on real world data. However, a more sophisticated approach, that avoided manually chosen thresholds or perhaps that exploited polarisation cues, would clearly be a good target for future work.

The first heuristic simply computes the percentile of the unpolarised intensity:

$$h_1(\mathbf{u}) = \frac{\text{idx}(\mathbf{u})}{K}, \tag{9}$$

where $\text{idx}(\mathbf{u})$ is the rank order of the unpolarised intensity when sorted in ascending order and $K = |\mathcal{F}|$ the number of foreground pixels.

The second heuristic is based on the dichromatic model [1]. This heuristic is particularly effective when the object's diffuse colour is different to the colour of

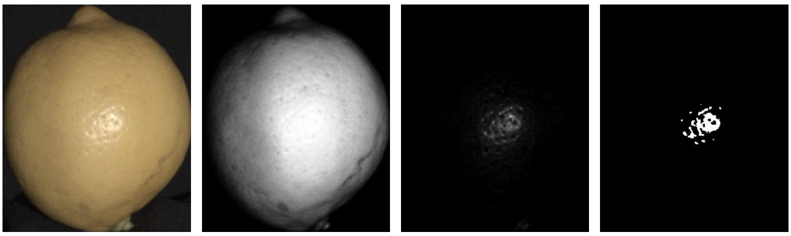

(a) Input (b) h_1 (c) h_2 (d) $\mathbf{u} \in \mathcal{S}$

Fig. 4: Specular labelling process: (a) colour input image, (b-c) heuristics, (d) labelling result.

the illumination. We begin by fitting a plane passing through the origin to the distribution of RGB values for the image foreground. In this two dimensional colour space, we then robustly fit the diffuse colour by maximising inliers when a line passing through the origin is fitted to the data ($\mathbf{c}_{\text{diff}} \in \mathbb{R}^3$) and finally complete the skewed-T fit by robustly fitting the specular colour ($\mathbf{c}_{\text{spec}} \in \mathbb{R}^3$) to the remaining degree of freedom. Hence, a pixel's colour can be expressed as a weighted combination of these two basis vectors: $\mathbf{i}(\mathbf{u}) \approx k_d(\mathbf{u})\mathbf{c}_{\text{diff}} + k_s(\mathbf{u})\mathbf{c}_{\text{spec}}$. The second heuristic is simply the specular coefficient:

$$h_2(\mathbf{u}) = k_s(\mathbf{u}). \tag{10}$$

Finally, any pixel with a degree of polarisation larger than the maximum possible diffuse degree of polarisation, ρ_{max}, must be a specular pixel.

The specular labelling is based on a logical AND of thresholds of the two heuristics, OR'd with the comparison to the maximum diffuse degree of polarisation:

$$(h_1(\mathbf{u}) > t_1 \wedge h_2(\mathbf{u}) > t_2) \vee (\rho(\mathbf{u}) > \rho_{\text{max}}) \Rightarrow \mathbf{u} \in \mathcal{S}. \tag{11}$$

We use $t_1 = 0.2$ and $t_2 = 0.9$ in our experiments. In Figure 4 we show an example of the labelling procedure including the values of the two heuristics and the final labelling result.

Note that, although we use colour here (and as input for both the synthetic and real image experiments in the paper), we only use one colour channel for illumination and surface height estimation. Colour provides a useful cue for specular labelling and may provide additional useful information for shape estimation. However, for simplicity, we subsequently use only the colour channel with highest average intensity over the foreground for shape and lighting estimation.

4 Finite difference gradient approximations

Having expressed the polarisation constraints as linear equations in the surface gradient, we subsequently substitute in finite difference approximations to the surface gradient in terms of the surface depth. This allows us to write a linear system in terms of the unknown surface depth and solve for depth directly. In the paper, we refer to using the Sobel operator. Here we provide more details on this and specifically what version of the operator we use.

The central difference approximation to the first derivative in the horizontal direction can be obtained by convolution with the kernel $[-1\ 0\ 1]$ (assuming unit spacing of the pixel grid). To reduce sensitivity to noise and improve robustness, the depth values are first smoothed using a centre-weighted kernel approximating a Gaussian. By associativity of the convolution operator we can pre-convolve the finite difference and smoothing kernels leading to the Sobel operators. The classical form of the Sobel operator uses a standard deviation for the Gaussian kernel of 0.85. In our implementation, we use a standard deviation of 0.6, giving the following kernels for computing the surface gradients in the horizontal and vertical gradient respectively:

$$\partial_x z \approx z * \frac{1}{12}\begin{bmatrix} -1 & 0 & 1 \\ -4 & 0 & 4 \\ -1 & 0 & 1 \end{bmatrix}, \quad \partial_y z \approx z * \frac{1}{12}\begin{bmatrix} 1 & 4 & 1 \\ 0 & 0 & 0 \\ -1 & -4 & -1 \end{bmatrix}. \tag{12}$$

This gives a higher weight to the central row/column and slightly reduces the smoothing effect. We found this made a very small, but not insignificant, improvement in performance.

5 Implementation of [2,3]

The first comparison method is due to Miyazaki et al. [2] and Atkinson and Hancock [3]. First, polarisation normals on the object boundary are disambiguated by choosing the direction that is closest to the outward facing boundary normal. Next, disambiguation is propagated inwards by processing pixels in decreasing order of estimated zenith angle and choosing the disambiguation that leads to the smoothest local neighbourhood when considering already-disambiguated normals. We show our implementation of this second step in Figure 5. The only unspecified aspect of the approach is the definition of smoothness used. In our implementation, we define the smoother solution as the one that has smaller mean angular deviation between the chosen normal and a 7×7 neighbourhood around the pixel. One extra detail is that, in the presence of noise, it is possible for a considered pixel to have no disambiguated normals in its neighbourhood. Hence, in our implementation the pixel considered next is the unprocessed pixel with largest zenith angle that also has at least one disambiguated pixel in its neighbourhood.

6 Implementation of [4]

The second comparison method is due to Mahmoud et al. [4]. The idea is as follows. First, the ambiguous polarisation normals are computed using the diffuse polarisation model, yielding two possible surface normals $\bar{\mathbf{n}}$ and $\mathbf{T}\bar{\mathbf{n}}$. Second, the degree of polarisation constraint is combined with a Lambertian shading constraint, yielding two further normals (we refer to these as the shading normals).

```
% Sort zenith angles of interior pixels into descending order
[~,idx]=sort(theta(interior),1,'descend');

[col,row]=meshgrid(1:cols,1:rows);
r = row(interior);
c = col(interior);

% At each iteration, we choose the unprocessed pixel with smallest zenith
% angle that has at least one neighbour
while ~isempty(idx)
   flag = false;
   selected = 1;
   % Consider pixels in ranked-theta order to find first one with at least
   % one neighbour (over 7x7 neighbourhood)
   while ~flag
      neighbourhood = [];
      for i=-3:3
         for j=-3:3
            if (i~=0) || (j~=0)
               if available_estimates(r(idx(selected))+i, ...
                                        c(idx(selected))+j)
                  neighbourhood = [neighbourhood;
                                     r(idx(selected))+i c(idx(selected))+j];
               end
            end
         end
      end
      if ~isempty(neighbourhood)
         % We need at least one neighbour to test smoothness
         flag=true;
      else
         selected=selected+1;
      end
   end
   % We now have a pixel with at least one neighbour
   r=r(idx(selected));
   c=c(idx(selected));
   % Select the normals from the neighbourhood
   Ns=[];
   for i=1:size(neighbourhood,1)
      Ns = [Ns;
            N(neighbourhood(i,1),neighbourhood(i,2),1) ...
            N(neighbourhood(i,1),neighbourhood(i,2),2) ...
            N(neighbourhood(i,1),neighbourhood(i,2),3)];
   end
   % Compute which local solution has smaller mean angular deviation from
   % its neighbours, our definition of smoothness
   n1 = [sin(phi(r,c))*sin(theta(r,c));
         cos(phi(r,c))*sin(theta(r,c));
         cos(theta(r,c))];
   n2 = [sin(phi(r,c)+pi)*sin(theta(r,c));
         cos(phi(r,c)+pi)*sin(theta(r,c));
         cos(theta(r,c))];
   % Select smoothest disambiguation and store chosen normal
   if mean(acos(Ns*n1))<mean(acos(Ns*n2))
      N(r,c,:)=n1;
   else
      N(r,c,:)=n2;
   end
   available_estimates(r,c)=true;
   idx(selected)=[];
end
```

Fig. 5: Matlab implementation of boundary propagation [2,3].

These two normals can be seen as the intersection of two cones whose axes are the viewer and light source directions and whose opening angles are determined by the degree of polarisation and unpolarised intensity respectively. With no noise, one of the polarisation normals will coincide with one of the shading normals, resolving the ambiguity. In the presence of noise, Mahmoud et al. [4] propose finding the azimuth angle that minimises the difference to the azimuth angle of the closest polarisation and shading normal. Put another way, we first find the polarisation/shading normal pair that are closest together and then average their azimuth angle.

In the original paper, this is expressed directly in terms of angles. However, we found that the expression given in the paper always yields azimuth angles in range $0 \dots \pi$, so we suspect there is an unwritten additional step to correctly handle the quadrant in which the vectors lie. For this reason, we implemented their method in terms of unit vectors rather than angles. A code snippet showing our implementation of this method is shown in Figure 6. The intersection of the two cones results in a quadratic equation whose two solutions are the two possible shading normals (`n1` and `n2` in the code snippet).

7 Additional results

In Figures 7 and 8 we show results related to the objects reported in Figure 7 of the main paper for the two comparison methods (note that the results are only for point source illumination since the comparison methods are not designed for outdoor illumination). It is clear from these examples that the comparison methods exhibit severe failures on real world images. The boundary propagation method obtains reasonable results for simple, convex objects (e.g. the lemon) although even here it is sensitive to specularities that cause noise. On objects containing internal concavities (e.g. the lobster), the recovered surface is highly distorted. On the other hand, the Lambertian disambiguation method does better on diffuse objects (e.g. the lobster), even when they contain internal concavities, but it is highly sensitive to noise, particularly on specular objects. Noise in the normals leads to over flattening of the recovered surfaces.

References

1. Shafer, S.A.: Using color to separate reflection components. Color Research & Application **10**(4) (1985) 210–218
2. Miyazaki, D., Tan, R.T., Hara, K., Ikeuchi, K.: Polarization-based inverse rendering from a single view. In: Proc. ICCV. (2003) 982–987
3. Atkinson, G.A., Hancock, E.R.: Recovery of surface orientation from diffuse polarization. IEEE Trans. Image Process. **15**(6) (2006) 1653–1664
4. Mahmoud, A.H., El-Melegy, M.T., Farag, A.A.: Direct method for shape recovery from polarization and shading. In: Proc. ICIP. (2012) 1769–1772

```
% Two possible ambiguous polarisation normals for every pixel:
N1(:,:,1)=sin(phi).*sin(theta);
N1(:,:,2)=cos(phi).*sin(theta);
N1(:,:,3)=cos(theta);
N2(:,:,1)=sin(phi+pi).*sin(theta);
N2(:,:,2)=cos(phi+pi).*sin(theta);
N2(:,:,3)=cos(theta);

for row=1:size(theta,1)
   for col=1:size(theta,2)
       % Compute two intersections between Lambertian cone
       % and degree of polarisation cone
       a = s(1);
       b = s(2);
       c = s(3)*cos(theta(row,col))-i_un(row,col);
       d = -sin(theta(row,col))^2;
       ny1 = -(b*c + a*(- d*a^2 - d*b^2 - c^2)^(1/2))/(a^2 + b^2);
       ny2 = -(b*c - a*(- d*a^2 - d*b^2 - c^2)^(1/2))/(a^2 + b^2);
       nx1 = sqrt(sin(theta(row,col))^2-ny1^2);
       nx2 = -sqrt(sin(theta(row,col))^2-ny1^2);
       % These are the two possible shading normals:
       n1 = [nx1 ny1 cos(theta(row,col))]';
       n2 = [nx2 ny2 cos(theta(row,col))]';
       % Find which is closer to its closest polarisation normal:
       n1best = max( dot(n1,squeeze(N1(row,col,:))), ...
                     dot(n1,squeeze(N2(row,col,:))) );
       n2best = max( dot(n2,squeeze(N1(row,col,:))), ...
                     dot(n2,squeeze(N2(row,col,:))) );
       if n1best>n2best
          na = n1;
       else
          na = n2;
       end
       % and take the polarisation normal closest to the chosen
       % cone normal:
       if dot(na,squeeze(N1(row,col,:)))>dot(na,squeeze(N2(row,col,:)))
          nb = squeeze(N1(row,col,:));
       else
          nb = squeeze(N2(row,col,:));
       end
       % Finally, find the vector whose azimuth angle is halfway between
       % the two chosen normals (averaging is done on vectors to avoid
       % angle wrap around issues)
       n = na+nb;
       n(1:2) = n(1:2)./norm(n(1:2));
       n(1:2) = n(1:2).*sin(theta(row,col));
       n(3) = cos(theta(row,col));
       N(row,col,:)=n;
   end
end
```

Fig. 6: Matlab implementation of Mahmoud et al. [4].

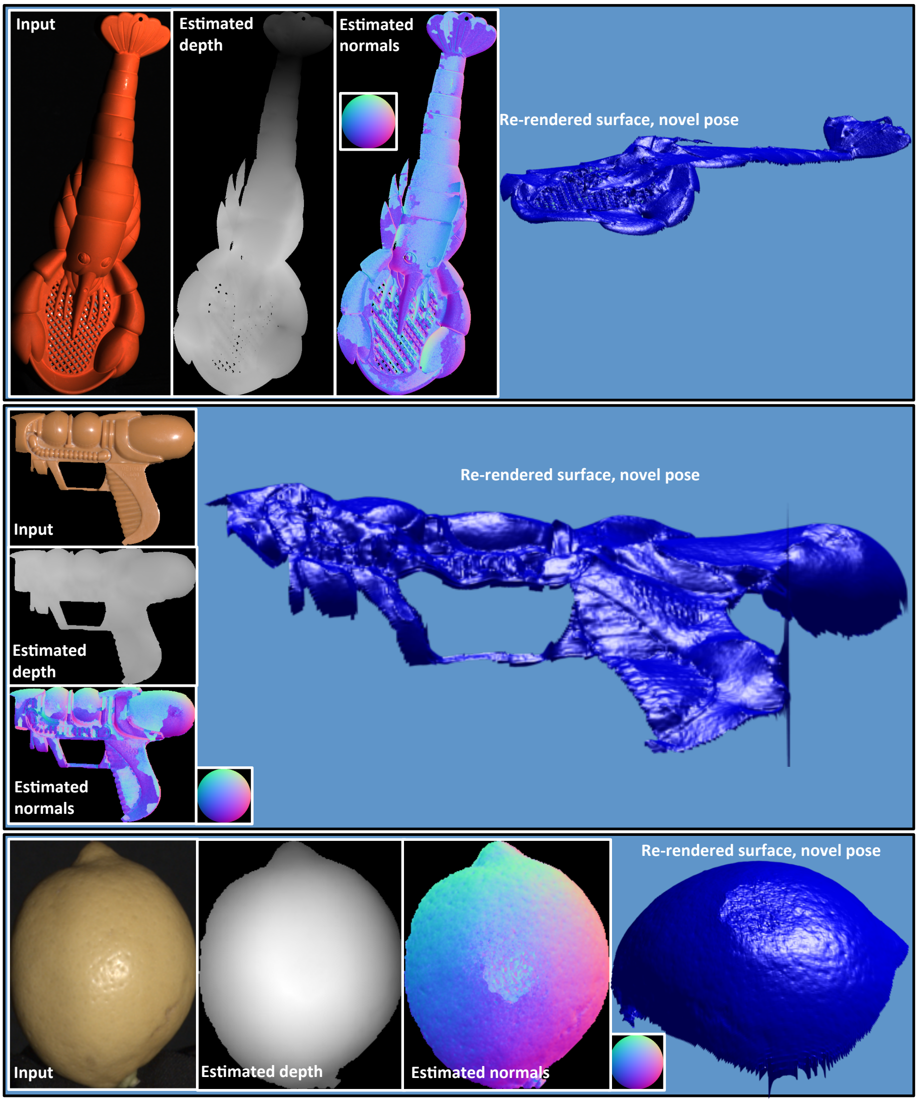


Fig. 7: Additional qualitative results for [2,3].

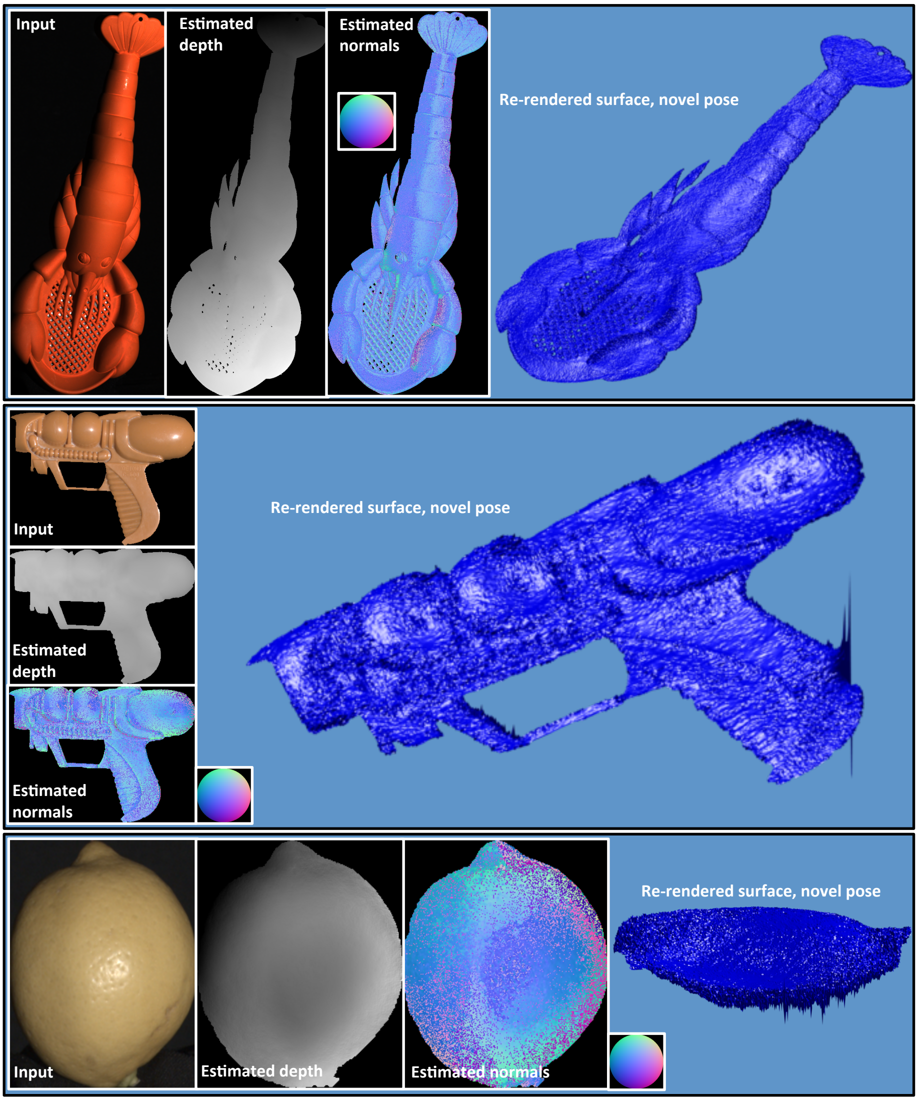


Fig. 8: Additional qualitative results for Mahmoud et al. [4].