

# Towards Fast and Scalable Normal Integration using Continuous Components

Francesco Milano<sup>1</sup>

Jen Jen Chung<sup>2</sup>

Lionel Ott<sup>1</sup>

Roland Siegwart<sup>1</sup>

<sup>1</sup>ETH Zurich

<sup>2</sup>The University of Queensland

## Abstract

*Surface normal integration is a fundamental problem in computer vision, dealing with the objective of reconstructing a surface from its corresponding normal map. Existing approaches require an iterative global optimization to jointly estimate the depth of each pixel, which scales poorly to larger normal maps. In this paper, we address this problem by recasting normal integration as the estimation of relative scales of continuous components. By constraining pixels belonging to the same component to jointly vary their scale, we drastically reduce the number of optimization variables. Our framework includes a heuristic to accurately estimate continuous components from the start, a strategy to rebalance optimization terms, and a technique to iteratively merge components to further reduce the size of the problem. Our method achieves state-of-the-art results on the standard normal integration benchmark in as little as a few seconds and achieves one-order-of-magnitude speedup over pixel-level approaches on large-resolution normal maps.*

## 1. Introduction

The problem of reconstructing a surface from its normal map, known as normal integration, arises naturally in many applications of computer vision, particularly in photometric shape recovery, such as shape from shading [12] and from polarization [1, 2] and photometric stereo [13, 25], in which the objective is to reconstruct a 3D surface from shading information, using estimated normals as an intermediate step.

The mathematical problem underlying normal estimation has been extensively studied and several recent methods have been proposed that achieve sub-millimeter accuracy on object-level normal maps [5, 16, 19]. While accurate, state-of-the-art approaches are predominantly based on a costly global optimization that jointly estimates the depth value at each pixel, with the number of optimization variables and constraints thereby scaling with the number of pixels. Additionally, multiple optimization steps with iterative reweighting are necessary to capture surface discontinuities, which cannot be correctly estimated a priori in the

general case and would otherwise cause the reconstruction to be suboptimal [5]. As a consequence, these methods have runtimes in the order of minutes for relatively low scales (typically in the order of  $10^4$  valid pixels) and up to hours for high resolutions and scales.

To address the above limitation, we observe that naturally occurring surfaces are often made of several smooth regions. If each such region was independently reconstructed, obtaining a single, global surface would reduce to optimally aligning each region to one another. Thus, we propose to recast normal integration into the estimation of the relative scales of continuous surface components. We introduce a simple and effective heuristic, based on normal similarity, to identify continuous components, and independently reconstruct each of them using the state-of-the-art formulation of [19]. We then jointly scale the depth of all pixels in each component by optimizing a single scale parameter for each component. For this, we reframe [19] and the discontinuity model of BiNI [5] to use relative component scales as optimization variables, and additionally introduce an outlier reweighting mechanism that rebalances the optimization terms. Importantly, we find that this reweighting significantly speeds up convergence also for previous, pixel-level methods, although their scalability remains limited due to the size of their optimization problem. Through its component-based formulation, our method greatly reduces the number of variables and constraints used in the optimization, resulting in a reduction of one order of magnitude in the execution time for mid-to-high resolution normal maps. Additionally, our approach achieves state-of-the-art reconstruction accuracy on the DiLiGenT benchmark [22] for normal integration in as little as a few seconds.

Thus, our contribution is a framework that recasts normal integration as an estimation of relative scales of continuous components, which (i) achieves state-of-the-art reconstruction accuracy on normal integration benchmarks, and (ii) enables scaling discontinuity-preserving normal integration to larger resolutions with an order of magnitude reduction in the execution time. Our code is publicly available<sup>1</sup>.

<sup>1</sup>[https://github.com/francescomilano172/normal\\_integration\\_continuous\\_components](https://github.com/francescomilano172/normal_integration_continuous_components).

## 2. Related work

We refer the reader to [20, 21] for extensive surveys of methods for normal integration. In the following Section, we focus on the most recent, state-of-the-art approaches.

A key challenge in normal integration is that the input normal map might represent a surface with discontinuities, which naturally arise, for instance due to self-occlusions. Failure to preserve such discontinuities results in overly continuous reconstructions with global distortions [4, 28]. To address this problem, a number of *single-analysis* methods have proposed detecting discontinuities as a preprocessing step, according to different strategies [15, 24, 26, 27]. These methods, however, tend not to be robust, since errors in the initial discontinuity detection cannot be later corrected during the reconstruction process. Recently, large improvements have been achieved by methods that estimate discontinuities through an iteratively reweighted optimization problem [5, 15, 19]. While accurate, these methods suffer from long execution times and scale poorly to larger resolutions, due to the need to jointly recover a depth value at all pixels through a single, global optimization problem.

To address this problem, Heep and Zell [10] have recently proposed a meshing-based approach for normal integration, in which a triangle mesh is precomputed from the input normal map based on estimated curvature and a surface reconstruction is obtained by optimizing the depth of each mesh vertex. While greatly reducing the number of variables and the execution time, the single, fixed mesh representation does not support modeling discontinuities, which is admittedly not straightforward and would require operations to realign the mesh and adjust its topology [10]. In this work, we aim at achieving the best of both discontinuity-preserving and computationally efficient approaches. Our component-based formulation is naturally compatible with state-of-the-art pixel-level methods, but also allows effectively preserving discontinuities while significantly improving scalability.

## 3. Surface normal integration using continuous components

An overview of our method and of its components is shown in Fig. 2 and in the form of pseudocode in Algorithm 1 (Supplementary). Section 3.1 formally introduces the problem and reviews state-of-the-art approaches from a unifying perspective. Section 3.2 illustrates our general framework and presents our reformulation of the underlying mathematical model. Section 3.3 describes our proposed heuristics to form continuous components from an input normal map (Fig. 2a), and Section 3.4 presents our strategy for optimizing their relative scales to retrieve a globally optimal reconstruction (Fig. 2b). Finally, Section 3.5 describes an optional step to merge multiple components (Fig. 2c).

### 3.1. Background

Surface normal integration is the problem of reconstructing a depth map from an input surface normal map, assuming known camera intrinsic parameters. Using the notation of [19], for a generic pair of neighboring pixels  $(a, b)$  with pixel coordinates  $(u_a, v_a)^\top$  and  $(u_b, v_b)^\top$ , respectively, we denote with  $\mathbf{n}_a = (n_{ax}, n_{ay}, n_{az})^\top$  and  $\mathbf{n}_b = (n_{bx}, n_{by}, n_{bz})^\top$  the surface normal vectors at those pixels. The values of the depth  $z_a, z_b$  at these pixels can then be modeled through a linear relationship in logarithmic space, which we refer to as a *model of continuity*:

$$\tilde{z}_a - \tilde{z}_b - \tilde{\omega}_{b \rightarrow a} = 0, \quad (1)$$

where  $\tilde{z}_a := \log(z_a)$ ,  $\tilde{z}_b := \log(z_b)$ , and the coefficient  $\tilde{\omega}_{b \rightarrow a}$  assumes a different value depending on the formulation. For instance, in BiNI [5],  $\tilde{\omega}_{b \rightarrow a}$  has the following form for a pinhole camera of focal lengths  $f_x$  and  $f_y$  and principal point  $(c_x, c_y)$ :

$$\tilde{\omega}_{b \rightarrow a} := \frac{\delta_{b \rightarrow a}}{n_{ax}(u_a - c_x) + n_{ay}(v_a - c_y) + n_{az}f}, \quad (2)$$

with  $f = f_x$ ,  $\delta_{b \rightarrow a} = \pm n_{ax}$  i.f.f.  $(u_b, v_b) = (u_a \pm 1, v_a)$  and  $f = f_y$ ,  $\delta_{b \rightarrow a} = \pm n_{ay}$  i.f.f.  $(u_b, v_b) = (u_a, v_a \pm 1)$ .

Milano *et al.* [19] instead derive the following alternative equation for generic central cameras:

$$\tilde{\omega}_{b \rightarrow a} := \log \left( \frac{\mathbf{n}_a^\top \boldsymbol{\tau}_m \cdot \mathbf{n}_b^\top \boldsymbol{\tau}_b}{\mathbf{n}_a^\top \boldsymbol{\tau}_a \cdot \mathbf{n}_b^\top \boldsymbol{\tau}_m} \right), \quad (3)$$

where  $\boldsymbol{\tau}_a$ ,  $\boldsymbol{\tau}_b$ , and  $\boldsymbol{\tau}_m$  denote, respectively, the ray direction vectors at pixel  $a$ , pixel  $b$ , and at a subpixel  $m$  that interpolates between  $a$  and  $b$ .

Typically, a global least-squares optimization problem is set up to recover the depth map, jointly enforcing the constraint (1) at all pairs of neighboring pixels. However, all models of continuity are approximate, since the input normal map only provides a discrete sampling of the surface normals. In addition, surface discontinuities may be present between two neighboring pixels, which act effectively as unknown constants of the integration problem and therefore cannot be estimated beforehand in the general case. As an example, consider the case of the chair in Fig. 1: while the surface of the chair and that of the wall behind it can be separately reconstructed, the exact depth discontinuity at the boundary cannot be estimated from the normals alone, since infinitely many solutions exist. As a result, an unknown residual term  $\chi_{b \rightarrow a}$  should be introduced in the model of continuity to account for discontinuities:

$$\chi_{b \rightarrow a} := \tilde{z}_a - \tilde{z}_b - \tilde{\omega}_{b \rightarrow a}. \quad (4)$$

While in the general case the integration solution is up to these residuals, a number of assumptions can be made

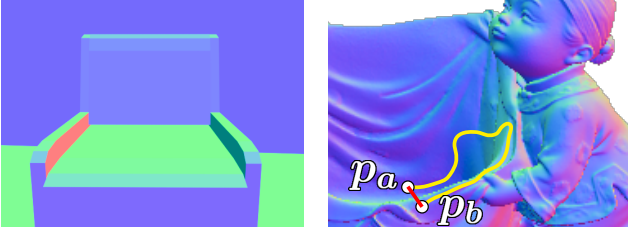


Figure 1. **Example of surface discontinuities.** *Left:* The chair in the foreground is separated by a full surface discontinuity from the wall in the background. Infinitely many depth maps can describe the input normal map, and solutions of the integration are up to the relative scales of the chair and the wall. *Right:* The surface points  $p_a$  and  $p_b$  are separated by a discontinuity along the path in red, but the yellow path connects them along the surface without discontinuities. *Source of object meshes :* [17] (left), [22] (right).

about the discontinuities. In particular, discontinuities often arise from self-occlusions rather than from fully disconnected surfaces, for instance, in normal maps that capture a single object [22]. In general, even in the presence of multiple objects, while the residual  $\chi_{b \rightarrow a}$  between two neighboring pixels  $a$  and  $b$  might be significant in magnitude, there often exists an alternative path along the surface that connects the two corresponding surface points along a trajectory with no discontinuities (Fig. 1, right).

State-of-the-art methods implicitly exploit this fact by assigning a lower weight in the optimization objective to the constraints (1) that, over the course of the optimization, are found to have a large residual magnitude. The rationale is that if all constraints were equally weighted, those associated with discontinuities would incorrectly drive the optimization to assign them a small residual, causing the reconstruction to converge to an overly continuous surface. By iteratively adapting the weights of the constraints, these methods instead focus on the more continuous paths along the surface, and thereby more accurately recover discontinuities, while still reconstructing a connected surface. A complementary approach, recently proposed by [19], consists in dynamically updating the coefficients  $\tilde{w}_{b \rightarrow a}$  in the equations (1) that have large residuals, so as to explicitly take into account the magnitude of the discontinuities. This is achieved through an additive term in the logarithm (3), which admits a geometrical interpretation. We refer to the exact mechanism by which the above methods achieve the reweighting of the constraints as a *model of discontinuity*.

The leading model of discontinuity is the bilateral weighting scheme proposed by BiNI [5] which acts through two different mechanisms. First, it relies on the assumption that surfaces are semi-smooth. This implies that, indexing the two neighboring pixels  $b$  and  $-b$  on opposite sides of a pixel  $a$ , the depth map has a discontinuity between at most one of the sides  $(a, b)$  or  $(a, -b)$ . This is modeled by weighting the constraint (1) between  $a$  and  $b$  by the factor,

$$w_{b \rightarrow a}^{\text{BiNI}} = \sigma_k(\gamma_{b \rightarrow a}^2 \cdot ((\tilde{z}_{-b} - \tilde{z}_a)^2 - (\tilde{z}_b - \tilde{z}_a)^2)), \quad (5)$$

where  $\sigma_k(x)$  is the sigmoid function  $(1 + e^{-kx})^{-1}$  with hyperparameter  $k$ , and  $\gamma_{b \rightarrow a}$  is a pixel-specific factor. If at a given point in the optimization it holds that  $(\tilde{z}_{-b} - \tilde{z}_a)^2 \ll (\tilde{z}_b - \tilde{z}_a)^2$ , i.e., the surface is estimated to be significantly more continuous on the side  $(a, -b)$  than on the side  $(a, b)$ , the value of  $w_{b \rightarrow a}^{\text{BiNI}}$  will tend to 0 and the constraint between  $a$  and  $b$  will therefore be down-weighted.

The second mechanism is through the factor  $\gamma_{b \rightarrow a}$  itself, the square of which is used to scale the term  $w_{b \rightarrow a}^{\text{BiNI}}$ , so that the overall weight associated to each constraint (1) is,

$$W_{b \rightarrow a} = \gamma_{b \rightarrow a}^2 \cdot w_{b \rightarrow a}^{\text{BiNI}}. \quad (6)$$

As noted by [19], the factor  $\gamma_{b \rightarrow a}$  can be expressed as

$$\gamma_{b \rightarrow a} = f \cdot \mathbf{n}_a^\top \boldsymbol{\tau}_a, \quad (7)$$

where  $f$  is the focal length of the camera, and is crucial to the optimization convergence, which it affects through (6) by introducing a multiplicative factor  $(\mathbf{n}_a^\top \boldsymbol{\tau}_a)^2$ . Since the quantity  $\mathbf{n}_a^\top \boldsymbol{\tau}_a$  tends to 0 for pixels  $a$  that lie close to an object boundary [18], this weighting effectively assigns lower confidence to constraints involving pixels that are likely to lie next to a depth discontinuity.

These approaches enforce all constraints in a global optimization problem of the form  $(\mathbf{A}\tilde{\mathbf{z}} - \mathbf{b})^\top \mathbf{W}(\mathbf{A}\tilde{\mathbf{z}} - \mathbf{b})$ , where  $\mathbf{W} := \text{diag}(W_{b \rightarrow a})$ . The optimization variable  $\tilde{\mathbf{z}}$  represents the log-depth at each pixel and has dimensionality equal to the number of pixels. Similarly, the design matrix  $\mathbf{A}$ , while sparse, scales with the number of constraints and the number of pixels. Thus, the overall complexity of the optimization scales poorly as the input dimensionality increases, quickly becoming infeasible for medium-to-large resolution normal maps. In the next Sections, we present our method for effective and scalable discontinuity-aware normal integration using continuous components.

### 3.2. General framework

Our proposed framework draws inspiration from the observation that surfaces typically consist of large regions that are locally smooth, and therefore contain no surface discontinuities. As a consequence, if each of these surface “patches” – which we refer to as *continuous components* – was independently described using the model of continuity (4), the residuals  $\chi_{b \rightarrow a}$  within the component would all be close to 0 in magnitude. As a result, each of the component-level reconstructions, taken separately, would approximate the corresponding surface patch with high accuracy. Obtaining a global reconstruction of the full surface would then reduce to estimating the discontinuities between each pair of neighboring components, which as detailed in the previous Section, could be achieved through a *model of discontinuity*. Crucially, since these components separately and accurately describe a fixed surface patch, estimating the discontinuities

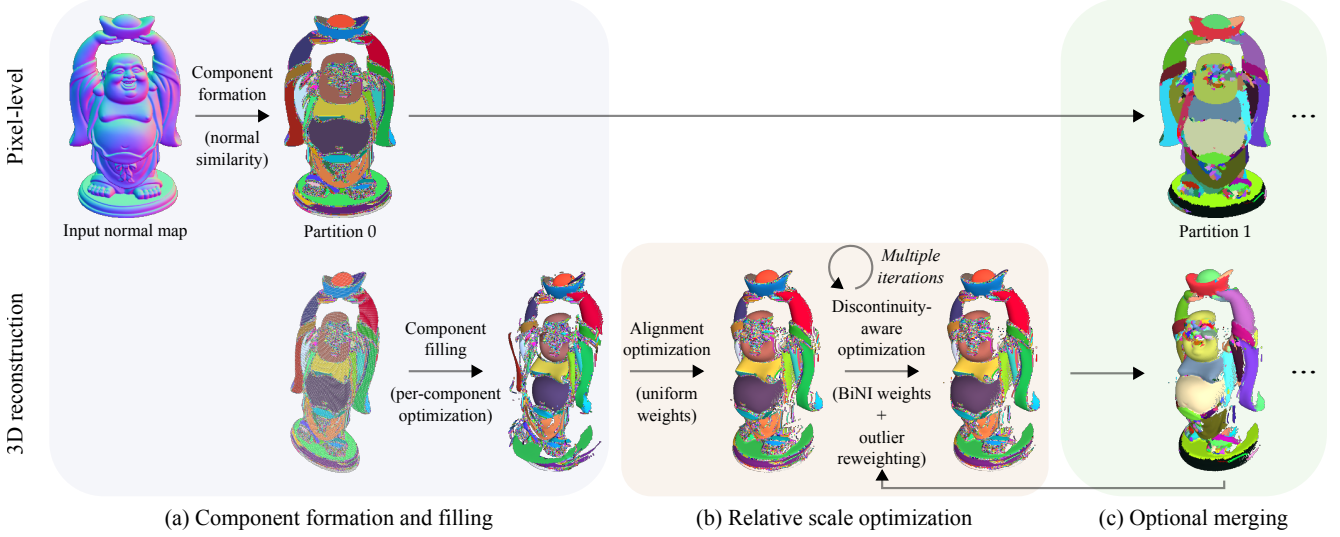


Figure 2. **Method overview.** We recast normal estimation as the estimation of the relative scale of continuous components. (a) Form continuous components based on the similarity of normals at neighboring pixels, independently reconstruct associated surface patches via per-component optimization (Sec. 3.3). (b) Align surface patches and recover discontinuities using relative scale optimization based on inter-component constraints (Sec. 3.4). (c) Optionally, during optimization, merge components to further reduce complexity (Sec. 3.5).

between them can be reframed as adjusting the *scale* of each component relative to one another (Fig. 2b). For instance, in Fig. 1, a different magnitude of discontinuity between the chair and the wall in the background could be achieved by scaling the depth of all points on the surface of the chair by the same factor  $s_1$ ; a factor  $s_1 > 1$  would allow rigidly moving the chair closer to the wall, while a value  $s_1 < 1$  would move the chair further away from it and closer to the camera. While conceptually similar to optimizing the depth of each chair- and wall pixel, which relies on all inter-pixel constraints, estimating the *relative scales* of the chair and the wall greatly reduces the optimization complexity, since it only requires solving for two values and involves only constraints on the inter-object boundaries.

To better appreciate the difference in complexity, let us look at the problem from a graph-theoretic perspective, which also allows us to introduce a notation that will be convenient to describe our framework’s components in the subsequent Sections. Existing approaches based on global optimization of per-pixel (log-)depth values are akin to modeling the problem using a dense graph  $\mathcal{G}_0 = (V, E)$ , where the set of vertices  $V$  comprises a node for each valid pixel in the input normal map and the set of edges  $E$  is defined by all pairs of valid neighboring pixels according to the chosen pixel connectivity (Fig. 3a). The optimization problem then consists of estimating the value of a variable for each node in the graph, with constraints defined by the graph edges.

When operating with our proposed *continuous components*, optimization can instead be seen as based on a meta-graph, or *quotient graph*  $\mathcal{Q}_m$ , obtained from  $\mathcal{G}_0$  by partitioning its vertices  $V$  into a set of components  $\{\mathcal{C}_c^{(m)}\}$ , s.t.  $V = \bigcup_c \mathcal{C}_c^{(m)}$  (Fig. 3b), where we index with  $m$  a spe-

cific version of the partitioning, which can subsequently be updated. In particular, since all pixels in a component  $\mathcal{C}_c^{(m)}$  belong to the same surface patch, we let their scale change jointly and define a single meta-node  $\hat{\mathcal{C}}_c^{(m)}$  in the meta-graph. Since each surface patch is considered fixed up to scale, all constraints relating pixels in the same component are ignored when optimizing the relative scale of the components. Therefore, the edges in the meta-graph only include constraints between pixels in different components.

Formally, for each component  $\mathcal{C}_c^{(m)}$  let us denote the set of its *intra-component* edges (with both endpoints in  $\mathcal{C}_c^{(m)}$ ) and the set of its *inter-component* edges (with only one endpoint in  $\mathcal{C}_c^{(m)}$ ) as

$$\begin{aligned} E(\mathcal{C}_c^{(m)}) &:= \{(a, b) \in E \mid a, b \in \mathcal{C}_c^{(m)}\} \\ \partial\mathcal{C}_c^{(m)} &:= \{(a, b) \in E \mid a \in \mathcal{C}_c^{(m)} \wedge b \notin \mathcal{C}_c^{(m)}\}. \end{aligned} \quad (8)$$

Let us further refer to the set of all intra-component edges and the set of all inter-component edges respectively as

$$E_{\text{intra}}^{(m)} := \bigcup_c E(\mathcal{C}_c^{(m)}), \quad E_{\text{inter}}^{(m)} := \bigcup_c \partial\mathcal{C}_c^{(m)}, \quad (9)$$

and let us denote with  $\pi_m$  the mapping from a pixel to the index of its corresponding node in the quotient graph  $\mathcal{Q}_m$ :

$$\pi_m : V \mapsto \{0, \dots, |\mathcal{C}^{(m)}| - 1\}, \text{ s.t. } \forall a \in V, a \in \mathcal{C}_{\pi_m(a)}^{(m)}. \quad (10)$$

With the above notation, we define the quotient graph as

$$\mathcal{Q}_m = (\mathcal{C}^{(m)}, E_{\text{inter}}^{(m)}), \text{ with } \mathcal{C}^{(m)} := \{\hat{\mathcal{C}}_c^{(m)}\}. \quad (11)$$

At optimization iteration  $t$ , with partitioning version  $m$ , the goal of the optimization is to scale, for each component



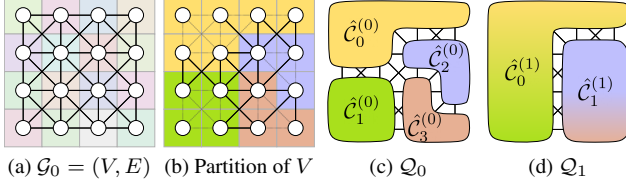


Figure 3. **Graph-theory interpretation of our framework.** (a) Pixel-level optimization can be seen as operating on a dense graph  $\mathcal{G}_0 = (V, E)$ . (b) We propose partitioning the per-pixel vertices  $V$  into components. (c) The resulting optimization operates on a *quotient graph*  $\mathcal{Q}_0$  (11). (d) Optionall, components can be merged into one another, forming a reduced quotient graph (Sec. 3.5).

$\mathcal{C}_c^{(m)}$ , the depth values  $\mathbf{z}_c^{(t)} := (z_a \mid a \in \mathcal{C}_c^{(m)})^\top$  of all pixels in  $\mathcal{C}_c^{(m)}$  by a multiplicative factor  $s_c$ , or equivalently to estimate an additive factor  $\tilde{s}_c := \log(s_c)$  to apply to all corresponding log-depth values  $\tilde{\mathbf{z}}_c^{(t)} = (\tilde{z}_a \mid a \in \mathcal{C}_c^{(m)})^\top$ . We therefore set as optimization variable the vector of relative scales to apply to each component,

$$\tilde{\mathbf{s}}^{(t)} := \left( \tilde{s}_0^{(t)}, \dots, \tilde{s}_{|C^{(m)}|-1}^{(t)} \right)^\top, \quad (12)$$

and we define the optimization constraints by imposing the following updated version of the model of continuity uniquely on the inter-component edges  $(a, b) \in E_{\text{inter}}^{(m)}$ :

$$\bar{\chi}_{b \rightarrow a}^{(t)} := \tilde{z}_a^{(t-1)} - \tilde{z}_b^{(t-1)} - \tilde{\omega}_{b \rightarrow a} + \tilde{s}_{\pi_m(a)}^{(t)} - \tilde{s}_{\pi_m(b)}^{(t)}. \quad (13)$$

After each optimization iteration, the log-depth at each pixel is efficiently updated in parallel by rigidly scaling all pixels in the same component by the same factor, i.e.,  $\forall a \in V$ ,

$$\tilde{z}_a^{(t+1)} \leftarrow \tilde{z}_a^{(t)} + \tilde{s}_{\pi_m(a)}^{(t)}, \quad (14)$$

or equivalently,  $\forall c \in \{0, \dots, |C^{(m)}| - 1\}$ ,

$$\tilde{\mathbf{z}}_c^{(t+1)} \leftarrow \tilde{\mathbf{z}}_c^{(t)} + \tilde{s}_c^{(t)} \mathbf{1}. \quad (15)$$

As a final remark, it is worth noting that our relative-scale framework can also be applied in the limit where each component only contains a single pixel. In this case, the quotient graph coincides with the dense graph  $\mathcal{G}_0$ , hence there are no advantages over per-pixel log-depth optimization in terms of number of variables and constraints. However, as we show in Section 4, we find that our formulation converges to a slightly more accurate solution.

### 3.3. Formation and filling of the initial components

To exploit the computational advantages of our relative-scale framework, we need a decomposition of the input normal map into regions that are likely to correspond to continuous surface patches. For this, we propose a simple but effective heuristic based on the observation that if the derivative  $f'$  of a generic signal  $f$  is continuous at a point, then the signal itself is continuous at that point. Since surface

normals represent a form of derivative of the depth map to reconstruct (hence the name “normal *integration*”), we propose to exploit the local continuity of surface normals as a proxy for the continuity of the underlying surface. In particular, for each pair of neighboring pixels  $(a, b) \in E$ , we compute the relative angle,  $\theta_{a,b} := \arccos(\mathbf{n}_a^\top \mathbf{n}_b)$ , between the normals  $\mathbf{n}_a$  and  $\mathbf{n}_b$  at each pixel. We then form continuous components  $\{\mathcal{C}_c^{(0)}\}$  by finding the connected components of the subgraph obtained from  $\mathcal{G}_0$  by only selecting edges for which  $\theta_{a,b} < \theta_c$ , where  $\theta_c$  is a hyperparameter. As we show in Section 4, we find that for small values of  $\theta_c$ , this simple heuristic allows effectively recovering large, approximately continuous surface regions.

Once the continuous components are detected, we perform a separate optimization for each component  $\mathcal{C}_c^{(0)}$ , through which we “fill” the log-depth values of its pixels. We apply a regular log-depth model of continuity (4) and the BiNI model of discontinuity (6) on its intra-component edges  $(a, b) \in E(\mathcal{C}_c^{(0)})$ , expressed in matrix form as:

$$\mathbf{A}_c = \begin{bmatrix} 1 & -1 & 0 & \dots \\ -1 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \quad \mathbf{b}_c = \begin{bmatrix} \tilde{\omega}_{b \rightarrow a} \\ \tilde{\omega}_{a \rightarrow b} \\ \vdots \end{bmatrix}, \quad (16)$$

$$\mathbf{W}_c = \text{diag} \left( \{W_{b \rightarrow a}\}_{(a,b) \in E(\mathcal{C}_c^{(0)})} \right).$$

Importantly, since no two components share any variables or constraints, the above optimization problem can be solved independently and in parallel for each of the components. Similarly to previous log-depth-based methods [5, 19], we initialize all log-depth values to 0 and perform optimization using the conjugate-gradient method [11], denoted as  $\text{cg}(\mathbf{A}, \mathbf{b})$ , on the normal equations:

$$\tilde{\mathbf{z}}_c^{(0)} \leftarrow \text{cg} \left( \mathbf{A}_c^\top \mathbf{W}_c \mathbf{A}_c, \mathbf{A}_c^\top \mathbf{W}_c \mathbf{b}_c \right). \quad (17)$$

We find a single iteration of the above optimization to be sufficient to accurately reconstruct each surface patch.

### 3.4. Relative scale optimization

Once the continuous components have each been independently reconstructed, an optimization of their relative scales is necessary to recover the full surface. For a given partitioning version  $m$  and an optimization iteration  $t$ , we rewrite our updated model of continuity (13) as follows,

$$\bar{\mathbf{A}}_m = \begin{bmatrix} \dots & 0 & 1 & \dots & -1 & \dots \\ \dots & 0 & -1 & \dots & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \quad (18)$$

$$\bar{\mathbf{b}}_m = \begin{bmatrix} \tilde{\omega}_{b \rightarrow a} - (\tilde{z}_a^{(t-1)} - \tilde{z}_b^{(t-1)}) \\ \tilde{\omega}_{a \rightarrow b} - (\tilde{z}_b^{(t-1)} - \tilde{z}_a^{(t-1)}) \\ \vdots \end{bmatrix},$$

where the constraints are defined on the inter-component edges  $(a, b) \in E_{\text{inter}}^{(m)}$ , rearranged so that the non-zero columns of the design matrix correspond to the indices  $\pi_m(a)$  and  $\pi_m(b)$  of the connected components. We then retrieve the logarithmic scale to be applied to each component ((14), (15)) through conjugate gradient:

$$\tilde{\mathbf{s}}^{(t)} \leftarrow \text{cg} \left( \overline{\mathbf{A}}_m^\top \overline{\mathbf{W}}_m^{(t)} \overline{\mathbf{A}}_m, \overline{\mathbf{A}}_m^\top \overline{\mathbf{W}}_m^{(t)} \overline{\mathbf{b}}_m \right). \quad (19)$$

Importantly, we note that after the initial filling of the components, the scale of each per-component reconstruction might differ significantly, since each optimization retrieves log-depth maps only up to scale (Fig. 2a). Thus, the inter-component residuals may be arbitrary and hence imbalance the subsequent optimization. Therefore, in the first iterations of relative scale optimization (we find 2 iterations to yield accurate results), we weight all constraints (13) equally, to align the components in the most continuous way possible; *i.e.*, for  $t \leq 1$ , we set  $\overline{\mathbf{W}}_m^{(t)} = \text{diag}(1)$ . In subsequent iterations, to retrieve the discontinuities we adopt the BiNI model of discontinuity (6).

However, we find that directly applying such a model results in suboptimal reconstructions. The reason for this is that the quotient graph  $\mathcal{Q}_m$  has in general a much lower number of edges than  $\mathcal{G}_0$ . As a consequence, constraints with large residuals that might have only partially affected the optimization in a dense graph assume a much larger weight in the smaller component-based optimization. To address this, we introduce an *outlier reweighting* strategy that reduces the influence of large residuals in subsequent iterations. In particular, we define two thresholds  $L, U$ , with  $0 < L < U$ , such that an equation with residual  $\bar{\chi}_{b \rightarrow a}$  is considered an outlier if  $|\bar{\chi}_{b \rightarrow a}| \geq U$  and an inlier if  $|\bar{\chi}_{b \rightarrow a}| \leq L$ . We model this through a soft weight,

$$W_{b \rightarrow a}^{\text{out}^{(t)}} := \sigma_1 \left( \frac{4}{\tilde{L} - \tilde{U}} (2 \log_{10}(|\bar{\chi}_{b \rightarrow a}^{(t-1)}|) - (\tilde{L} + \tilde{U})) \right), \quad (20)$$

where  $\tilde{\cdot} := \log_{10}(\cdot)$ , and we set in our experiments  $U = 10^{-3}$  and  $L = 10^{-5}$ . This provides an affine mapping in log space, with  $U$  mapped to  $-4$  and  $L$  mapped to  $4$ , resulting in an outlier weight of  $\sigma_1(-4) \approx 0.02$  when  $|\bar{\chi}_{b \rightarrow a}| = U$  and of  $\sigma_1(4) \approx 0.98$  when  $|\bar{\chi}_{b \rightarrow a}| = L$ . For all iterations  $t > 1$ , we multiply (20) to the BiNI weight, so that the overall weight matrix is,

$$\overline{\mathbf{W}}_m^{(t)} = \text{diag} \left( \{W_{b \rightarrow a} \cdot W_{b \rightarrow a}^{\text{out}^{(t)}}\}_{(a,b) \in E_{\text{inter}}^{(m)}} \right). \quad (21)$$

We run relative scale optimization until the optimization energy  $E_t := (\overline{\mathbf{A}}_m \tilde{\mathbf{s}}^{(t)} - \overline{\mathbf{b}}_m)^\top \overline{\mathbf{W}}_m^{(t)} (\overline{\mathbf{A}}_m \tilde{\mathbf{s}}^{(t)} - \overline{\mathbf{b}}_m)$  changes in relative terms by less than a threshold  $\Delta E_{\text{max}}$  or the number of iterations reaches a pre-defined value  $T$ .

### 3.5. Optional merging

Optionally, to further reduce the number of variables and constraints of the optimization, components can be iteratively merged into one another (Fig. 2c). After a certain number,  $\text{freq}_{\text{merging}}$ , of relative-scale optimization iterations with a specific partitioning  $\{\mathcal{C}_c^{(m)}\}$ , we can identify a new set of components as follows: (i) For each node  $\hat{\mathcal{C}}_c^{(m)}$  in the quotient graph  $\mathcal{Q}_m$ , select the edge among those incident to it that has the smallest residual magnitude  $|\bar{\chi}_{b \rightarrow a}|$ , *i.e.*,  $(\hat{a}, \hat{b})_c^{(m)} := \arg \min_{(a,b) \in \partial \mathcal{C}_c^{(m)}} |\bar{\chi}_{b \rightarrow a}|$ . (ii) Form a subgraph  $\hat{\mathcal{Q}}_m = (\mathcal{C}^{(m)}, \{(\hat{a}, \hat{b})_c^{(m)}\})$  from  $\mathcal{Q}_m$  using the selected edges. (iii) Find a new set of components  $\{\mathcal{C}_c^{(m+1)}\}$  by computing the connected components of  $\hat{\mathcal{Q}}_m$ . The process can then continue by optimizing for the relative scale of the new components (Sec. 3.4).

## 4. Experiments

### 4.1. Experimental settings

Our implementation relies on a combination of GPU-accelerated tensor operations and CPU-based operations (*e.g.*, conjugate-gradient-based optimization, for which we use the SciPy library [23]). We compare our method to the current state-of-the-art approach of [19], noting that we also use their model of continuity in our framework. We reimplement the baseline and integrate it into our framework, to decouple the effects of GPU acceleration and formulation-specific computational aspects. All experiments are run on a single NVIDIA RTX 3080 Laptop GPU.

### 4.2. Benchmark experiments

We evaluate on the DiLiGenT benchmark [22] for normal integration, which features ground-truth, object-level normal maps of resolution  $608 \times 512$ . In these maps the background is masked out, and a normal vector is defined only at the object pixels, resulting in 24 706 to 56 560 depth values to be estimated. For each object and run we report the mean average depth error (MADE) and the total execution time.

For both methods we use 8-connectivity and convergence parameters  $\Delta E_{\text{max}} = 10^{-3}$ ,  $T = 150$ . We refer the reader to the Supplementary for ablations on these parameters. Additionally, while slight speed-ups of our method can be achieved through our optional merging (see Supplementary), we find that this effect is not particularly significant, given the small number of valid pixels in the input normal maps. We therefore do not use merging in our DiLiGenT experiments, and show instead its benefit on larger-resolution maps in the next Section. We test our method for different values of the normal similarity threshold  $\theta_c$ , including the limit case in which each pixel is assigned to a component, which we denote as  $\theta_c = \text{None}$ . For the baseline, we include both the version with and without discon-

tinuity computation ( $\alpha_{b \rightarrow a}$  in [19]). Finally, we note that our proposed outlier reweighting mechanism can also be applied to the log-depth-based optimization of [19] and we therefore evaluate both methods with and without it.

The results of our evaluation are shown in Table 1. We start by noting that our outlier reweighting significantly improves the convergence time for *pixel-level* approaches, not only for our relative scale optimization, but also for the log-depth-based baseline of [19]. Importantly, for this baseline, outlier reweighting also improved accuracy, although with the exception of some objects (*cf.*, *buddha*, *cow*, *pot1*). Additionally, we note that while the discontinuity computation strategy of [19] is beneficial for objects with large discontinuities (*buddha*, *harvest*, *D vs. ND* in Table 1), its effect is partially reduced by the use of outlier reweighting.

Crucially, for our component-level versions, outlier reweighting is critical for achieving optimal accuracy, resulting in significantly better and more stable convergence across the board, particularly for objects with large discontinuities (*buddha*, *harvest*, *reading*, *goblet*), as can be seen by comparing successive rows in Table 1. Overall, while with outlier reweighting the performance of the two methods is comparable for the most continuous objects, our component-based variants achieve better accuracy than log-depth optimization for the more discontinuous objects (*buddha*, *harvest*, *reading*). This effect is a function of the chosen value for the threshold  $\theta_c$ , with smaller values generally resulting in lower error at the cost of slightly increased runtime. We refer to the Supplementary for details on the minimum theoretical error that can be achieved for different values of  $\theta_c$ . Notably, our pixel-level version also slightly outperforms the (pixel-level) formulation of [19], suggesting that solving for relative scales instead of log-depth may lead to a more well-posed optimization problem.

Finally, we note that, when using outlier reweighting both our component-based formulation and the pixel-level method of [19] achieve comparable runtime on the DiLiGenT benchmark. As we show in the next Section, however, the gap between the execution times of the two frameworks becomes significantly larger as the number of valid pixels in the normal map increases.

### 4.3. Evaluation of scalability

To assess the scalability of component-based and pixel-level formulations, we evaluate our method and the baseline on a set of mid-to-high-resolution normal maps that, unlike those from object-level datasets, have no masked-out pixels. We include normals rendered from meshes (using Blender-Proc [8]), real-world normals from photometric stereo [13], and normals predicted by the state-of-the-art normal estimation method DSINE [3] on in-the-wild images. The number of pixels with a valid normal vector varies between 311 296 and 1 264 640. We run our method with merging enabled

(with  $\text{freq}_{\text{merging}} = 5$ ) both in its component-based version (with  $\theta_c = 2.0^\circ$ ) and in its pixel-level variant, including for the latter also the version without merging. For all methods, we use outlier reweighting, with convergence criteria  $\Delta E_{\text{max}} = 10^{-3}$  and  $T = 15$ .

The results of our evaluation, together with the input normal maps and the output reconstructions, are shown in Fig. 4. Our component-based framework achieves convergence on average one order of magnitude faster than our pixel-level version and the pixel-level method of [19]. This result is a consequence of the smaller scale of our optimization problem. Using our normal similarity criterion, our method is able to detect large continuous regions in the input maps and thereby reduce the number of optimization variables by a factor between 5 and 50. It follows that while pixel-level methods require several minutes for convergence even at the same resolution used in the DiLiGenT dataset (*cf.* first column in Fig. 4), our method converges in a few seconds for the same input and in less than 2 minutes for the larger resolutions. While the log-depth-based formulation of [19] converges on average faster than our *pixel-level* variant under the same conditions, we note that, as opposed to the small-scale DiLiGenT normal maps, enabling merging for our version on large-scale normal maps results in significantly reduced runtime with little to no impact on the reconstruction. This highlights that merging can be particularly beneficial in settings where iterations with the full number of pixels are costly and therefore a reduction in the number of variables is desirable.

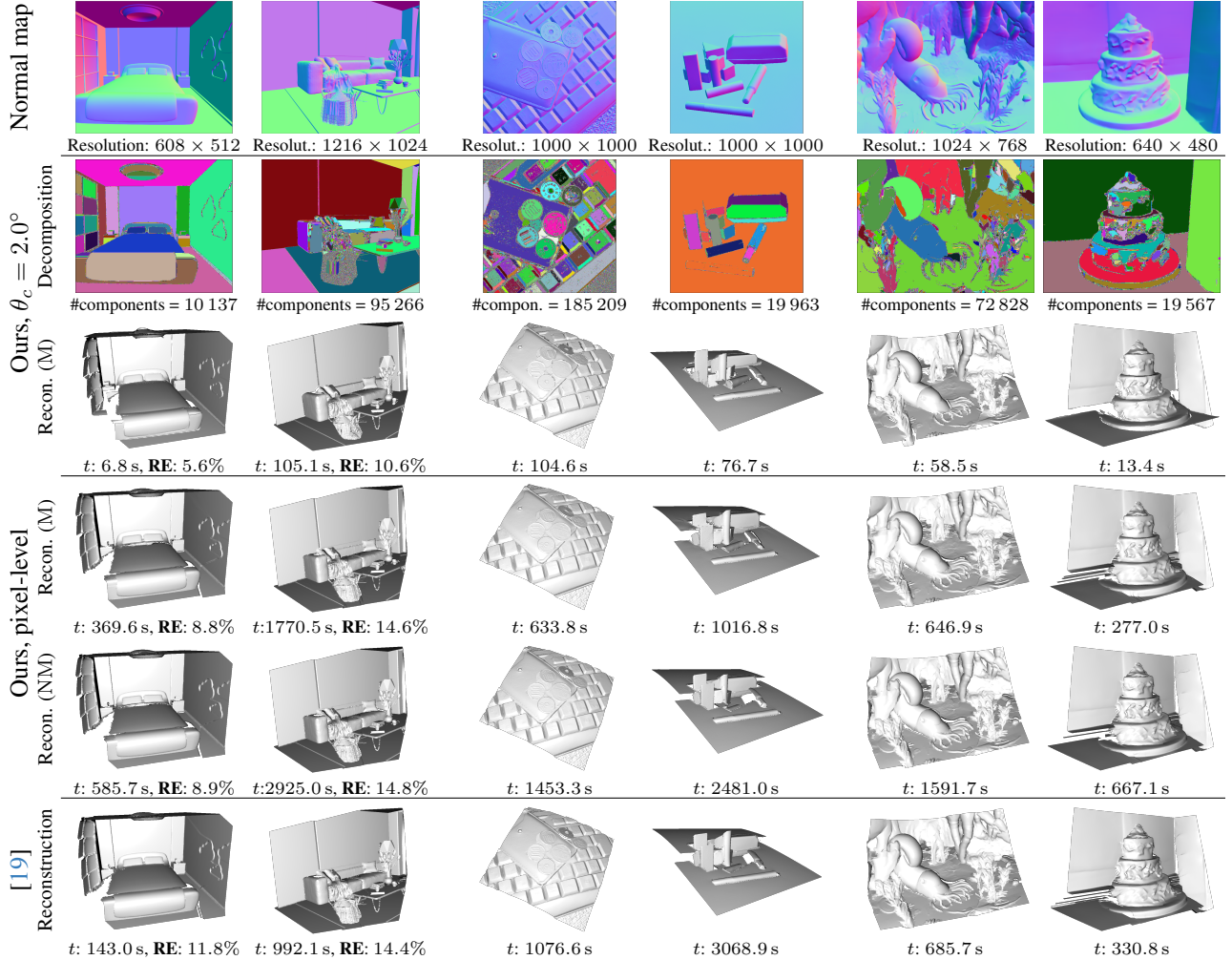
Our component-based approach achieves more accurate reconstructions than pixel-level methods also for these large-scale normal maps. Importantly, we note that our identification and separate optimization of components allows more accurately reconstructing particularly large continuous regions, for which the global optimization of pixel-level methods tends to introduce spurious discontinuities (Fig. 4: floor in columns 1, 4 and wall in column 6). We note however that, for a similar reason, due to the model of discontinuity, our method may occasionally separate continuous surface regions from one another if these are assigned to different components, particularly to background ones (Fig. 4: tentacles of the octopus in column 5). We discuss this and other limitations in more detail in Sec. H.

## 5. Conclusions

We proposed a framework for fast and scalable normal integration by reframing the problem as the optimization of the relative scales of continuous surface components. Our approach achieves state-of-the-art accuracy on the standard normal integration benchmark and enables scaling normal integration to large-resolution normal maps, with an order of magnitude reduction in the execution time while preserving discontinuities.

Method	$\theta_c$	$W_{b \rightarrow a}^{\text{out}}$	bear		buddha		cat		cow		harvest		pot1		pot2		reading		goblet*	
			Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$
[19], D	N/A	✓	<b>0.02</b>	2.70	0.62	3.32	0.05	2.98	0.13	1.49	2.07	6.96	0.47	<u>3.74</u>	0.15	2.86	0.25	2.02	<b>4.04</b>	3.45
	N/A	✗	0.07	5.02	0.44	19.89	<u>0.04</u>	32.68	<b>0.06</b>	15.18	2.80	51.00	<b>0.36</b>	54.31	<u>0.14</u>	24.73	0.92	6.57	7.96	31.52
	N/A	✓	<b>0.02</b>	1.84	0.70	3.04	0.05	1.87	0.13	1.15	2.21	5.99	0.47	4.13	<u>0.14</u>	<b>1.79</b>	0.24	<u>1.62</u>	<b>4.04</b>	1.62
[19], ND	N/A	✗	<u>0.05</u>	20.62	0.49	37.97	0.21	90.13	<u>0.07</u>	21.73	2.38	73.30	<u>0.37</u>	5.31	<u>0.14</u>	6.96	1.31	59.01	9.38	27.67
	None	✓	<b>0.02</b>	7.39	0.20	19.56	<b>0.03</b>	36.98	0.09	3.35	1.31	44.35	<b>0.36</b>	32.51	<b>0.13</b>	9.83	0.17	3.40	9.41	4.78
	None	✗	0.06	143.81	0.46	18.25	0.20	119.53	<b>0.06</b>	63.15	10.10	181.82	<b>0.36</b>	81.85	<b>0.13</b>	40.81	1.24	48.81	9.33	71.70
Ours	2.0°	✓	<b>0.02</b>	2.55	0.17	19.07	<b>0.03</b>	3.11	0.09	2.54	<b>1.04</b>	28.33	<b>0.36</b>	6.26	<u>0.14</u>	5.59	<u>0.10</u>	6.54	9.37	2.33
	2.0°	✗	<b>0.02</b>	7.86	0.42	6.39	<u>0.04</u>	3.21	0.10	7.28	1.94	6.19	0.38	28.34	<u>0.14</u>	11.51	0.74	3.43	9.56	4.90
	3.5°	✓	<b>0.02</b>	1.27	<b>0.11</b>	8.03	<u>0.04</u>	1.50	0.09	1.53	<u>1.07</u>	18.81	0.38	<b>3.51</b>	<u>0.14</u>	2.66	<b>0.09</b>	2.68	9.49	1.40
	3.5°	✗	0.20	3.16	0.91	2.91	0.16	1.54	0.10	3.19	2.17	<u>4.39</u>	0.46	8.21	<u>0.14</u>	6.50	0.87	1.77	<u>5.37</u>	<u>0.80</u>
	5.0°	✓	<b>0.02</b>	<u>0.88</u>	<u>0.15</u>	<u>2.76</u>	0.51	<u>1.24</u>	0.39	<u>1.04</u>	1.75	7.29	0.55	5.80	<b>0.13</b>	<u>1.92</u>	0.16	<b>1.49</b>	9.62	0.84
	5.0°	✗	0.17	<b>0.82</b>	1.04	<b>2.23</b>	0.51	<b>1.10</b>	0.40	<b>0.97</b>	2.51	<b>3.51</b>	0.39	4.89	<u>0.14</u>	1.97	0.24	3.40	6.78	<b>0.68</b>

Table 1. **Mean absolute depth error (MADE, abbreviated as Err) [mm] and total execution time (abbreviated as  $t$ ) [s] on the DiLiGenT benchmark [22].** For each object, **bold** and underlined denote respectively the best and the second-best result across the methods. All experiments use  $\Delta E_{\max} = 10^{-3}$ ,  $T = 150$ , and 8-connectivity, without merging. D and ND denote the version of [19] with and without discontinuity computation, respectively ( $\alpha_{b \rightarrow a}$  in [19]). \*This object contains a full depth discontinuity.





## References

- [1] Gary A. Atkinson and Edwin R. Hancock. [Recovery of Surface Orientation From Diffuse Polarization](#). *IEEE Trans. Image Processing*, 15:1653–1664, 2006. 1
- [2] Yunhao Ba, Alex Ross Gilbert, Franklin Wang, Jinfa Yang, Rui Chen, Yiqin Wang, Lei Yan, Boxin Shi, and Achuta Kadambi. [Deep Shape from Polarization](#). In *ECCV*, 2020. 1
- [3] Gwangbin Bae and Andrew J. Davison. [Rethinking Inductive Biases for Surface Normal Estimation](#). In *CVPR*, 2024. 7, 8, 14
- [4] Xu Cao, Boxin Shi, Fumio Okura, and Yasuyuki Matsushita. [Normal Integration via Inverse Plane Fitting with Minimum Point-to-Plane Distance](#). In *CVPR*, 2021. 2
- [5] Xu Cao, Hiroaki Santo, Boxin Shi, Fumio Okura, and Yasuyuki Matsushita. [Bilateral Normal Integration](#). In *ECCV*, 2022. 1, 2, 3, 5, 14, 16
- [6] cara fealy choate. Image “Winter Wedding Cake”. [https://commons.wikimedia.org/wiki/File:Winter\\_Wedding\\_Cake.jpg](https://commons.wikimedia.org/wiki/File:Winter_Wedding_Cake.jpg), 2007. CC Attribution 2.0 Generic. 8
- [7] ChristyHsu. Mesh “Cozy living room baked”. <https://sketchfab.com/3d-models/cozy-living-room-baked-581238dc5fda4dc990571cdc02827783>, 2022. CC BY 4.0. 8
- [8] Maximilian Denninger, Dominik Winkelbauer, Martin Sundermeyer, Wout Boerdijk, Markus Knauer, Klaus H. Strobl, Matthias Humt, and Rudolph Triebel. [BlenderProc2: A Procedural Pipeline for Photorealistic Rendering](#). *Journal of Open Source Software*, 8(82):4901, 2023. 7
- [9] dylanheyes. Mesh “Cozy Modern Bedroom”. <https://sketchfab.com/3d-models/cozy-modern-bedroom-ea25adc53f514eaba9dd043f5eac9c77>, 2023. CC BY 4.0. 8
- [10] Moritz Heep and Eduard Zell. [An Adaptive Screen-Space Meshing Approach for Normal Integration](#). In *ECCV*, 2024. 2
- [11] Magnus R. Hestenes and Eduard Stiefel. [Methods of Conjugate Gradients for Solving Linear Systems](#). *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952. 5
- [12] Berthold K.P Horn. [Obtaining Shape from Shading Information](#). *The Psychology of Computer Vision*, pages 115–155, 1975. 1
- [13] Satoshi Ikehata. [Scalable, Detailed and Mask-free Universal Photometric Stereo](#). In *CVPR*, 2023. 1, 7, 8
- [14] Jay. Image “Cake Pops”. [https://commons.wikimedia.org/wiki/File:Cake\\_Pops\\_\(8445856654\).jpg](https://commons.wikimedia.org/wiki/File:Cake_Pops_(8445856654).jpg), 2013. CC Attribution 2.0 Generic. 17
- [15] Bilge Karaçalı and Wesley Snyder. [Reconstructing discontinuous surfaces from a given gradient field using partial integrability](#). *Computer Vision and Image Understanding*, 92(1):78–111, 2003. 2
- [16] Hyomin Kim, Yucheol Jung, and Seungyong Lee. [Discontinuity-preserving Normal Integration with Auxiliary Edges](#). In *CVPR*, 2024. 1
- [17] Iakawaka. Mesh “Office Props Lowpoly”. <https://sketchfab.com/3d-models/office-props-lowpoly-36b97aeac9fc46499262c036eea265f3>, 2021. CC Attribution. 3
- [18] David Marr. [Analysis of occluding contour](#). *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 197(1129):441–475, 1977. 3
- [19] Francesco Milano, Manuel López-Antequera, Naina Dhingra, Roland Siegwart, and Robert Thiel. [Discontinuity-aware Normal Integration for Generic Central Camera Models](#). In *ICCV*, 2025. 1, 2, 3, 5, 6, 7, 8
- [20] Yvain Quéau, Jean-Denis Durou, and Jean-François Aujol. [Normal Integration: A Survey](#). *Journal of Mathematical Imaging and Vision*, 60:576–593, 2018. 2
- [21] Yvain Quéau, Jean-Denis Durou, and Jean-François Aujol. [Variational Methods for Normal Integration](#). *Journal of Mathematical Imaging and Vision*, 60:609–632, 2018. 2
- [22] Boxin Shi, Zhe Wu, Zhipeng Mo, Dinglong Duan, Sai-Kit Yeung, and Ping Tan. [A Benchmark Dataset and Evaluation for Non-Lambertian and Uncalibrated Photometric Stereo](#). In *CVPR*, 2016. 1, 3, 6, 8, 11, 12, 13, 15, 16
- [23] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. [SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python](#). *Nature Methods*, 17: 261–272, 2020. 6
- [24] Yinting Wang, Jiajun Bu, Na Li, Mingli Song, and Ping Tan. [Detecting Discontinuities for Surface Reconstruction](#). In *ICPR*, 2012. 2
- [25] Robert J. Woodham. [Photometric Method For Determining Surface Orientation From Multiple Images](#). *Optical Engineering*, 19(1):139–144, 1980. 1
- [26] Tai-Pang Wu and Chi-Keung Tang. [Visible Surface Reconstruction from Normals with Discontinuity Consideration](#). In *CVPR*, 2006. 2
- [27] Wuyuan Xie, Miaohui Wang, Mingqiang Wei, Jianmin Jiang, and Jing Qin. [Surface Reconstruction From Normals: A Robust DGP-Based Discontinuity Preservation Approach](#). In *CVPR*, 2019. 2
- [28] Dizhong Zhu and William A. P. Smith. [Least Squares Surface Reconstruction on Arbitrary Domains](#). In *ECCV*, 2020. 2

## Supplementary Material

The following Sections constitute the Supplementary Material. Section A provides a pseudocode of our method. In Sec. B, a detailed profiling of our run time is presented. Section C investigates the effect of different hyperparameters for our outlier reweighting. Section D studies the impact of the convergence parameters. Section E provides visualizations of our continuous components on the DiLiGenT dataset. In Sec. F we analyze the impact of our merging operation on reconstruction error on the DiLiGenT dataset. Section G provides an ablation on the pixel connectivity used by our method. Finally, in Sec. H we discuss the limitations of our method.

### A. Pseudocode of our method

A pseudocode for our method is shown in Algorithm 1.

---

#### Algorithm 1 Pseudocode of our method.

---

**Require:**  $\theta_c$ , can merge (bool),  $\text{freq}_{\text{merging}}$ ,  $\Delta E_{\text{max}}$ ,  $T$ .

- 1: Initialize  $\tilde{\mathbf{z}} \leftarrow \mathbf{0}$
- 2: Form components  $\{\mathcal{C}_c^{(0)}\}$  based on  $\theta_{a,b} < \theta_c$
- 3: Compute intra-component matrices  
 $\mathbf{A}_c, \mathbf{b}_c, \mathbf{W}_c = \text{diag}(\{W_{b \rightarrow a}\}_{(a,b) \in E(\mathcal{C}_c^{(0)})})$  (16)
- 4: Fill each component  $\mathcal{C}_c$  in parallel:  
 $\tilde{\mathbf{z}}_c^{(0)} \leftarrow \text{cg}(\mathbf{A}_c^\top \mathbf{W}_c \mathbf{A}_c, \mathbf{A}_c^\top \mathbf{W}_c \mathbf{b}_c)$
- 5: converged  $\leftarrow \text{False}$ ,  $m \leftarrow 0$ ,  $t \leftarrow 0$ ,  $E_0 \leftarrow \epsilon$
- 6: Form inter-component matrices  $\bar{\mathbf{A}}_0, \bar{\mathbf{b}}_0$  (18)
- 7: **while** not converged **do**  $\triangleright$  Relative-scale optimization
- 8:   **if**  $t \leq 1$  **then**  $\triangleright$  Alignment optimization
- 9:     Uniform weights:  $\bar{\mathbf{W}}_m^{(t)} \leftarrow \text{diag}(1)$
- 10:   **else**  $\triangleright$  Discontinuity-aware optimization
- 11:     BiNI weights with outlier reweighting (20):  
 $\bar{\mathbf{W}}_m^{(t)} \leftarrow \text{diag}(\{W_{b \rightarrow a} \cdot W_{b \rightarrow a}^{\text{out}}\}_{(a,b) \in E_{\text{inter}}^{(m)}})$
- 12:   **end if**
- 13:    $\tilde{\mathbf{s}}^{(t)} \leftarrow \text{cg}(\bar{\mathbf{A}}_m^\top \bar{\mathbf{W}}_m^{(t)} \bar{\mathbf{A}}_m, \bar{\mathbf{A}}_m^\top \bar{\mathbf{W}}_m^{(t)} \bar{\mathbf{b}}_m)$
- 14:   Scale components (in parallel, via broadcasting):  
 $\forall c \in \{0, \dots, |\mathcal{C}^{(m)}| - 1\}, \tilde{\mathbf{z}}_c^{(t+1)} \leftarrow \tilde{\mathbf{z}}_c^{(t)} + \tilde{s}_c^{(t)} \mathbf{1}$
- 15:    $t \leftarrow t + 1$
- 16:   **if** can merge  $\wedge (t \equiv 0 \pmod{\text{freq}_{\text{merging}}})$  **then**
- 17:      $\forall \mathcal{C}_c^{(m)}, (\hat{a}, \hat{b})_c \leftarrow \arg \min_{(a,b) \in \partial \mathcal{C}_c^{(m)}} |\bar{\chi}_{b \rightarrow a}|$
- 18:     Compute subgraph  $\hat{\mathcal{Q}}_m \leftarrow (\mathcal{C}^{(m)}, \{(\hat{a}, \hat{b})_c^{(m)}\})$
- 19:      $\{\mathcal{C}_c^{(m+1)}\} \leftarrow \text{connected components}(\hat{\mathcal{Q}}_m)$
- 20:      $m \leftarrow m + 1$
- 21:     Form inter-component matrices  $\bar{\mathbf{A}}_m, \bar{\mathbf{b}}_m$  (18)
- 22:   **end if**
- 23:    $E_t \leftarrow (\bar{\mathbf{A}}_m \tilde{\mathbf{s}}^{(t)} - \bar{\mathbf{b}}_m)^\top \bar{\mathbf{W}}_m^{(t)} (\bar{\mathbf{A}}_m \tilde{\mathbf{s}}^{(t)} - \bar{\mathbf{b}}_m)$
- 24:   converged  $\leftarrow \frac{|E_t - E_{t-1}|}{E_{t-1}} < \Delta E_{\text{max}} \vee t = T$
- 25: **end while**
- 26: **return**  $\tilde{\mathbf{z}}$

---

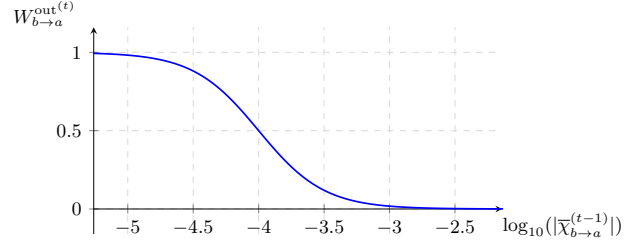


Figure 5. **Outlier reweighting (20) as a function of  $\log_{10}(|\bar{\chi}_{b \rightarrow a}^{(t-1)}|)$ , for  $L = 10^{-3}$  and  $U = 10^{-5}$ .**

### B. Detailed profiling of our run time

We provide a detailed profiling of the run time of our method for the parameter configuration used in our main experiments, both on the DiLiGenT dataset (Tab. 2) and on the large-scale normals maps of Fig. 4 (Tab. 3).

It is worth noting that, in both cases, two operations that account for a significant fraction of the total execution time are the two pre-processing steps of forming the intra-component matrices  $\mathbf{A}_c$ ,  $\mathbf{b}_c$ , and  $\mathbf{W}_c$  and filling the components. For the latter, we use the Python `joblib` library to parallelly execute multiple instances of per-component conjugate-gradient optimization. While this usually converges in few fractions of a second due to the smaller scale of the per-component optimization compared to the global optimization, larger resolutions might produce larger components, resulting in increased time for this initial step. Additionally, parallelization is capped by the number of processes that are available to the program (we set this to 4 in our experiments), thereby still requiring iterative processing. On the other hand, the formation step of the intra-component matrices  $\mathbf{A}_c$ ,  $\mathbf{b}_c$ , and  $\mathbf{W}_c$  is non-optimized in our current implementation, and alone contributes to a factor of up to respectively 39% (`coinskeyboard`) and 44% (`cow`) of the total execution time for the large-scale normal maps and the smaller-scale DiLiGenT dataset. The reason for the long time required to perform this step lies in the fact that the matrices  $\mathbf{A}_c$  and  $\mathbf{W}_c$  are represented in our implementation as sparse matrices (`scipy.sparse.csr_matrix`) of different shape, and as such cannot benefit from parallel-access optimized broadcasting operations. Implementation improvements in this direction are left to future work.

### C. Ablation on outlier reweighting

To complement the definition provided in the main paper, we provide an illustration of our outlier reweighting function in Fig. 5, using the parameters from the main experiments. Additionally, we ablate on different values for its hyperparameter  $L$  and also include a variant of the reweighting which introduces a *hard* outlier thresholding, so that equations with residual magnitude  $|\bar{\chi}_{b \rightarrow a}^{(t-1)}|$  are assigned weight

	bear	buddha	cat	cow	harvest	pot1	pot2	reading	goblet
Formation of $\mathcal{G}_0$	0.020	0.018	0.019	0.020	0.019	0.018	0.019	0.019	0.020
Computation of $\{\mathcal{C}_c^{(0)}\}$	0.092	0.090	0.091	0.087	0.091	0.097	0.089	0.088	0.094
Formation of $\mathbf{A}_c, \mathbf{b}_c, \mathbf{W}_c$	0.546	1.857	0.731	0.758	3.345	1.581	1.384	1.206	0.344
Filling of components	1.168	2.442	1.428	1.360	4.166	2.325	2.103	1.719	0.745
Iteration 0	1.174	2.587	1.439	1.368	4.311	2.342	2.118	1.734	0.775
Iteration 1	1.183	2.977	1.464	1.378	4.810	2.394	2.148	1.766	0.840
Iteration 2	1.193	3.490	1.484	1.391	5.747	2.471	2.187	1.822	0.858
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Convergence	1.270 [it. 8]	8.033 [it. 11]	1.500 [it. 3]	1.525 [it. 12]	18.805 [it. 17]	3.511 [it. 18]	2.655 [it. 17]	2.679 [it. 20]	1.398 [it. 49]

Table 2. **Profiling of our method on the the DiLiGenT benchmark [22].** Intermediate execution times from the start, after the completion of each step are reported [s].  $\theta_c = 3.5^\circ$ ,  $\Delta E_{\max} = 10^{-3}$ ,  $T = 150$ , 8-connectivity, and outlier reweighting are used, without merging.

	bedroom	livingroom	coinskeyboard	schooldesk	seafloor	cake
Formation of $\mathcal{G}_0$	0.019	0.058	0.058	0.062	0.050	0.020
Computation of $\{\mathcal{C}_c^{(0)}\}$	0.138	0.522	0.375	0.447	0.395	0.147
Formation of $\mathbf{A}_c, \mathbf{b}_c, \mathbf{W}_c$	1.554	12.226	42.058	5.759	15.663	4.486
Filling of components	3.804	50.536	51.855	69.922	29.548	8.240
Iteration 0	3.955	51.232	54.343	70.075	29.919	8.382
Iteration 1	4.290	55.196	60.506	70.448	31.546	8.890
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Merging $m = 0$	5.876	92.018	96.427	75.205	51.690	12.382
Merging $m = 1$	6.198	102.327	102.591	75.783	57.139	12.883
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Convergence	6.796	105.081	104.630	76.648	58.524	13.374

Table 3. **Profiling of our method on the the large-scale normal maps from Fig. 4.** Columns 1 to 6 in Fig. 4 are referred to, from left to right, as: bedroom, livingroom, coinskeyboard, schooldesk, seafloor, cake. Intermediate execution times from the start, after the completion of each step are reported [s].  $\theta_c = 2.0^\circ$ ,  $\Delta E_{\max} = 10^{-3}$ ,  $T = 15$ , 8-connectivity,  $\text{freq}_{\text{merging}} = 5$ , and outlier reweighting are used.

Reweighting type	$U$	$\theta_c$	bear	buddha	cat	cow	harvest	pot1	pot2	reading	goblet
Soft	$10^{-5}$	None	0.02	0.13	0.03	0.09	1.35	0.40	0.14	0.19	9.37
		$2.0^\circ$	0.02	0.17	0.03	0.09	1.02	0.37	0.14	0.20	9.35
		$3.5^\circ$	0.02	0.10	0.04	0.10	1.10	0.39	0.14	0.10	8.08
		$5.0^\circ$	0.02	0.15	0.51	0.39	1.61	0.55	0.14	0.17	4.45
Soft	$10^{-4}$	None	0.02	0.20	0.03	0.09	1.31	0.36	0.13	0.17	9.41
		$2.0^\circ$	0.02	0.17	0.03	0.09	1.04	0.36	0.14	0.10	9.37
		$3.5^\circ$	0.02	0.11	0.04	0.09	1.07	0.38	0.14	0.09	9.49
		$5.0^\circ$	0.02	0.15	0.51	0.39	1.75	0.55	0.13	0.16	9.62
Soft	$10^{-6}$	None	0.02	0.17	0.03	0.09	0.99	0.36	0.13	0.13	9.41
		$2.0^\circ$	0.02	0.13	0.03	0.09	0.88	0.36	0.14	0.10	9.40
		$3.5^\circ$	0.02	0.17	0.04	0.09	1.18	0.38	0.13	0.10	7.53
		$5.0^\circ$	0.02	0.18	0.51	0.39	1.70	0.55	0.13	0.15	9.62
Hard	N/A	None	0.03	0.27	0.05	0.09	1.78	0.41	0.13	0.19	9.33
		$2.0^\circ$	0.02	0.42	0.05	0.09	1.13	0.39	0.14	0.22	8.19
		$3.5^\circ$	0.02	0.36	0.05	0.10	1.09	0.40	0.14	0.19	3.54
		$5.0^\circ$	0.02	0.16	0.51	0.39	1.42	0.38	0.14	0.24	4.42

Table 4. **Ablation on the outlier reweighting mechanism on the DiLiGenT benchmark [22].** The mean absolute depth error (MADE) [mm] of our method is reported. The upper outlier reweighting threshold  $U$  is set to  $10^{-3}$ , and soft threshold with different lower thresholds  $L$  as well as hard thresholding based on  $U$  are compared. All experiments use convergence criteria  $\Delta E_{\max} = 10^{-3}$  and  $T = 150$ , 8-pixel connectivity, without merging.

$W_{b \rightarrow a}^{\text{out}^{(t)}} = 0$  if  $|\bar{\chi}_{b \rightarrow a}^{(t-1)}| \geq U$  and weight  $W_{b \rightarrow a}^{\text{out}^{(t)}} = 1$  if  $|\bar{\chi}_{b \rightarrow a}^{(t-1)}| < U$ , where we set  $U = 10^{-3}$ .

Table 4 reports the results of the ablation. Overall, the differences across the variants for outlier reweighting are marginal for most objects. However, for objects with larger discontinuities (buddha, harvest, reading)

soft reweighting achieves generally better accuracy, with a small degree of object specificity for the value of  $L$ , indicating that it might be effective particularly in controlling outlier residuals that arise due to discontinuities.

$\Delta E_{\max}$	$T$	$\theta_c$	bear		buddha		cat		cow		harvest		pot1		pot2		reading		goblet*	
			Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$
$10^{-3}$	5	None	0.02	4.08	0.51	10.56	0.03	5.17	0.09	2.24	1.77	17.40	0.39	9.49	0.13	4.23	0.17	4.74	9.42	5.54
		2.0°	0.02	2.49	0.18	7.92	0.03	2.91	0.09	2.37	1.24	11.65	0.38	5.26	0.13	4.05	0.14	4.75	9.37	1.27
		3.5°	0.02	1.22	0.15	4.89	0.04	1.51	0.10	1.45	1.10	7.55	0.41	2.63	0.13	2.26	0.13	1.92	9.47	0.87
		5.0°	0.02	0.86	0.18	2.65	0.51	1.11	0.39	0.98	1.78	5.30	0.56	1.96	0.13	1.67	0.18	1.33	9.56	0.75
$10^{-3}$	15	None	0.02	6.09	0.20	17.69	0.03	9.17	0.09	3.92	1.31	46.33	0.36	22.12	0.13	9.64	0.17	4.31	9.41	5.15
		2.0°	0.02	2.55	0.17	17.08	0.03	3.11	0.09	2.50	1.05	28.62	0.36	6.27	0.14	5.40	0.10	7.01	9.37	1.58
		3.5°	0.02	1.28	0.11	7.62	0.04	1.52	0.09	1.54	1.06	16.82	0.38	3.27	0.14	2.61	0.09	2.43	9.46	0.99
		5.0°	0.02	0.89	0.15	2.75	0.51	1.22	0.39	1.04	1.75	7.87	0.55	2.26	0.13	1.93	0.16	1.49	9.62	0.86
$10^{-3}$	150	None	0.02	7.39	0.20	19.56	0.03	36.98	0.09	3.35	1.31	44.35	0.36	32.51	0.13	9.83	0.17	3.40	9.41	4.78
		2.0°	0.02	2.55	0.17	19.07	0.03	3.11	0.09	2.54	1.04	28.33	0.36	6.26	0.14	5.59	0.10	6.54	9.37	2.33
		3.5°	0.02	1.27	0.11	8.03	0.04	1.50	0.09	1.53	1.07	18.81	0.38	3.51	0.14	2.66	0.09	2.68	9.49	1.40
		5.0°	0.02	0.88	0.15	2.76	0.51	1.24	0.39	1.04	1.75	7.29	0.55	5.80	0.13	1.92	0.16	1.49	9.62	0.84
$10^{-5}$	5	None	0.02	3.43	0.51	7.43	0.03	4.85	0.09	2.54	1.77	17.24	0.39	8.65	0.13	2.93	0.17	4.29	9.42	5.52
		2.0°	0.02	2.46	0.18	7.55	0.03	2.94	0.09	2.31	1.24	12.28	0.38	5.18	0.13	3.97	0.14	4.78	9.37	1.28
		3.5°	0.02	1.24	0.15	4.93	0.04	1.52	0.10	1.42	1.10	7.98	0.41	2.61	0.13	2.29	0.13	1.95	9.47	0.87
		5.0°	0.02	0.85	0.18	2.65	0.51	1.12	0.39	0.98	1.78	5.18	0.56	1.97	0.13	1.67	0.18	1.35	9.56	0.81
$10^{-5}$	15	None	0.01	10.17	0.15	25.28	0.03	8.78	0.09	4.99	1.28	53.51	0.36	25.19	0.13	7.16	0.09	14.50	9.40	12.20
		2.0°	0.02	2.93	0.17	16.94	0.03	3.49	0.09	2.64	1.05	29.43	0.36	8.36	0.14	5.46	0.09	7.93	9.37	1.59
		3.5°	0.02	1.38	0.11	8.88	0.04	1.69	0.09	1.56	1.06	15.89	0.38	3.25	0.14	2.62	0.09	2.46	9.46	1.01
		5.0°	0.02	0.95	0.12	3.90	0.51	1.21	0.39	1.05	1.69	9.50	0.55	2.32	0.13	1.93	0.15	1.62	9.62	0.84
$10^{-5}$	150	None	0.01	11.20	0.19	253.51	0.03	42.47	0.09	12.51	1.32	387.01	0.36	169.22	0.13	80.35	0.10	66.33	9.41	66.55
		2.0°	0.02	3.04	0.17	137.36	0.03	4.97	0.09	2.86	1.03	336.65	0.36	78.28	0.14	12.60	0.10	73.13	9.37	6.63
		3.5°	0.02	1.60	0.11	52.42	0.04	2.88	0.09	1.93	1.18	98.90	0.38	11.63	0.14	4.24	0.09	10.85	9.49	3.32
		5.0°	0.02	1.10	0.12	14.57	0.51	2.08	0.39	1.36	1.69	121.54	0.55	7.42	0.13	2.64	0.16	5.83	9.62	2.44

Table 5. **Ablation on the convergence criteria  $\Delta E_{\max}$  and  $T$  on the DiLiGenT benchmark [22].** The mean absolute depth error (MADE, abbreviated as Err) [mm] and the total execution time (abbreviated as  $t$ ) [s] of our method are reported. All experiments use outlier reweighting  $W_{b \rightarrow a}^{\text{out}}$  (20) with  $L = 10^{-5}$  and  $U = 10^{-3}$ , without merging.

## D. Ablation on the convergence parameters

Table 5 reports the reconstruction error and run time achieved by our method for different values of the parameters  $\Delta E_{\max}$  and  $T$  that control the termination of its execution. We observe that with limited exceptions, mostly restricted to our pixel-level variant ( $\theta_c = \text{None}$ ), enforcing a stricter relative-energy convergence criterion ( $\Delta E_{\max} = 10^{-5}$ ) produces no significant changes in the accuracy of the reconstruction, while requiring a longer run time. Notably, the same conclusion applies to the maximum number of optimization steps, for which we find that our method effectively achieves convergence in 5 or at most 15 iterations for all objects. While earlier termination of the optimization results in a slight increase in the running time, this speedup is not particularly significant for the small-scale of the normal maps from DiLiGenT, especially in the case of component-based (rather than pixel-level) optimization. For our main experiments on the DiLiGenT benchmark (*cf.* main paper), we therefore chose to use  $T = 150$ , to enable convergence for the baselines without outlier reweighting.

## E. Ablation on the threshold for component formation

Figure 6 shows the continuous components identified by our heuristic on the DiLiGenT benchmark, for the different values of the normal similarity threshold  $\theta_c$  that we use in our experiments.

We note that for each decomposition it is possible

to compute the minimum theoretical reconstruction error that could be achieved by the relative scale optimization, which coincides with the global mean average depth error (MADE) that would be attained if the optimal combination of scales was applied to the individual reconstructions formed in the component filling stage. In general, each of such per-component reconstructions introduces an error, for however small, in its corresponding surface region. This is due to the approximations inherent in the models of continuity and discontinuity, as well as to the convergence of the optimization. To compute the minimum MADE, it is sufficient to compute the sum of the per-component MADEs, weighted by the number of pixels in each component.

As shown in Fig. 6, choosing a smaller threshold for  $\theta_c$  results in a smaller minimum theoretical MADE, at the cost of a larger number of components. This is coherent with the fact that the minimum theoretical MADE decreases as the number of components approaches the total number of pixels, with a minimum value of 0 in the limit of per-pixel components, since by definition a perfect reconstruction could be achieved by appropriately scaling the depth of each pixel.

Importantly, we note that for all objects in the benchmark the minimum theoretical MADE achievable by our method is significantly smaller than the accuracy reached by state-of-the-art methods, by different margins depending on the exact value of  $\theta_c$ . On the one hand, this validates the effectiveness of our component formation and filling, which does not compromise the best quality that the reconstruction can achieve. On the other hand, the remaining gap between



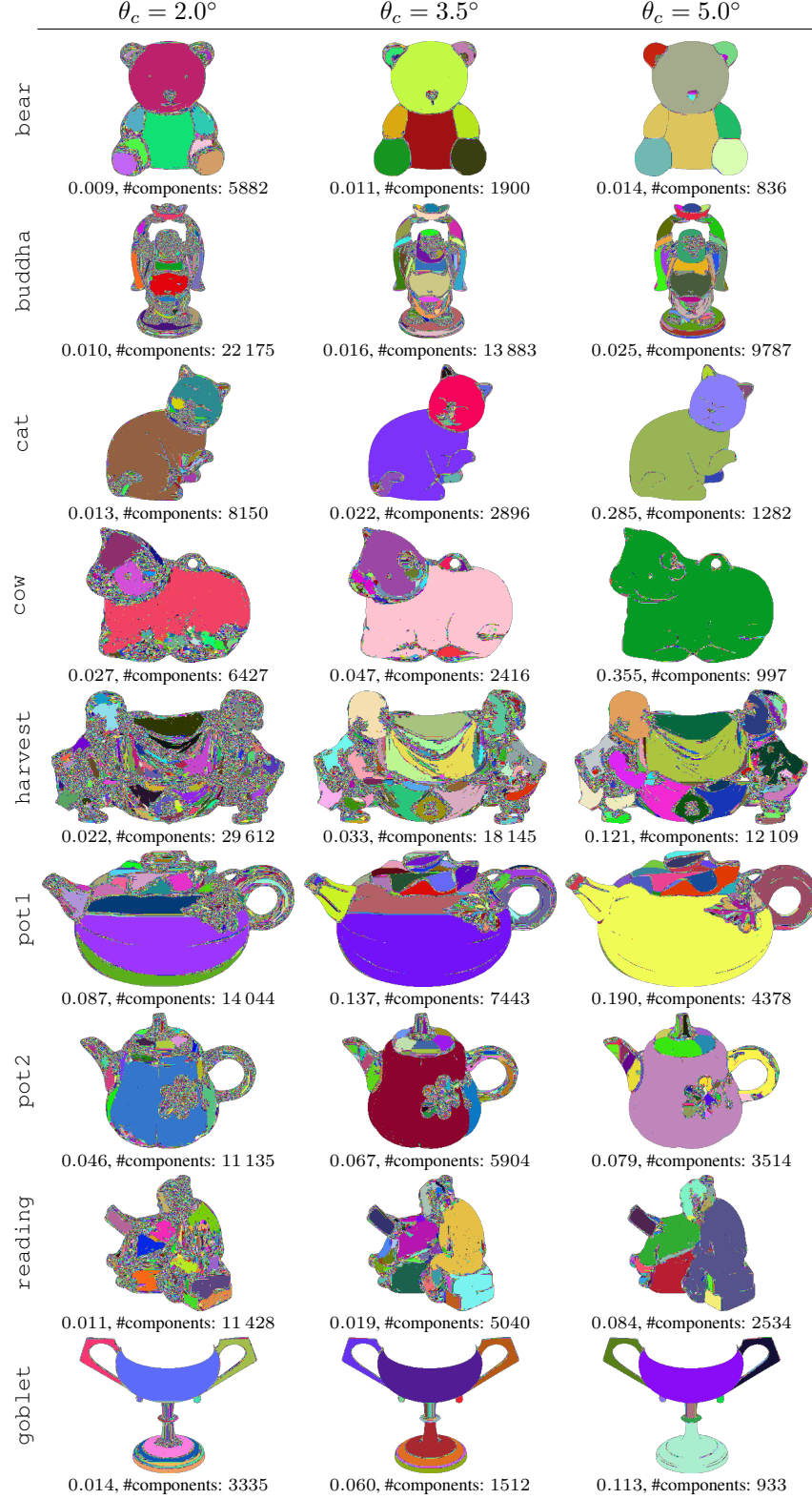


Figure 6. **Continuous components identified by our method for different normal similarity thresholds  $\theta_c$  and 8– connectivity, DiLiGenT benchmark [22].** For each object and threshold, different colors indicate different components. Below each component image are: the minimum mean average depth error (MADE, in mm) that can be theoretically achieved by scaling the continuous components; the number of components.

the minimum theoretical MADE and the MADE actually achieved by our optimization presents an opportunity for the future emergence of more accurate models of continuity and discontinuity, which could lead to even more optimal relative scale optimization.

## F. Ablation on the effect of merging in the DiLiGenT benchmark

Figure 7 shows the progression of the component decomposition, of the minimum theoretical MADE, and of the actual MADE computed from the reconstruction, at different stages of our optional merging process. The illustrated example uses a merging frequency of 5 optimization iterations; as previously noted (*cf.* Sec. D and Tab. 5), on the small-scale normals from DiLiGenT our method achieves convergence for most object already after this number of iterations. This is reflected in the fact that the reconstruction error (indicated within brackets in Fig. 7) does not change significantly after the first merge operation for most objects. However, small improvements can be observed for objects with larger discontinuities (buddha, harvest, reading). This indicates that in the presence of discontinuities optimization requires a larger number of iterations to converge. For this reason, merging can be a viable option to reduce the size of the optimization problem (hence also the execution time of later iterations) while allowing the convergence process to continue. As already observed (Sec. D), while the computational effect of this operation might be negligible for small-scale normal maps, its impact becomes significantly more important for larger-scale normal maps, for which a reduction in the number of variables can largely reduce the run time of the optimization (*cf.* Fig. 4).

We note, finally, that merging in general increases the minimum theoretical MADE, since it “fixes” the relative scale of neighboring components to a value that in general does not coincide with its optimal one. However, we stress that the merging operation per se does not increase the reconstruction error with respect to the previous optimization steps. This is because it does not alter the relative scales to which the optimization had converged at the preceding step, but simply relabels pixels in different components so that their scale is jointly optimized in subsequent iterations.

## G. Ablation on pixel connectivity

We ablate the impact of the chosen pixel connectivity on the DiLiGenT benchmark, comparing 8- connectivity, which we use in our main experiments, with 4- connectivity. For a given configuration, we use the connectivity both in the component detection/filling stage and in the subsequent relative scale optimization. The results of the ablation are shown in Tab. 6. Overall, no major differences emerge be-

tween the two connectivities, with comparable accuracies and runtimes across all objects. A minor exception is to be found when using normal similarity threshold  $\theta_c = 5.0^\circ$ , for which for some objects (cat, harvest, reading) 4- connectivity achieves better accuracy by a significant margin.

## H. Limitations

**Component formation and model of discontinuity.** Since our heuristic for component formation relies on the similarity between neighboring normals, if the normals are continuous across a surface discontinuity it cannot detect disconnected surface regions as separate components, as depicted in the example of Fig. 8. We note, however, that this example represents a corner case more in general of normal integration methods, as previously noted for instance by [5] (Fig. 14, Supplementary). In particular, in this case the integration problem itself is ill-posed, since in this setting it is not possible to determine whether a discontinuity is present from the normal map alone.

A similar issue arises in the case of overly-smooth input normals, with indistinct boundaries, which sometimes occur in normal maps produced as output by learning-based approaches for normal estimation, such as DSINE [3]. For these normals maps, our heuristic for component formation might sometimes merge multiple surfaces into a single component, depending on the similarity threshold  $\theta_c$  (Fig. 9). We note, however, that even for an incorrect component decomposition (*i.e.*, that merges surface regions separated by a discontinuity) the corresponding reconstruction can in theory still correctly capture the discontinuity if the model of discontinuity is able to describe it. This is because in the initial phase each surface patch is reconstructed using the same model of discontinuity deployed for relative scale optimization. Viceversa, if the model of discontinuity is unable to capture the discontinuity, as is the case for full discontinuities such as the foreground-background boundaries in Fig. 9, the general problem of retrieving the scale of surface regions across such discontinuities cannot be addressed without additional knowledge.

An interesting future direction is to explore alternative, more sophisticated heuristics for component formation, for instance through the use of learning-based methods that could learn priors over the discontinuities from the normal maps. In the more general case, additional input or knowledge available depending on the setting might be incorporated in the component formation phase. For instance, if an input color image was provided, as is the case in photometric stereo or surface normal estimation, edges might be extracted from the image and composed with the heuristic based only on surface normals.

**Decreased benefit for highly non-smooth normal maps.** The computational advantage resulting from our method

	$m = 0$	$m = 1$	$m = 2$	$m = \lfloor \frac{M_{\text{tot}}}{2} \rfloor$	$m = M_{\text{tot}} - 1$	$M_{\text{tot}}$
bear	 0.011 ( <u>0.018</u> ) #components: 1900	 0.013 ( <u>0.018</u> ) #components: 361	 0.014 ( <u>0.018</u> ) #components: 47	...	 0.016 ( <u>0.018</u> ) #components: 7	4
buddha	 0.016 ( <u>0.148</u> ) #components: 13 883	 0.049 ( <u>0.116</u> ) #components: 2904	 0.063 ( <u>0.115</u> ) #components: 476	...	 0.089 ( <u>0.115</u> ) #components: 4	6
cat	 0.022 ( <u>0.044</u> ) #components: 2896	 0.029 ( <u>0.044</u> ) #components: 568	 0.032 ( <u>0.044</u> ) #components: 76	...	 0.036 ( <u>0.044</u> ) #components: 2	5
cow	 0.047 ( <u>0.096</u> ) #components: 2416	 0.054 ( <u>0.095</u> ) #components: 455	 0.062 ( <u>0.095</u> ) #components: 57	...	 0.067 ( <u>0.095</u> ) #components: 5	4
harvest	 0.033 ( <u>1.095</u> ) #components: 18 145	 0.100 ( <u>1.086</u> ) #components: 3867	 0.123 ( <u>1.079</u> ) #components: 677	...	 0.641 ( <u>1.079</u> ) #components: 4	6
pot1	 0.137 ( <u>0.409</u> ) #components: 7443	 0.178 ( <u>0.395</u> ) #components: 1408	 0.211 ( <u>0.394</u> ) #components: 216	...	 0.386 ( <u>0.395</u> ) #components: 3	6
pot2	 0.067 ( <u>0.135</u> ) #components: 5904	 0.084 ( <u>0.135</u> ) #components: 1224	 0.094 ( <u>0.135</u> ) #components: 181	...	 0.131 ( <u>0.135</u> ) #components: 4	5
reading	 0.019 ( <u>0.132</u> ) #components: 5040	 0.044 ( <u>0.111</u> ) #components: 1006	 0.055 ( <u>0.106</u> ) #components: 148	...	 0.102 ( <u>0.106</u> ) #components: 2	6
goblet	 0.060 ( <u>9.471</u> ) #components: 1521	 0.086 ( <u>9.499</u> ) #components: 244	 0.386 ( <u>9.499</u> ) #components: 28	...	 1.095 ( <u>9.499</u> ) #components: 3	5

Figure 7. **Continuous components at different stages  $m$  of the merging process, DiLiGenT benchmark [22].** For each object and threshold, different colors indicate different components. Below each component image are: *first row*: the minimum mean average depth error (MADE, in mm) that can be theoretically achieved by scaling the continuous components and the MADE (mm) achieved by the optimization at the end of the merging stage, the latter in brackets and underlined; *second row*: number of components. For each object,  $M_{\text{tot}}$  denotes the total number of merging operations required to obtain a single component. For all objects, normal similarity threshold  $\theta_c = 3.5^\circ$  and 8- connectivity are used, with  $\text{freq}_{\text{merging}} = 5$  and  $\Delta E_{\text{max}} = 10^{-3}$ .

$\theta_c$	Conn.	bear		buddha		cat		cow		harvest		pot1		pot2		reading		goblet	
		Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$	Err	$t$
None	8	0.02	7.39	0.20	19.56	0.03	36.98	0.09	3.35	1.31	44.35	0.36	32.51	0.13	9.83	0.17	3.40	9.41	4.78
2.0°	8	0.02	2.55	0.17	19.07	0.03	3.11	0.09	2.54	1.04	28.33	0.36	6.26	0.14	5.59	0.10	6.54	9.37	2.33
3.5°	8	0.02	1.27	0.11	8.03	0.04	1.50	0.09	1.53	1.07	18.81	0.38	3.51	0.14	2.66	0.09	2.68	9.49	1.40
5.0°	8	0.02	0.88	0.15	2.76	0.51	1.24	0.39	1.04	1.75	7.29	0.55	5.80	0.13	1.92	0.16	1.49	9.62	0.84
None	4	0.03	15.52	0.13	40.19	0.03	16.58	0.08	4.19	1.26	64.00	0.37	209.27	0.13	7.69	0.08	7.73	9.45	111.51
2.0°	4	0.03	2.98	0.12	26.65	0.03	9.15	0.09	2.97	1.09	29.39	0.38	8.29	0.14	27.20	0.09	17.03	9.27	1.89
3.5°	4	0.04	1.18	0.14	38.11	0.04	1.65	0.09	2.58	1.09	17.91	0.35	25.06	0.14	2.37	0.08	2.80	9.43	0.96
5.0°	4	0.02	0.77	0.12	4.46	0.03	1.23	0.09	1.14	0.80	11.11	0.57	7.72	0.13	4.84	0.10	2.21	9.51	0.73

Table 6. **Ablation on the pixel connectivity (abbreviated as Conn.) on the DiLiGenT benchmark [22].** The mean absolute depth error (MADE, abbreviated as Err) [mm] and the total execution time (abbreviated as  $t$ ) [s] of our method are reported. All experiments use outlier reweighting  $W_{b \rightarrow a}^{\text{out}}$  (20) with  $L = 10^{-5}$  and  $U = 10^{-3}$ , convergence criteria  $\Delta E_{\max} = 10^{-3}$  and  $T = 150$ , without merging.

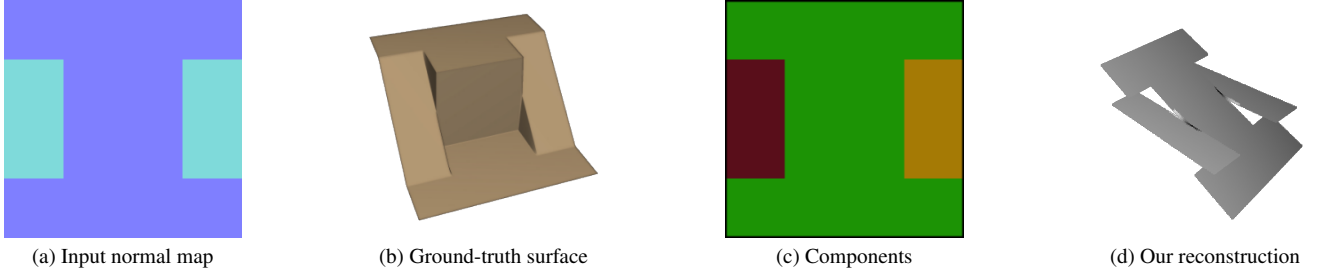


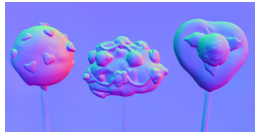
Figure 8. **Example corner case not handled by our component-formation heuristic.** When the input normals are continuous across a surface discontinuity, our heuristic for component formation cannot detect separate components. Since the model of discontinuity that we adopt [5] cannot recover discontinuities under the same corner case, the discontinuity will not be incorporated during the component filling and the two incorrectly merged pieces of surface will jointly adjust their scale in subsequent steps of the optimization. *Source of the input normal map and ground-truth surface visualization:* [5] (Fig. 14, Supplementary).

reducing the size of the optimization problem through the use of components is partially reduced when the input normal map is highly non-smooth. In particular, if the normal vectors vary often between neighboring pixels in an area of the input map, as for instance in the case of finely-textured regions, many single-pixel components may arise (*cf.*, *e.g.*, the woven reed basket in the second column of Fig. 4). While our method can still be applied in such a setting, its exact computational advantage will depend on the overall balance between smooth and non-smooth regions. An interesting future direction is to design heuristics for component formation that could reduce the occurrence of such single-pixel components, for instance by detecting that highly-textured areas belong to the same connected surface region, through learned priors. Additionally, we note that in many cases the number of components is greatly increased by single-pixel components that occur at the boundaries of larger components (*cf.*, *e.g.*,  $m = 0$  and  $m = 1$  in Fig. 7). For such cases, postprocessing of the initial decomposition could be explored, for instance by merging small components into larger ones without causing the larger ones to collapse into one another.

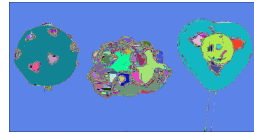




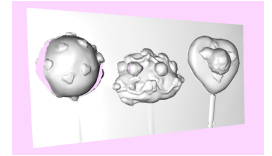
(a) Input color image



(b) Normals from DSINE



(c) Components



(d) Our reconstruction

Figure 9. **Example limitation due to overly-smooth input normal map.** In the case of overly-smooth input normal maps, our heuristic for component formation cannot detect separate components across the non-sharp boundaries, as is the case in this example of the boundary between the sticks in the foreground and the background. As a consequence, the foreground object is partly merged into the background. *Source of input color image:* [14].