Multi-Scale Integration of Slope Data on an Irregular Mesh

R. Saracchini *J. Stolfi* *H. Leitão*
G. Atkinson *M. Smith*

Technical Report - IC-11-11 - Relatório Técnico

May - 2011 - Maio

Multi-Scale Integration of Slope Data on an Irregular Mesh

Rafael F. V. Saracchini Jorge Stolfi [*]

Helena C. G. Leitão [†]Gary A. Atkinson Melvyn L. Smith [‡]

02-05-2011

Abstract

We describe a fast and robust method for computing scene depths (or heights) from surface gradient (or surface normal) data, such as would be obtained by photometric stereo or interferometry. Our method allows for uncertain or missing samples, which are often present in experimentally measured gradient maps; for sharp discontinuities in scene's depth, e .g. along object silhouette edges; and for irregularly spaced sampling points. To accomodate these features of the problem, we introduce an original and flexible representation of slope data, the weigth-delta mesh. Like other state of the art solutions, our algorithm reduces the problem to a system of linear equations that is solved by Gauss-Seidel iteration with multi-scale acceleration. Tests with various synthetic and measured gradient data show that our algorithm is as accurate and efficient as the best available integrators for uniformly sampled data. Moreover, thanks to the use of the weight-delta mesh representation, our algorithm remains accurate and efficient even for large sets of weakly-connected data, which cannot be efficiently handled by any existing algorithm.

1 Introduction

The *integration of a gradient map* to yield a height map is a computational problem that arises in several computer vision contexts, such as shape-from-shading [11, 10] and multiple-light photometric stereo [12, 24]. These methods usually determine the mean surface normal vector within each image pixel, from which one can obtain the height gradient (the partial derivatives of the surface's height Z with respect to the spatial coordinates X and Y). Although this information alone does not determine the absolute surface heights, it can yield height differences between parts of the same surface. This relative height information is sufficient for many important applications, such as industrial quality control [19], pottery fragment reassembly [13], surveillance and customs inspections [20], face recognition [9], and many others.

[*]Institute of Computing, State University of Campinas, Brazil

[†]Institute of Computing, Federal Fluminense University, Brazil

[‡]Machine Vision Laboratory, University of West England, United Kingdom

1.1 Computational difficulties

In practical contexts, the problem of computing the heights from given slopes faces at least three difficulties. First, the gradient data is usually *discretized*, that is, given as a finite set of *gradient samples*, each being an average of the gradient ∇Z over some neighbourhood of a *gradient sampling point*. Therefore, the height function cannot be precisely determined. It can only be approximated by a member of a finite-dimensional function space, defined by a finite function basis. Such a function can be (and usually is) uniquely represented by a finite set of discrete *height samples*, each being the average of the height Z over some neighbourhood of a *height sampling point*.

Second, the gradient data is usually contaminated with *noise* arising from unavoidable measurement, quantization, and computation errors. At some points, the expected magnitude of the error may be so high that the gradient is essentially unknown; and this may happen over large regions of the domain D. In the case of photometric stereo and shape-from-shading, for example, these regions include any pixels where the scene's surface is affected by shadows or specular highlighs, too dark, or poorly illuminated. Gaps (or large errors) in the data will also arise wherever the height or gradient functions are poorly defined, e.g. where the scene is highly porous, covered with hair-like structures, or transparent.

Third, the height function $Z(X, Y)$ of a real scene is usually *discontinuous*. In particular, it almost always has step-like discontinuities, or *cliffs*, at the edges of solid objects. At any sampling point that straddles those cliffs, photometric stereo and other gradient acquisition techniques usually fail to detect the (very large) gradient across the edge, and return an incorrect gradient sample that gives no clue as to the height of the cliff. See figure 1.

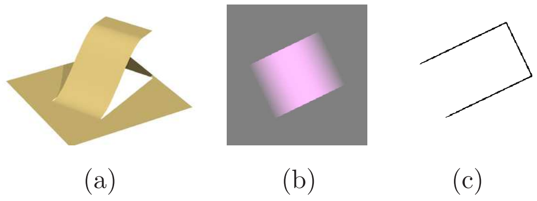

Figure 1: A height map with cliff-like discontinuities (a), its color-coded gradient map (b), as could be obtained by photometric stereo methods, and a binary mask (c) showing the location of the cliffs. Note that the gradient map is oblivious to the cliffs, and gives no clue as to which end of the ramp (if any) is at ground level.

Finally, even if the data is initially acquired over a regular X, Y grid of sampling points, the samples may become irregularly spaced when the data is subjected to optical rectification, filtering, or interpolation.

2 Previous solutions

There is a substantial bibliography on the gradient-to-height problem of computer vision, beginning with B. K. P. Horn's seminal papers [15, 10]. Three surveys have been published by Aggarwal [4], Ng *et al* [14], and Saracchini *et al* [17]. The published solution methods fall into a few major classes:

Path integration methods [6, 16, 3] compute the relative heigh of each pixel as a line integral along a single path from some reference pixel. These methods are very efficient ($\Theta(N)$ time and space, where N is the number of data pixels), but are extremely sensisitive to noise present in the gradient data and generally yield height maps with spurious cliffs. See figure 2(a).

Spectral methods, such as those of Frankot-Chellapa [7], Georghiades [8], and Wei [23] use the fast Fourier transform (FFT) to perform the integration by filtering the gradient data in the frequency domain. These methods are only slightly more expensive than path integration ($\Theta(N \log N)$ time and $\Theta(N)$ space) but fairly immune to random data noise. However, they cannot handle data with cliffs or missing samples, since the FFT only works with regularly spaced data and gives the same weight to every sample. When applied to scenes with cliffs, these methods return severely distorted height maps. See figure 2(b).

Poisson-like methods reduce the problem to an $N \times N$ sparse system of equations. The system can be obtained in many equivalent ways, such as an analogy with the Poisson second-order differential equation [4], an energy minimization formulation [4], a least squares solution to an overdetermined system [10], or a local averaging principle [18].

These methods can take into account a user-given binary *mask* that specifies the location of missing data and cliffs; or even a real-valued *weight map* that specifies the reliability of each gradient sample. With that information, they can modify the Poisson system so as to use only valid data, and avoid integrating around cliffs. As a result, they can accurately integrate scenes with cliffs, and can smoothly fill across isolated gaps in the data.

On the other hand, Poisson-like methods can be quite expensive. If the equations are linear, the system can be solved directly through Gaussian or Cholesky factorization, as described by Agrawal [4]. This method requires approximately $\Theta(N^{1.5})$ time and $\Theta(N^{1.15})$ space, with large constants factors. The high cost makes this approach impractical for megapixel gradient maps [17].

Alternatively, the linear system can be solved by the iterative Gauss-Seidel method using only $\Theta(N)$ space. However, the time needed to achieve a preset accuracy grows at least proportionally to N^2; so that even modest-size (100×100) gradient maps may require more than 10^5 iterations to produce a minimally usable result. See figure 2(c).

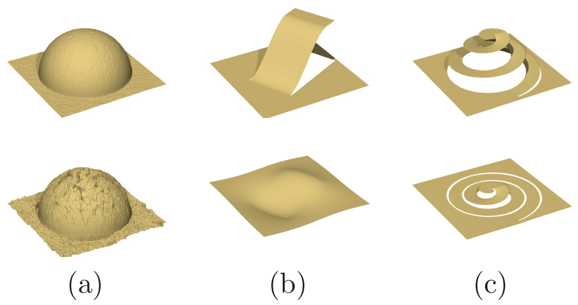

Figure 2: Ouputs of some existing integration algorithms (bottom) when fed with the simulated gradient maps of problematic scenes (top). (a) A path integration method applied to noisy data. (b) A Fourier integrator applied to a scene with cliffs. (c) A Poisson-like integrator with 50000 iterations of the Gauss-Seidel algorithm.

Kernel methods, introduced by Ng *et al* [14] assume a sparse gradient field, and reduce the problem to data fitting with a high-dimensional function approximation space. This approach can accomodate irregularly spaced gradient sampling points and is claimed to provides better "fill in" for missing data than Poisson methods. However it requires solving an even larger ($3N \times 3N$) linear equation system, and is therefore even more expensive in time and space.

2.1 Multi-scale acceleration

Let $\varepsilon^{(k)}$ be the residual error, namely the difference between the current guess and the true solution, after k Gauss-Seidel iterations. As observed by Terzopoulos in 1986 [21], the slow convergence of the Gauss-Seidel method is due to the Fourier components of $\varepsilon^{(k)}$ with low spatial frequency, which decrease very little at each iteration. The high-frequency components of the $\varepsilon^{(k)}$, on the other hand, are quickly eliminated after a few iterations.

He shows in [22, 21] that the Gauss-Seidel iterative algorithm can be accelerated by the use of *multi-scale techniques.* The idea is to recursively solve a coarse version of the original problem, with the gradient maps reduced to half size; and then use the resulting heigh map, expanded back to the original scale, as the initial guess for the Gauss-Seidel iterator. The initial guess will provide the correct low-fequency components of the solution, and the Gauss-Seidel loop quickly fixes the high frequency components. A fast Poisson-like integrator along these principles was developed by Saracchini *et al* [17]. Their algorithm was designed to cope with cliffs and missing data and tuned for accuracy and speed. It was shown to be as accurate and reliable as the direct-solving methods, but substantially faster. In particular, the the Gauss-Seidel algorithm converged in less than 200 iterations, instead of hundreds of thousands.

2.2 Weakly connected data

However, the multi-scale approach loses its effectivity when the slope maps contain narrow bands of data surrounded by cliffs or missing samples. As the resolution of the slope maps is reduced, the relative area affected by the missing samples expands, until the narrow bands disappear and/or the connectivity of the slope map is broken. See figure 3. At that point, the solution computed for the reduced problem is no longer a suitable starting guess, since its low-frequency components are usually quite wrong.

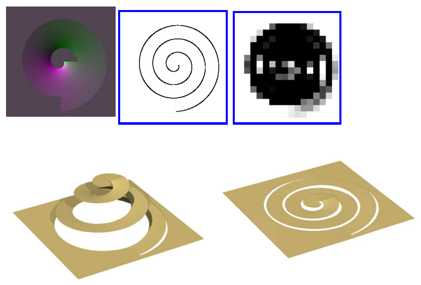
Figure 3: A pathological case for multi-scale integration. Clockwise from lower left: a scene with cliffs, its simulated photometric gradient map (256×256), two weight masks showing the position of cliffs at the original scale and at the reduced 16×16 scale, and the output of a multiscale integrator [17] with 200 iterations at the original scale.

The algorithm we describe here was devised to get around this limitation.

3 Integration on an irregular mesh

3.1 The weight-delta mesh model

Most published integration algorithms assume that that the data is presented as a regular (if incomplete) grid of samples. We depart from that tradition by using instead a *weight-delta mesh* representation for the gradient data. This is an abstract directed planar graph G with vertices (nodes) $\mathcal{V}\,G$ and edges (arcs) $\mathcal{E}\,G$, where each vertex v is associated to an unknown height sample value $z[v]$, and each directed edge e is associated to two numeric parameters: the *edge delta* $d[e]$, and the *edge weight* $w[e]$.

The edge delta $d[e]$ is an estimate for the difference $z[v] - z[u]$ between the height values at its destination vertex $v = \textsc{dst}(e)$ and its origin vertex $u = \textsc{org}(e)$. This estimate is presumably derived from measured surface gradients between the corresponding height sampling points (see section 4). The edge weight $w[e]$ is a positive number that expresses the reliability of that estimate. More precisely, we assume that the edge delta $d[e]$ includes some Gaussian measurement error, whose expected value is zero and whose variance is proportional to $1/w[e]$.

By definition, weight-delta meshes have no loop edges. A mesh is *simple* if it is free from parallel edges (two or more edges with the same origin and same destination). In a simple mesh, we can identify each edge e with the ordered pair (u, v) of its origin and destination vertices. In that case we may denote $d[e]$ also by $d[u, v]$, and $w[e]$ by $w[u, v]$. Also by definition, for every directed edge e in a weight-delta mesh, the oppositely directed edge $\text{SYM}(e)$ is also present in the mesh, with $d[\text{SYM}(e)] = -d[e]$ and $w[\text{SYM}(e)] = w[e]$. Therefore, when drawing the mesh it suffices to draw only one directed edge out of each pair $e, \text{SYM}(e)$.

A weight-delta mesh G can be interpreted as a system of linear *edge equations*

$$z[\text{DST}(e)] - z[\text{ORG}(e)] = d[e] \tag{1}$$

for every directed edge e. The problem is then to compute the most likely estimate for the height $z[v]$ of each vertex v, given the mesh and the parameters $d[e], w[e]$ for every graph edge e.

It is evident that each connected component of G can be treated as a separate instance of this problem. Therefore we will henceforth assume that G is a connected graph. Moreover, since equations (1) only depend on height differences, the solution for a connected graph has at least one degree of freedom, an additive constant.

A solution z to the problem is said to be *strain-free* if it satisfies all the edge equations (1) exacly. This is the case if, and only if, the sum of the (signed) edge deltas along any directed cycle of G is zero. Ths property is the mesh equivalent of the *integrability* or *curl-free* condition commonly stated for grid-based data. It is well known that this condition needs to be tested only at a *cycle basis* of G, such as the set of simple circuits that are created by adding to a spanning tree T of G each of the edges in $\mathcal{E}\, G \setminus \mathcal{E}\, T$, in turn. In particular, if the mesh is a single path (or, more generally, a tree), it always has a strain-free solution.

A strain-free solution $z[v]$, if it exists, can be computed sequentially by choosing a spanning tree T for G, assigning an arbitrary height to any vertex v_0, and then unsing equation (1) to compute the heights of all other vertices, in order of increasing graph distance from v_0 along T. Note that the edge weights are irrelevant in this case. This algorithm is a recasting of the path-integration methods for mesh data instead of gradient/weight maps.

If G has cycles , on the other hand, the equation system (1) is overdetermined. In that case, it is often impossible to satisfy all equations at the same time. Given the assumption of independent Gaussian measurement errors in the edge steps, Bayesian analysis says that the most likely set of heights z is the least squares solution to the system (1); namely, the heights z that minimizes the quadratic *mismatch function*

$$Q(z) = \sum_{e \in \mathcal{E}\, G} w[e](z[\text{DST}(e)] - z[\text{ORG}(e)] - d[e])^2 \tag{2}$$

The following mechanical analogy may help understand the problem. Each vertex v is modeled by a weightless horizontal plate that is free to slide vertically but cannot tilt or move horizontally; the height $z[v]$ is the vertical position of that plate. Each edge $e = (u, v)$

is modeled by an ideal vertical spring with stiffness coefficient $w[e]$, connected to plates u and v in such a way that it exerts on each of these two plates a vertical force with magnitude $w[e](z[v]-z[u]-d[e])$, that attempts to make the distance $z[v]-z[u]$ closer to $d[e]$. The desired height function z is the situation of mechanical equilibrium, where the net force on each plate is zero.

As we know from mechanics, the total potential energy of the springs will be $\frac{1}{2}Q(z)$, and the system will be at rest (with no net force acting on each vertex) when Q is minimum. Seuppose the mesh G is simple, and let $G[u]$ denote the neighbors of a vertex u in G. The function Q is minimized when for each vertex u,it is in equilibrium. That is if and only if we have Then a vertex u will be in equilibrium if and only if we have

$$\sum_{v\in G[u]} w[u,v](z[v]-z[u]-d[u,v]) = 0 \tag{3}$$

We can rewrite equation (3) as

$$z[u] = \frac{1}{w_{\text{tot}}} \sum_{v\in G[u]} w[u,v](z[v]-d[u,v]) \tag{4}$$

where $w_{\text{tot}}[u] = \sum_{v\in G[u]} w[u,v]$. In other words, $z[u]$ is the weighted average of $z[v]-d[u,v]$, over its neighbors, each neighbor v taken with weight $w[u,v]$. Equation (3) can be written also as

$$z[u] - \sum_{v\in G[u]} \lambda[v] z[v] = \sum_{v\in G[u]} \frac{w[u,v]}{w_{\text{tot}}[u]} d[u,v] \tag{5}$$

where $\lambda[v]$ is $w[u,v]/w_{\text{tot}}$, the relative weight of v among the neighbors of u

The solution z is strain-free, in particular, if the potential energy $Q(z)$ is zero; that is, if it leaves every spring is at its rest length.

Another physical analog for same mathematical problem is an electric circuit where each vertex v is a conducting node, the variable $z[v]$ is the node's electric potential (in volts), and each edge (u,v) is a battery with voltage $d[u,v]$ and internal resistance $1/Gw[u,v]$ (in ohms) connected to nodes u and v. The current traversing edge (u,v) (in amperes) is therefore $-w[u,v](z[v]-z[u]-d[u,v])$. The $Q(z)$ functional is the extra power that is dissipated by the resistors when the electric potential of each node v is $z[v]$. Equation (3) is then Kirchoff's node law, that holds when the circuit is at electrical equilibrium and the net current into or out of every node is zero.

3.2 The system in matrix form

The numerical can be expressed in matrix form. Let

- $v_1, v_2, \ldots, v_n$, be the vertices of G, in arbitrary order;
- $e_1, e_2, \ldots, e_m$ be an arbitrary list of directed edges of G, such that they include exactly one directed version of each undirected edge;

- $\mathbf{A}$ be the $n \times m$ incidence matrix of the mesh, such that $\mathbf{A}_{ij}$ is $+1$ if vertex v_i is the destination of edge e_j, is -1 is v_i is the origin of e_j, and is 0 otherwise;
- $\mathbf{W}$ be an $m \times m$ diagonal matrix, such that $\mathbf{W}_{jj} = w[e_j]$;
- $\mathbf{d}$ be an m-element row vector, such that $\mathbf{d}_j = d[e_j]$.
- $\mathbf{z}$ be an n-element column vector, such that $\mathbf{z}_i = z[v_i]$.

The Q functional can be expressed as the matrix product

$$Q(\mathbf{z}) \;=\; (\mathbf{z}^{\mathrm{T}}\mathbf{A} - \mathbf{d})\mathbf{W}(\mathbf{z}^{\mathrm{T}}\mathbf{A} - \mathbf{d})^{\mathrm{T}} \tag{6}$$

where $^{\mathrm{T}}$ denotes transposition. The vector $\mathbf{z}$ that minimizes Q is found by differentiating this formula with respect to each $\mathbf{z}_i$ and equating the result to zero. In matrix form, theseequations are

$$\mathbf{M}\mathbf{z} = \mathbf{b} \tag{7}$$

where

$$\mathbf{M} = \mathbf{A}\mathbf{W}\mathbf{A}^{\mathrm{T}} \tag{8}$$

and

$$\mathbf{b} = 2\mathbf{A}\mathbf{W}\mathbf{d}^{\mathrm{T}} \tag{9}$$

4 Grid to mesh conversion

The weight-delta mesh properly generalizes the concept of weighted gradient map, in the sense that the latter can be converted to the former.

For example, suppose we are given two arrays $f[p], g[p]$, with n_x columns and n_y rows, containing samples of the derivatives $\partial Z/\partial X$ and $\partial Z/\partial Y$, respectively, taken at a regular grid of sample points p. Suppose also that we are given an array $r[p]$, of the same size, with non-negative weights that express the reliability of those gradient values. More precisely, suppose $1/r[p]$ is the variance of the (zero-mean, Gaussian) measurement errors that contaminate $f[p]$ and $g[p]$. In particular, $r[p] = 0$ means that the data samples $f[p]$ and $g[p]$ are completely unknown.

To represent this information, we use a planar mesh G with the geometry of a regular grid of square "pixels", whose centers are gradient sampling points. The vertices of the mesh are therefore the corners of those pixels, which form a grid of points with n_x+1 columns and $n_y + 1$ rows. The edges of the mesh correspond to the sides of those squares. To compute the delta for each edge e, from a vertex q' to an adajcent vertex q'', we estimate the average gradient vector $t = (\partial Z/\partial X, \partial Z/\partial Y)$ along the edge, and we set $d[e] = t \cdot (q'' - q')$.

We obtain the vector t by combining the four gradient samples $(f[p], g[p])$ that flank the edge e (two on each side) by an interpolation formula that also takes into account the respective weights $r[p]$, based in a linear extrapolation. Let $t_0, \ldots, t_3$ be those gradient samples, in sequence, and $r_0, \ldots, r_3$ be their weights. We set

$$\begin{array}{rcl} t_- & = & \frac{3(t_1 - t_0)}{2} \\ t_o & = & \frac{(t_1 + t_2)}{2} \\ t_+ & = & \frac{3(t_2 - t_3)}{2} \end{array} \tag{10}$$

and

$$\begin{array}{rcl} r_- & = & 4/(9/r_1 + 1/r_0) \\ r_o & = & 4/(1/r_1 + 1/r_2) \\ r_+ & = & 4/(9/r_2 + 1/r_3) \end{array} \tag{11}$$

and compute r and t to

$$\begin{array}{rcl} r & = & r_- + r_o + r_+ \\ t & = & \frac{(r_- t_-) + (r_o t_o) + (r_+ t_+)}{r} \end{array} \tag{12}$$

In the simplest case, r is the (unweighted) average of the two gradient samples adjacent to e. Each weight r_i is set to 0 if the corresponding sample point p falls outside the gradient map. The weight $w[e]$ is then set to the reciprocal of the variance of the measurement noise present in $t \cdot (q'' - q')$, that is r .

In any case, if the computed weight $w[e]$ is zero, the edge is removed from G. Any vertices that become isolated are also removed from G. We assume that result had better be a connected mesh. (If not, each connected component should be extracted and integrated separately.)

5 The algorithm

Our algorithm uses the abstract multi-scale approach, but with planar weight-delta meshes instead of gradient and weight arrays.

The core of the algorithm is a *mesh decimation* step that removes a certain fraction of the vertices of the input mesh G, producing a smaller mesh G'. The vertices of G' are a subset of those of G, and the edges of G' are defined so as to summarize the weight and delta information contained in the edges of G. The integration problem is then solved recursively for the mesh G', yielding a tentative height function z' for its vertices. These heights are interpolated to provide a starting guess z for the original mesh G. The heights z are then adjusted by Gauss-Seidel iteration.

The recursion stops when the graph G is reduced to a single vertex v, whose height $z[v]$ can be set to zero. In other words, we construct a pyramid $G^{(0)}, G^{(1)}, \ldots, G^{(m)}$, where $G^{(0)}$ is the input mesh G, $G^{(m)}$ is a single vertex v, and each mesh $G^{(k+1)}$ is obtained by decimation of the previous one $G^{(k)}$. Then we compute solutions $z^{(m)}, z^{(m-1)}, \ldots, z^{(0)}$, in that order; where $z^{(m)}[v]$ is zero, and each $z^{(k)}$ is obtained from $z^{(k+1)}$ by interpolation and Gauss-Seidel iteration. The height map $z^{(0)}$ is the result. Formally, the algorithm is the recursive procedure *Integrate* whose pseudocode is given in figure 4. It takes as inputs the weight-delta mesh G,the iteration limit q and the tolerance ε and outputs a height function z from $\mathcal{V}\, G$.

```
***Integrate***$(G, q, \varepsilon)$
  1. If $\#\mathcal{V}G = 1$ then
      2. Let $v$ be the only vertex in
         $\mathcal{V}G$; set $z[v] \leftarrow 0$;
  3. else
      4. $G' \leftarrow$ *Decimate*$(G)$;
      5. $\beta \leftarrow \#\mathcal{V}G'/\#\mathcal{V}G$;
      6. $z' \leftarrow$
         *Integrate*$(G', q/\sqrt{\beta}, \varepsilon\sqrt{\beta},)$;
      7. $z \leftarrow$ *Interpolate*$(z', G)$;
      8. $z \leftarrow$ *SolveSystem*$(z, G, q, \varepsilon)$;
  9. Return $z$.
```

Figure 4: The main procedure of the integrator.

5.1 Mesh decimation

The procedure *Decimate*, called in step 4, takes a simple mesh G, planar and connected, and outputs a smaller mesh G', which is also simple, planar, and connected. As a consequence, the connectivity and planarity of the original mesh is preserved at all levels of the multiscale pyramid. The algorithm is therefore immune to the problems caused by early disconnection, and works even in the presence of narrow corridors.

The *Decimate* procedure first partitions $\mathcal{V}G$ into a set R of vertices to be removed, and a set K of vertices to be kept. The set R is a maximal subset of $\mathcal{V}G$ whose elements are pairwise disconnected (that is, are an independent set) and have degree six or less. For this task, the procedure uses a *mark* field for each vertex, which may have three possible states: REMOVE, KEEP, and BLANK. Initially all vertices are marked BLANK. For each degree k, from 1 to 6, the procedure scans all the vertices that are still BLANK; when a vertex with degree k is found, it is marked REMOVE, and all its neighbors are marked KEEP. At the end, the set R consists of all vertices that are marked REMOVE, and K consists of those marked KEEP or are still BLANK.

Next, the vertices in the R set are removed from G. Whenever a vertex u is removed, the edges incident to u are removed, too. If u has degree 1, nothing else needs to be done. If u has degree 2 or more, new edges are added to G', connecting the neighbors of u. (Observe that all these neighbors are in K and therefore they will be vertices of G'.) The endpoints, weights and deltas of the new edges are chosen so that the solution $z'[v]$ for the mesh G' is as close as possible to the solution $z[v]$, on every vertex $v \in K$.

More precisely, let k be the degree of u in G; let $e_0, e_1, \ldots, e_{k-1}$ be the edges incident to u, oriented out from u, in counterclockwise order around u; and let $v_0, v_1, \ldots, v_{k-1}$ be the corresponding destination vertices. Let w_i be the weight of e_i, and d_i its delta. It can be shown that the solution z' for G' would exactly match the solution z for G if, for every pair i, j, we added an edge $e'_{i,j}$ from v_i to v_j with the following delta and weight:

$$d'_{ij} = d_j - d_i \qquad w'_{ij} = \frac{w_i w_j}{w_{\text{tot}}} \tag{13}$$

where w_{tot} is the sum of all weights w_i. We call this operation — removal of u, removal of all incident edges e_i, and the addition of all edges e'_{ij} — a *star-clique swap.* If the vertex has degree $k = 2$, the swap will add only one pair of opposite edges e'_{01} and e'_{10}. If the degree k is 3, there will be three new edge pairs: e_{01}, $e'_{1,2}$, $e'_{0,2}$, and their opposites. In both cases, the planarity of the mesh G is preserved.

However, when the degree k is 4 or more, adding all the $k(k-1)$ directed edges $e'_{i,j}$ would generally make G' non-planar, and would severely impact the algorithm's efficiency. Therefore, when $k \geq 4$ we use instead a *star-cycle swap*, which adds only the edges $e'_{i,i+1}$ that connects successive vertices v_i and v_{i+1}, for $i \in \{0, 1, \ldots, k-1\}$ into a cycle, and their opposites. (All indices are taken modulo k). The deltas $d'_{i,i+1}$ of these edges are given by formula (13), namely

$$d'_{i,i+1} = d_{i+1} - d_i \tag{14}$$

The weights $w'_{i,i+1}$, on the other hand, are given by different formulas for each degree k, listed in table 1.

Table 1: Formulas for the weight w'_{01} of the new edge $e'_{01} = (v_0, v_1)$ added by the star-cycle swap, for each degree k. The same formulas hold for any other edge $e'_{i,i+1}$ of the cycle, except that all indices are incremented by i modulo k.

k	w'_{01}
2	$w_0 w_1 / w_{\text{tot}}$
3	$(w_0 w_1 + 0.5(w_0 w_2 + w_1 w_3))/w_{\text{tot}}$
4	$(w_0 w_1 + 0.5(w_0 w_2 + w_1 w_3))/w_{\text{tot}}$
5	$(w_0 w_1 + 1.1690(w_2 w_4 + w_0 w_2 + w_1 w_4))/w_{\text{tot}}$
6	$(w_0 w_1 + 2w_5 w_2 + 1.5(w_5 w_1 + w_0 w_2))/w_{\text{tot}}$

Unlike the star-clique swap, the star-cycle swap does not ensure that the heights determined by G' are exactly equal to those implied by G. In fact, it is not possible to achieve this exact match by adding only a proper subset of the clique edges e'_{ij}, as long as the weights and deltas of these edges are computed by purely local formulas (that depend only on the edges e_i). To achieve this goal, one would have to analyze the entire mesh G and essentially solve the integration problem.

However, experiments indicate that by adding only the $2k$ cycle edges, with the weights of table 1, the solution z' for the mesh G' retains the "low-frequency" components of the solution z of G — in the sense that the error is highly localized, and can be removed by only a few Gauss-Seidel iterations.

The star-cycle swap may give rise to parallel edges. These may be new edges added by distinct star-cycle swaps, or edges of G which had both endpoints in K and therefore were not removed. Therefore, after performing all the star-cycle swaps, the *Decimate* procedure collapses every set of parallel edges into a single equivalent edge. Namely, if edges e' and e''

have the same origin and destination, they are replaced by a single edge e with the same endpoints, with attributes

$$w[e] = w[e'] + w[e''] \qquad d[e] = \frac{w[e']d[e'] + w[e']d[e']}{w[e'] + w[e'']} \tag{15}$$

It is easy to see that this simplification pass does not change the solution z defined by equations (4).

5.2 Interpolation

Once a solution z' has been obtained for the reduced mesh G' (step 6), it is turned into a starting guess z for the full mesh G, by the procedure *Interpolate* (step 7). First, for every vertex v in the set K (which is shared by the two meshes), we set $z[v] \leftarrow z'[v]$. Then, for every vertex u in R (which exists only in G), we compute $z[u]$ by the vertex equation (4). Note that every neighbor $v \in G[u]$ belongs to K, and therefore its height $z[v]$ is defined at this point.

5.3 Iterative adjustment

The initial guess z is then used as the starting guess for the Gauss-Seidel procedure *SolveSystem* (step 8). Each iteration of the latter scans every vertex $u \in \mathcal{V}G$ and uses equation (4) to recompute its eight $z[u]$ from the current heights $z[v]$ of its neighbors. The procedure terminates after a specified maximum number q of iterations, of after the maximum absolute change in any height $z[u]$ is less than a specified tolerance ε, whichever happens first.

Note that each level of the recursion, the number of iterations q is increased by a factor $1/\sqrt{\beta}$, and the tolerance is reduced by $\sqrt{\beta}$ (step 6); where β is the mesh size reduction factor achieved by *Decimate* (step 4).

6 Analysis of the algorithm

6.1 Correctness

The star-cycle transformation and the collapsing of parallel edges preserve both planarity and connectivity, so the recursive calls to *Integrate* satisfy its preconditions. Moreover the final application of the Gauss-Seidel algorithm, at scale 0, will eventually converge to the unique solution $z = z^{(0)}$ of the vertex equations (4), irrespective of the starting guess obtained from the decimated mesh $G^{(1)}$. The experimental tests (section 7) show that the convergence is achieved after only a few iterations, even in instances that cause other multiscale methods to fail.

6.2 Time and space costs

The first critical question for the analysis of algorithm 4 is the depth of the recursion.

Let G be any mesh, and G' the mesh obtained from G by the decimation algorithm. The following statements hold:

1. If G is planar, then G' is planar.
2. If G is connected, then G' is connected.
3. If G is connected, simple, and planar, then $\#\mathcal{E}\, G \leq 3\#\mathcal{V}\, G$.
4. If G is connected, simple, and planar, then G has at least $\#\mathcal{V}\, G/7$ vertices with degree 6 or less.
5. If G is connected, simple, and planar, then $\#\mathcal{V}\, G' \leq (41/42)\#\mathcal{V}\, G$ vertices.

Properties 1 and 2 are obvious because Property 3 is a well-known property of simple-planar graphs, that follows from Euler's equation [5]. Property 4 follows from 3 and from the fact that the sum of all vertex degrees in a graph is twice the number of edges. Finally, property 5 follows from 4 and the observation that whenever a vertex is marked REMOVE at most six other vertices are marked KEEP.

Let N, N_k, M, M_k be the number of vertices and undirected edges of G and $G^{(k)}$, respectively. It can be proved that the *Decimate* procedure will always remove a minimum fraction $\beta < 1$ of the vertices. Therefore the maximum scale m is at most $\log_{1/\beta} N = O(\log N)$. Moreover, the total vertex count in all meshes is bounded by

$$N\sum_{k=0}^{m}\beta^k \;<\; N\sum_{k=0}^{\infty}\beta^k \;=\; \frac{N}{1-\beta} \;=\; O(N) \tag{16}$$

This result allows us to estimate the amount of memory required by the algorithm, which is dominated by the representation of the mesh $G^{(k)}$. A simple representation that is sufficient for our purposes consists of an *edge table* with $2M_k$ records, each containing the destination, weight and delta of a directed edge of $G^{(k)}$, sorted by the origin vertex; and an *edge index table* with N_k entries, that, for each vertex v, gives the index of the first edge in the edge table with origin v. The total storage space is then uses only $N_k + 2 \times 3M_k \leq 19N_k$ words for the mesh $G^{(k)}$, and $(19/(1-\beta))N$ words for all meshes in the pyramid.

As for the time cost, the decimation algorithm runs in time $O(N+M) = O(N)$ for a planar graph, and therefore the whole pyramid is built in $O(N/(1-\beta)) = O(N)$ time. The time required for one Gauss-Seidel iteration at level is $\Theta(N_k + 2M_k) = \Theta(N_k)$. Considering the recursive call on step 6, the maximum number of iterations at that level is $q_k = q/(\sqrt{\beta}^k)$. The maximum time spent at level k is then proportional to $N_k q_k = (N\beta^k)(q/(\sqrt{\beta})^k) = N(\sqrt{\beta})^k$. Therefore, the total work at all levels is $O(N/(1-\sqrt{beta}))$

It is known that, for planar simple graphs, $M \leq 6N$ and $M_k \leq 6N_k$; and that any such graph has at least $N/7$ vertices with degree 6 or less. From these facts it follows that $\beta \leq 41/42 \approx 0.976$. However, experiments with realistic meshes (see section 7) give $\beta \approx 0.6$ in practice (geometric average of about 1000 instances). Therefore the total number of vertices in all levels of the pyramid is usually $2.5N$.

6.3 Convergence speed

Even though the cost of the algorithm is linear in N, a bad initial guess may lead to an extremely slow convergence (hundreds of thousands iterations in practical applications.)

The algorithm's efficiency depends critically on the quality of the starting guess. This point is best evaluated experimentally. As shown in section refs.tests, the Gauss-Seidel iteration generally converges to near-maximum accuracy in a few tens of iterations at most; not only at scale 0, but al all higher scales too. section 7.)

When the; specifically, the depth of the recursion (number of levels in the mesh pyramid) and the total amount of work performed on all recursive calls. gradient maps are reduced, the higher-frequency components of the data are lost, while the remaining lower-frequency components have their wavelengths reduced by one half. Therefore, the recursively computed solution $z^{(k+1)}$ to the reduced problem, after being expanded to the original scale, will be mostly correct in the lower frequencies; only the small detail (at the scale of one or two pixels) will be missing. These details will be fixed by the Gauss-Seidel solver after a small number of iterations, largely independent of N. So, the recursive process is fast because each Fourier component of the height map gets computed at the scale where its wavelength is only a few pixels. Therefore, the time spent at scale k will be proportional $N/4^k$; and the total time for all scales will be $(1 + 1/4 + 1/4^2 + \cdots 1/4^m)O(N)$, this is less than $\frac{4}{3}O(N)$ and therefore $O(N)$.

7 Tests

In this section we compare the cost and accuracy of our graph-based multiscale integrator (MG) with those of other published methods. We considered only Poisson-like algorithms since they are the ones that can cope with errors and discontinuities in the gradient data. Namely, we used Agrawal's the M-Estimators (ME) and Affine Transforms (AT) algorithms [4], with direct system solving; and the the multi-scale integrator (MS) of Saracchini *et al* [17]. For ME and AT we used Agrawal's Matlab implementations [2] under MS Windows, adapted to use our input and output file formats and a user-given (rather than internally computed) weight map. For MS we used Saracchini's implementation in C. Our algorithm MG was also implemented in C; both were compiled and tested on a GNU Linux platform. The maximum number of Gauss-Seidel iterations q was set to 200 for MS (as proposed by the authors) and to 20 for our method.

7.1 Datasets

We used four datasets, provided by Saracchini *et al.* See figure 5. Three of them (`spdome`, `cbabel`, and `cpiece`) are defined by mathematical functions, and one (`dtface`) is a terrain model obtained by a structured-light 3D scanner [1]. The `spdome` consists in a half-hemisphere connected to a plane, it does not shows any cliffs or regions with missing data,as far has non-zero curl in the borders of sphere. The `hwave` is a medium-frequency sine wave. The `cbabel` is a spiral ramp with very sharp cliffs in its edges, its slope map is ambiguous and it cannot have its surface determined without a weight map. The `dtface` is a gradient map of a humam bust terrain model obtained by a 3D scanner [1], it has large regions with missing and unreliable data, and a rather complex object. Finnaly, the `cpiece` dataset is a cylinder connected by a ramp into a pit, that is connected to a outer plane by another ramp. Those ramps are weakly connected regions that may be troublesome to multi-scale based methods. See figure 5.

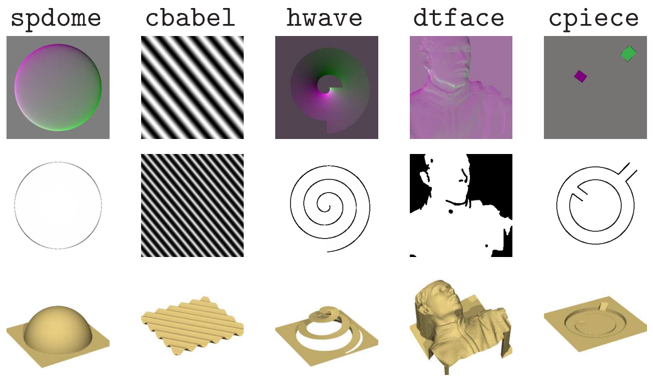

Figure 5: Datasets used in the tests, showing the gradient maps (top), the weight masks (middle), and the correct height map (bottom).

7.2 Accuracy and robustness

To test the robustness and accuracy of our method, we compared the outputs with each one of the algorithms with all the datasets available. Each dataset tries to cover different issues of surfaces, allowing the analysis of the integrators in each situation.

Gradient maps obtained from real objects often have a significant amount of noise which can interfere with the integration. In order to analyse the behaviour of the integration algorithms in its presence, we also tested gradient maps perturbed by a simulated noise. This noise is introduced by adding a random value under Gaussian distribution with zero mean and standard deviation $\sigma = 0.3$. This perturbation intends to emulate noise that occurs frequently in gradient maps obtained by physical means without significantly disrupting the gradient data.

AT and ME algorithms performs respectively a single-step and iterative method adjusting the weight maps to identify this noisy and outlier data on the gradient maps. Originally they are relied solely in the gradient map information and therefore were unable to detect discontinuities of the `cbramp` and `dtface` datasets, so we adapted those algorithms to accept our input weight maps assigning zero to the unreliable regions, otherwise, the weight assigned is the computed by its weight adustment step.

Each dataset was used in two versions: “clean” and “noisy”, the latter having its gradient map contaminated by Gaussian noise with zero mean and deviation 0.3. For each combination of dataset, noise, and algorithm, we computed the RMS error η between the two integrated height fields,and the relative RMS error η/R, where R is the RMS value of the two height fields. Both depth maps are first shifted to have zero mean, and the error is averaged only over the parts of the domain where the weight field w is nonzero.

	`spdome`		`hwave`		`cbabel`		`dtface`		`cpiece`	
Meth.	e	e/R	e	e/R	e	e/R	e	e/R	e	e/R
AT	1.64	4.7%	0.38	6.9%	0.02	0.1%	0.64	2.5%	0.21	1.6%
ME	0.56	1.6%	0.38	6.9%	0.02	0.1%	0.55	2.2%	0.21	1.6%
MS	0.16	0.4%	0.12	2.2%	25.90	138.2%	8.97	40.9%	10.91	119.0%
MG	0.02	0.1%	0.01	0.2%	0.02	0.1%	0.40	1.6%	0.24	1.9%
	`spdome-noise`		`hwave-noise`		`cbabel-noise`		`dtface-noise`		`cpiece-noise`	
Meth.	e	e/R	e	e/R	e	e/R	e	e/R	e	e/R
AT	3.32	9.8%	0.74	14.2%	0.80	3.0%	1.22	4.9%	0.52	4.1%
ME	0.63	1.8%	0.52	9.5%	0.86	3.3%	0.71	2.8%	0.55	4.3%
MS	0.36	1.0%	0.37	6.7%	25.90	138.3%	8.96	40.8%	10.87	118.3%
MG	0.34	1.0%	0.34	6.1%	0.80	3.1%	0.59	2.3%	0.52	4.1%

Table 2: Relative RMS errors of each method from the ground-truth reference solution for gradient maps without noise

As table 7.2 shows, the accuracy of our MG method was equivalent or better than that of the other three. Note that Saracchini's MS integrator failed on the `cbabel` and `cpiece` datasets, due to loss of connectivity after the first few levels of the pyramid.

7.3 Cost

To evaluate the efficiency of our method, we measured the computing time and memory needed for the integration of two square gradient fields, `spdome` and `dtface`, sampled with various grid sizes from 64×64 to 512×512. We choose two datasets,a slope map that takes maximum space to represent(`spdome`) in the system matrices and one with data that has to be discarded (`dtface`). Since we arent considering the weight determination step in this tests, for the Agrawal's methods we compared only its innermost loop of his M-Estimator, Energy Minimisation, and α-Surface methods. For all the methods involved we give a already computed weight map. Due the diversity of implementations (Matlab and C) and running platforms (Windows and Linux, their execution times aren't directly comparable. We timed only the construction and solution of the linear system, excluding other secondary tasks like data input and output. For the memory tests, we counted only the space used by the linear system itself and by the working storage used in its solution. In the case of multiscale algorithms, we counted also the time and memory used to prepare the pyramid of gradient and weight maps.

All the tests were run on a Dell XPS 1340 laptop with a 2.4 Ghz Intel P8600 Core Duo processor and 4 GB of available memory. The Matlab coded integrators were tested in Matlab 2008b under Windows Vista. Our algorithm was coded in C, compiled with GNU GCC and tested in Mandriva Linux 2010. To avoid memory address limitations of 32 bits platforms we used 64 bits versions of all software involved (operating systems, Matlab, compilers,etc...).
Note that our MG method is substantially faster than AT and ME, and only twice as slow as

Execution time (secs)								
	`spdome`				`dtface`			
Execution time (secs)	`spdome`				`dtface`			
N	AT	PC	MS	MG	AT	PC	MS	MG
$64 \times 64 = 4096$	0.43	0.89	0.06	0.10	0.24	0.17	0.06	0.07
$90 \times 90 = 8100$	1.12	0.89	0.11	0.19	0.58	0.43	0.12	0.16
$128 \times 128 = 16384$	2.94	2.01	0.22	0.42	1.32	1.23	0.25	0.33
$180 \times 180 = 32400$	9.17	7.21	0.43	0.90	3.22	3.16	0.48	0.63
$256 \times 256 = 65536$	25.58	18.93	0.96	2.00	11.71	12.66	0.96	1.37
$360 \times 360 = 129600$	65.92	69.07	1.71	3.75	28.60	29.81	1.89	2.63
$512 \times 512 = 262144$	202.91	131.12	3.54	7.63	94.23	82.30	3.86	5.41
Memory usage(MB)								
N	AT	PC	MS	MG	AT	PC	MS	MG
$64 \times 64 = 4096$	1.7	1.3	0.2	0.8	0.7	0.6	0.2	0.8
$90 \times 90 = 8100$	3.7	3.0	0.5	1.7	1.7	1.4	0.5	1.7
$128 \times 128 = 16384$	8.7	7.1	1.0	3.3	3.5	3.2	1.0	3.3
$180 \times 180 = 32400$	18.7	15.6	1.9	6.6	7.5	7.6	1.9	6.6
$256 \times 256 = 65536$	42.8	35.5	3.9	13.4	17.2	17.3	3.9	13.4
$360 \times 360 = 129600$	92.5	77.2	7.8	26.4	38.3	37.8	7.8	26.4
$512 \times 512 = 262144$	213.1	174.1	15.7	53.5	88.0	87.7	15.7	53.5

Table 3: Running times and work memory usage of the analyzed methods

MS (but succeeds in all cases). Moreover MG scales linearly with N, like MS, whereas ME and AT are superlinear. The table reftab.perf shows that the multiscale methods scales memory and running time in $O(N)$, meanwhile the direct solving ones scales it respectively as $O(N^{1.15})$ and $O(N^{1.5})$. The direct solving methods need to store the matrix A and also Gauss's triangular factor U (or Cholesky's R). For these methods, we counted the nonzero entries N_A in A and N_U in U, and estimated the memory usage conservatively as $12N_A + 16N_U$ bytes Our method requires more memory usage and execution time than the pure multiscale method because the cost of storing the graph and *decimating* it is more expensive than merely shrinking the slope and weight maps,even then it stills scales linearly.

8 Conclusions

We have described in this paper an algorithm that allows the integration of slope maps within linear time without loss of quality even in presence of issues that affects others methods. It has also potential to be used as extension for iterative methods such as described in [4], were the the computed heights are used to determine the weights of the next iteration, allowing the detection of outliers and noisy data. It can also be easily programmed for FPGAs and offer robust surface integration for real-time applications. In a near future we intend to determine more appropriate estimates for the equationsin table 1, reducing the need of even more iterations for the multi-scale step and improve the representation of the graph structure, reducing the memory usage and running time.

9 Acknowledgements

This work was partially funded by the UWE, by the Brazilian agencies FAPESP (grants 2007/59509-9 and 2007/52015-0), CNPq (grant 301016/92-5 and 550905/07-3), and CAPES (grant 342109-0) and by the UK Research Concil EPSRC (grant EP/E028659/1). We also thanks Antonio Robles, Jean-Denis Durou, Jean-François Aujol and Amit Agrawal, who provided kindly the source code and executable files of their integrators, needed for our tests.

References

[1] 3dMD. 3dMDFace system. Electronic document at `http://www.3dmd.com/3dmdface.html`, edit date 2010/10/26, last acessed on 2010/10/26, October 2010.

[2] A. Agrawal. Matlab/Octave code for robust surface reconstruction from 2D gradient fields. Available from `http://www.umiacs.umd.edu/~aagrawal/software.html`, 2006. See [4].

[3] A. Agrawal, R. Chellappa, and R. Raskar. An algebraic approach to surface reconstruction from gradient fields. In *Proc. 7th ICCV*, pages 174–181, 2005.

[4] A. Agrawal, R. Raskar, and R. Chellappa. What is the range of surface reconstructions from a gradient field? In *Proc. 9th ECCV*, pages 578–591, May 2006.

[5] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. MacMillan, London, 1976.

[6] R. Fraile and E. R. Hancock. Combinatorial surface integration. In *Proc. 18th ICPR'06*, volume 1, pages 59–62, 2006.

[7] R. T. Frankot and R. Chellappa. A method for enforcing integrability in shape from shading algorithms. *IEEE TPAMI*, 10(4):439–451, 1988.

[8] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE TPAMI*, 23:643–660, 2001.

[9] M. F. Hansen, G. A. Atkinson, L. N. Smith, and M. L. Smith. 3D face reconstructions from photometric stereo using near infrared and visible light. *CVIU*, 2010.

[10] B. K. P. Horn. Height and gradient from shading. *Intl. Journal of Computer Vision*, 5(1):37–75, 1990.

[11] B. K. P. Horn and Michael J. Brooks. *Shape from Shading*. MIT Press, Cambridge, Mass., 1989.

[12] B. K. P. Horn, R. J. Woodham, and W. M. Silver. Determining shape and reflectance using multiple images. Technical Report AI Memo 490, MIT, 1978.

[13] M. Kampel and R. Sablatnig. 3D puzzling of archeological fragments. In Danijel Skocaj, editor, *Proc. of 9th Computer Vision Winter Workshop*, pages 31–40, 2004.

[14] Heung-Sun Ng, Tai-Pang Wu, and Chi-Keung Tang. Surface-from-gradients without discrete integrability enforcement: A Gaussian kernel approach. *IEEE TPAMI*, 32(11):2085–2099, November 2010.

[15] B. K. P.Horn. Height and gradient from shading. Technical Report AI Memo 1105, Massachusetts Institute of Technology, 1989.

[16] A. Robles-Kelly and E. R. Hancock. Surface height recovery from surface normals using manifold embedding. In *Proc. Intl. Conf. on Image Processing (ICIP)*, October 2004.

[17] Rafael Saracchini, Jorge Stolfi, Helena Leitao, Gary Atkinson, and Melvyn Smith. Multi-scale depth from slope with weights. In *Proceedings of the British Machine Vision Conference*, pages 40.1–40.12. BMVA Press, 2010. doi:10.5244/C.24.40.

[18] G. D. J. Smith and A. G. Bors. Height estimation from vector fields of surface normals. In *Proc. IEEE Intl. Conf. on Digital Signal Processing (DSP)*, pages 1031–1034, 2002.

[19] M. L. Smith and L. N. Smith. *Polished Stone Surface Inspection using Machine Vision*, page 33. OSNET, 2004.

[20] J. Sun, M. L. Smith, A. R. Farooq, and L. N. Smith. Concealed object perception and recognition using a photometric stereo strategy. In *Proc. 11th Intl. Conf. on Advanced Concepts for Intelligent Vision Systems (ACIVS)*, volume 5807 of *Lecture Notes in Computer Science*, pages 445–455, October 2009.

[21] D. Terzopoulos. The computation of visible-surface representations. *IEEE TPAMI*, 10(4):417–438, 1988.

[22] Demetri Terzopoulos. Image analysis using multigrid relaxation methods. *IEEE TPAMI*, PAMI-8(2):129–139, March 1986.

[23] T. Wei and R. Klette. Height from gradient using surface curvature and area constraints. In *Proc. 3rd Indian Conf. on Computer Vision, Graphics and Image Processing*, 2002.

[24] R. J. Woodham. Photometric method for determining suface orientation from multiple images. *Optical Engineering*, 19(1):139–144, 1980.