Multi-scale Integration of Slope Data on an Irregular Mesh

Rafael F.V. Saracchini[1], Jorge Stolfi[1], Helena C.G. Leitão[2], Gary Atkinson[3], and Melvyn L. Smith[3]

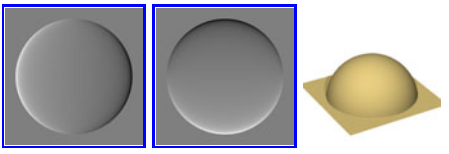
[1] State University of Campinas, Campinas, Brazil
{ra069320,stolfi}@ic.unicamp.br
[2] Fluminense Federal University, Niteroi, Brazil
hcgl@ic.uff.br
[3] University of West England, Bristol, United Kingdom
{Gary.Atkinson,Melvyn.Smith}@uwe.ac.uk

Abstract. We describe a fast and robust gradient integration method that computes 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 the scene's depth, e.g. along object silhouette edges; and for irregularly spaced sampling points. To accommodate these features of the problem, we use an original and flexible representation of slope data, the weight-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. Its novel key step is a mesh decimation procedure that preserves the connectivity of the initial mesh. 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 our algorithm remains accurate and efficient even for large sets of weakly-connected instances of the problem, 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 [9,8] and multiple-light photometric stereo [10,22]. 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). See figure 1.

Y.-S. Ho (Ed.): PSIVT 2011, Part I, LNCS 7087, pp. 109–120, 2011.



(a) (b) (c)

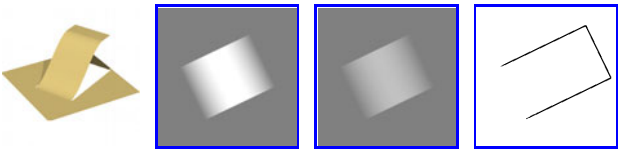
Fig. 1. Derivative maps $\partial Z/\partial X$ and $\partial Z/\partial Y$(b,c) of a hemisphere and the height map(c) obtained by integration

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 [18], pottery fragment reassembly [11], surveillance, face recognition [13], and many others.

In practical contexts, this problem faces at least four 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 neighborhood of a *gradient sampling point*.

Second, the gradient data is usually contaminated with *noise* arising from unavoidable measurement, quantization, and computation errors.

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. Most gradient acquisition methods, such as photometric stereo, will return meaningless values for any sample that straddles a cliff or that cannot be measured. For this reason, practical integration algorithms require an additional input, a real-valued *weight map* that specifies the reliability of each gradient sample. The weight map can be just a binary mask that is zero where there is invalid data or cliffs and 1 elsewhere. See figure 2.



(a) (b) (c) (d)

Fig. 2. A height map with cliff-like discontinuities (a), the derivative maps $\partial Z/\partial X$ and $\partial Z/\partial Y$(b,c), as could be obtained by photometric stereo methods, and a binary mask (d) 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 [9,8]. Three surveys have been published by Agrawal [3], Ng *et al.* [14], and Saracchini *et al.* [16]. The published solution methods fall into a few major classes:

Path integration methods [5,15,2] compute the relative height 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 3.

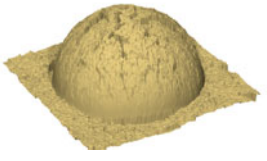


Fig. 3. Output of Fraile-Hancock's integrator [5] applied to the gradient data of figure 1 with noise added

Spectral methods, such as those of Frankot-Chellappa [6], Georghiades [7], and Wei [21] 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) and 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 4.

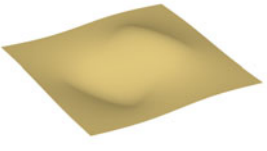


Fig. 4. Output of the Frankot-Chellappa integrator [6] applied to the gradient data of figure 2

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 a very large ($3N \times 3N$) linear equation system, and is therefore way expensive in time and space.

Direct Poisson-like methods reduce the problem to an $N \times N$ sparse system of equations which is solved directly through Gaussian or Cholesky factorization, as described by Agrawal [3]. The system can be obtained in many equivalent ways, such as by analogy to the Poisson second-order differential equation [3], through an energy minimization formulation [3], as the least squares solution to an overdetermined system [8], or by a local averaging principle [17]. These methods can take into account weight maps, modify the Poisson system so as to use only valid data and avoid integrating around cliffs. As result, they can handle problems that path-based and spectral methods cannot.

On the other hand, direct Poisson-like methods can be quite expensive. The solution of the system requires approximately $\Theta(N^{1.5})$ time and $\Theta(N^{1.15})$ space,

with large constants factors. The high memory cost makes this approach impractical for megapixel gradient maps [16].

Iterative Poisson-like methods build the same linear system as the direct variant, but solve it by the iterative Gauss-Seidel method [19]. With this approach the memory space needed is only $\Theta(N)$, but the time to achieve a preset accuracy grows at least proportionally to N^2; so that even modest (100×100) gradient maps may require more than 10^5 iterations to produce a minimally usable result.

Multi-scale Poisson-like methods, first described by Terzopoulos in 1986 [20,19], use *multi-scale techniques* to acelerate the Gauss-Seidel iterative algorithm. 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.

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 [19], 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 error, on the other hand, are quickly eliminated after a few iterations. Thus, the recursively computed 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 weighted Poisson-based integrator along these principles was developed in by Saracchini *et al.* [16].

2.1 The Problem of Weakly Connected Data

The multi-scale approach fails when the slope maps contain narrow bands of data surrounded by cliffs or missing samples. When the weight map is reduced, any pixel of the result that contains a zero weight pixel of the original must be set to zero too, since it may contain a cliff. It follows that the relative area affected by the missing samples expands at each successive reduction, until the narrow bands of data disappear and/or the connectivity of the gradient map is broken. See figure 5.

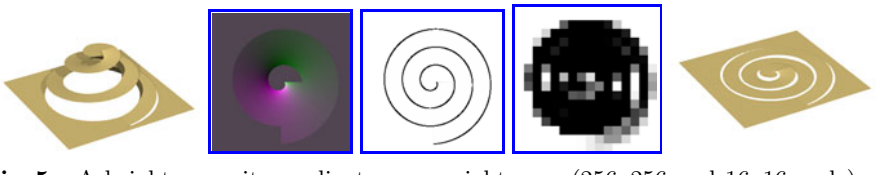


Fig. 5. A height map, its gradient map, weight map (256x256 and 16x16 scale) and the integrator's output [16] after 200 iterations

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. On such maps, the multiscale Gauss-Seidel solver becomes considerabily slower than the direct Gauss or Cholesky solver.

3 Integration on an Irregular Mesh

Our algorithm is a Poisson method with a novel multiscale iterative solver, that is effective even for weakly-connected instances like that of figure 5.

The Weight-Delta Mesh Model. We depart from tradition by using a graph representation for the gradient and weight data, instead a regular grid of samples. A *weight-delta mesh* (WDM) is an abstract directed planar graph G with vertices (nodes) $\mathcal{V}\,G$ and edges (arcs) $\mathcal{E}\,G$. Each vertex v represents a height sampling point and is associated to an unknown height value $z[v]$. Each directed edge e connects two close vertices and has 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 the edge's origin vertex $u = \text{ORG}(e)$ and its destination vertex $v = \text{DST}(e)$. This estimate is presumably derived from measured surface gradients between the corresponding height sampling points; the details of this computation depend on the application and are not relevant to this paper. 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 (provenient from camera noise, quantization,etc.), whose expected value is zero and whose variance is proportional to $1/w[e]$.

By definition, a weight-delta mesh has no loop edges. We say that a WDM is *simple* if it is free from parallel edges (two or more edges with same origin and destination). In a simple WDM, 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 WDM, 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 $e, \text{SYM}(e)$. See figure 6.

Edge Equations. A WDM can be interpreted as an equation system, with one *edge equation*

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

for every directed edge e. This equation is assumed to have "strength" $w[e]$. The problem is then to solve this system for the height $z[v]$ of each vertex v, given the mesh and the parameters $d[e], w[e]$ for every graph edge e.

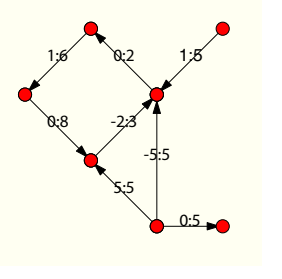


Fig. 6. A small WDM. The edge labels are $d[e]$:$w[e]$.

Since each connected component of the WDM implies a separate set of unknowns and equations, we will henceforth assume that the WDM is a connected graph. Note that the edge equations (1) only depend on height differences; therefore the solution for a connected mesh has at least one degree of freedom (an additive term corresponding to the integration constant of the continuous problem).

Vertex Equilibrium Equations. If G has cycles, the edge equation system (1) is overdetermined. In that case, measurement errors present in the deltas often make it impossible to satisfy all equations at the same time. Given the assumption of independent Gaussian measurement errors in the d values, Bayesian analysis says that the most likely set of heights z is the weighted least squares solution to the system (1). That solution turns out to satisfy the *vertex equilibrium equation*

$$z[u] - \sum_{v \in G[u]} \lambda[v] z[v] = - \sum_{v \in G[u]} \lambda[v] d[u, v] \tag{2}$$

for every vertex u, where $G[u]$is the set of vertices adjacent to u in the mesh and $\lambda[v]$ is $w[u, v] / \sum_s w[u, s]$, the *relative* weight of v among the neighbors of u.

4 The Algorithm

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 best summarize the weight and delta information contained in the edges of G. The algorithm then solves the problem recursively for the mesh G' yielding a tentative height function z' for its vertices. It then interpolates heights to provide a starting guess z for the original mesh G. Finally it adjusts the heights z by applying few Gauss-Seidel iterations to the equilibrium equations (2).

The recursion stops when G is reduced to a single vertex v, whose height $z[v]$ can be set to zero. In other words, we construct a pyramid $G^{(0)}, G^{(1)}, \ldots, G^{(m)}$ of meshes, 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 for its single vertex v, and each $z^{(k)}$ is obtained from $z^{(k+1)}$ by mesh interpolation and Gauss-Seidel iteration. The map $z^{(0)}$ is the result. See figure 7.

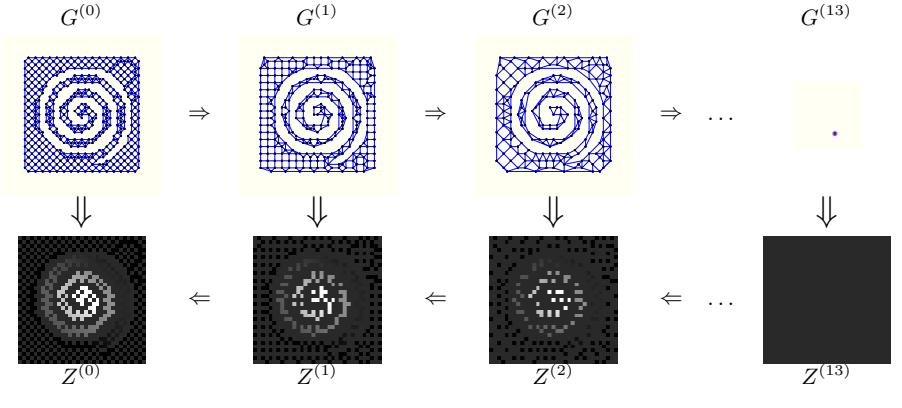


Fig. 7. The multiscale integration method

Formally, the algorithm is the recursive procedure *Integrate* whose pseudocode is given in figure 8. It takes as inputs the weight-delta mesh G, an iteration limit κ and a tolerance ε; and outputs a height function z from $\mathcal{V}\,G$ to $\mathbb{R}$.

Integrate(G, κ, ε)
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*$(G', \kappa/\sqrt{\beta}, \varepsilon\sqrt{\beta},)$;
   7. $z \leftarrow$ *Interpolate*(z', G', G);
   8. $z \leftarrow$ *SolveSystem*$(z, G, \kappa, \varepsilon)$;
9. Return z.

Fig. 8. The main procedure of the integrator

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.

First, the procedure 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 independent (that is, pairwise disconnected in G) and have degree six or less. The set R is found by a greedy algorithm [4].

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 delta $d'_{ij} = d_j - d_i$ and weight $w'_{ij} = w_i w_j / w_{\text{tot}}$, 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'_{12}, e'_{02}, 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; as well their opposites. (All indices are taken modulo k). The deltas $d'_{i,i+1}$ of these edges are those of the star-clique swap, namely $d'_{i,i+1} = d_{i+1} - d_i$. The weights $w'_{i,i+1}$, on the other hand, are given by different formulas for each degree k. For the new edge $e'_{01} = (v_0, v_1)$, we have

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	$(w_0 w_1 + 2 w_5 w_2 + 1.5(w_5 w_1 + w_0 w_2))/w_{\text{tot}}$

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.

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. However, 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.

An edge e'_{ij} introduced by the star-cycle swap may have the same endpoints as a preexisting edge. Therefore, after performing all the star-cycle swaps, the *Decimate* procedure collapses every set of parallel edges into a single equivalent edge in a way that preserves the final solution. Namely, if edges e' and e'' have the same origin and destination, with weights w', w'' and deltas d', d'', they are replaced by a single edge e with the same endpoints, with attributes

$$w[e] = w' + w'' \qquad d[e] = (w'd' + w''d'')/(w' + w'') \tag{3}$$

Interpolation. Once a solution z' has been obtained for the reduced mesh G' (step 6), it is expanded to a starting guess z for G, by the procedure *Interpolate* (step 7). First, for every vertex v in the shared set K, we set $z[v] \leftarrow z'[v]$. Then, for every vertex u in the deleted set R, we compute $z[u]$ by its vertex equilibrium equation (2). Note that every neighbor $v \in G[u]$ belongs to K, and therefore its height $z[v]$ is defined at this point.

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 the equilibrium equation (2) to recompute its height $z[u]$ from the current heights $z[v]$ of its neighbors. The procedure terminates after a specified maximum number κ of iterations, or after the maximum absolute change in any height $z[u]$ is less than the specified tolerance ε, whichever happens first. Note that the iteration limit κ is increased by a factor $1/\sqrt{\beta}$, and the tolerance ε is reduced by $\sqrt{\beta}$, at each level of the recursion (step 6); where β is the mesh size reduction factor achieved by *Decimate* (step 4).

5 Analysis of the Algorithm

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 that G' be simple, connected and planar. Therefore, the connectivity and planarity of the original mesh is preserved at all levels of the pyramid; even within narrow corridors the relevant gradient information is retained all the way to the top. Moreover, if κ is large enough, 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 (2), irrespective of the starting guess obtained from the decimated mesh $G^{(1)}$. The experimental tests (section 6) show that convergence is achieved after only a few iterations, even in instances that cause other multiscale methods to fail.

Space and Time Costs. Let $N = \#\,\mathcal{V}\,G, N_k = \#\,\mathcal{V}\,G^{(k)}$, $M = \#\,\mathcal{E}\,G, M_k = \#\,\mathcal{E}\,G^{(k)}$. It is known that, for planar simple graphs, $M \le 6N$ and $M_k \le 6N_k$; and that any such graph has at least $N/7$ vertices with degree 6 or less. From these facts it follows that the vertex reduction factor β of the *Decimate* procedure has a theoretical upper bound $\hat{\beta} \le 41/42 \approx 0.976$ [12]. In practice, the reduction factor β is usually 0.6.

The maximum scale m is therefore at most $\log_{1/\hat{\beta}} N = O(\log N)$. Moreover, the total vertex count in all meshes is at most $N/(1-\hat{\beta}) = O(N) \approx 2.5N$ in practice. The amount of memory required by the algorithm is dominated by the representation of the mesh $G^{(k)}$; a simple representation that is sufficient for our purposes uses only $N_k + 2 \times 3M_k \le 19N_k$ words for the mesh $G^{(k)}$, and $(19/(1-\hat{\beta}))N$ words for all meshes in the pyramid.

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

6 Tests

In this section we experimentally compare the cost and accuracy of our graph-based multiscale integrator (MG) with those of other published methods. We consider only weighted Poisson-based algorithms since they are the ones that can cope with errors and discontinuities in the gradient data within an acceptable execution time. Another methods such were not tested due being unable to cope with discontinuities [6], high sensibility towards gradient noise [5] or too high memory/time requirements to be comparable [14].

Specifically, we used the M-Estimators (ME) and Affine Transforms (AT) algorithms [3] of Agrawal *et al.* with direct system solving; and the the multi-scale iterative integrator (MS) of Saracchini *et al.* [16]. For ME and AT we used

the author's Matlab implementations [1] 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 the author'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 κ was set to 200 for MS (as proposed by the authors) and to 20 for our method.

Datasets. In our tests we used four datasets provided by Saracchini *et al.*, as shown in figure 9. Three of them (`spdome`, `cbabel`, and `cpiece`) are defined by mathematical functions, and one (`dtbust`) is a terrain model of a human torso obtained by a structured-light 3D scanner. In order to simulate the measurement noise usually present in real datasets, we added to each gradient sample an independent Gaussian random number with zero mean and deviation 0.3. Each gradient and its weight map were converted to a WDM whose vertices were the pixels of desired height map and whose arcs connected pixels that were vertically or horizontally adjacent in that map. The final vertex height z were then output in the regular grid format.

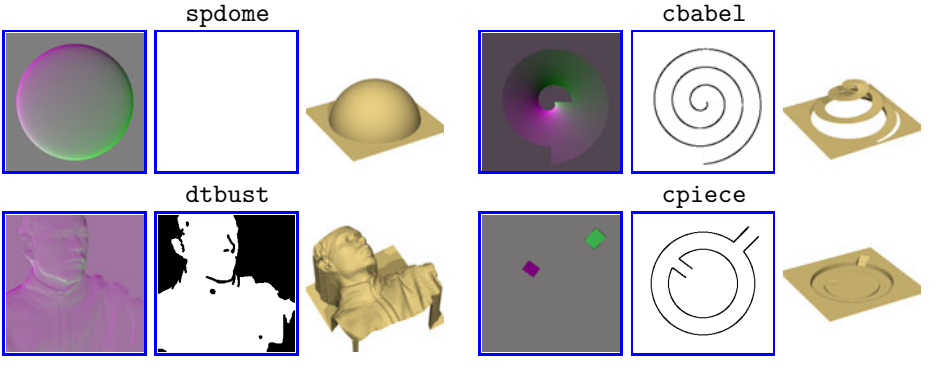


Fig. 9. Datasets used in the tests, showing the gradient maps (left), the weight masks (middle), and the correct height map (right)

Accuracy and Robustness. For each combination of dataset and algorithm, we computed the RMS value ρ of the correct and integrated height fields, and the RMS difference η between them. In these computations, the height fields were first shifted to have zero mean, and all averages are weighted by the input weight maps.

Table 1. Relative RMS errors of each method

Results - datasets with 30% of Gaussian noise								
	`spdome`		`cbabel`		`dtbust`		`cpiece`	
Meth.	η	η/ρ	η	η/ρ	η	η/ρ	η	η/ρ
AT	3.32	9.8%	0.80	3.0%	1.22	4.9%	0.52	4.1%
ME	0.63	1.8%	0.86	3.3%	0.71	2.8%	0.55	4.3%
MS	0.34	1.0%	23.02	121.0%	0.67	2.7%	5.74	52.5%
MG	0.34	1.0%	0.80	3.1%	0.59	2.3%	0.52	4.1%

As table 1 shows in these tests the accuracy of our MG method was equivalent or better than that of the other three. Note that MS integrator failed on the `cbabel` and `cpiece` datasets, due to loss of connectivity after the first few levels of the pyramid. On the `dtbust` dataset, MS gives the correct solution but only after 200 iterations at the base level.

Cost. To evaluate the efficiency of our method, we measured the computing time and memory needed for the integration of two gradient fields sampled with various grid sizes from 64×64 to 512×512. We used the two datasets which where correctly integrated by all methods (`spdome` and `dtbust`), without noise.

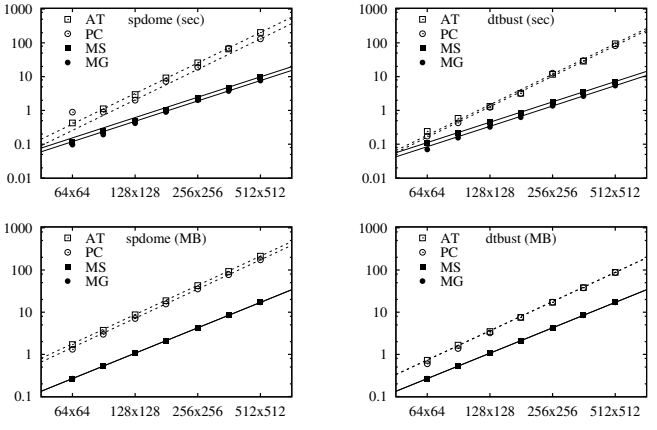


Fig. 10. Log-log plots of the running time (top) and memory usage (bottom) of PC, AT,MS and MG

For AT and ME, we measured only the system solving step; namely, we aborted the algorithm after a single iteration of its weight-computing step. For MS and MG, we included the cost of their decimation/interpolation steps as well as of the Gauss-Seidel solver. The direct solving methods AT and ME need to store the Poisson system's matrix A and also its Gaussian triangular factor U (or Cholesky's R). For those methods, we counted the nonzero entries N_A in the system's matrix A and N_U in its Gauss or Cholesky's factor U, and estimated the memory usage conservatively as $12N_A + 16N_U$ bytes For MS we used the memory estimate given by the authors [16]. For our method we used the estimate $19N_{tot}$ where N_{tot} was the actual number of vertices in all meshes.

The running times of MS and MG cannot be compared directly to those of AT and ME, since Matlab code is inherently slower than C code. However, figure 10 shows that memory and time costs of MS and MG scale linearly with N, where as those of AT and ME scale as $O(N^{1.15})$ and $O(N^{1.5})$, respectively.

7 Conclusions

Our algorithm allows robust integration of slope maps with cliffs and missing data. Unlike previous linear-cost algorithms, it can handle gradient maps with

narrow corridors. Also it can be used as the inner loop of iterative methods such as described in [3], were the the computed heights are used to determine the weights of the next iteration, allowing the detection of outliers and noisy data.

References

1. Agrawal, A.: Matlab/Octave code for [3] (2006), `http://www.umiacs.umd.edu/~aagrawal/software.html`
2. Agrawal, A., Chellappa, R., Raskar, R.: An algebraic approach to surface reconstruction from gradient fields. In: Proc. 7th ICCV, pp. 174–181 (2005)
3. Agrawal, A., Raskar, R., Chellappa, R.: What is the Range of Surface Reconstructions From a Gradient Field? In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3951, pp. 578–591. Springer, Heidelberg (2006)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. McGraw-Hill (1990)
5. Fraile, R., Hancock, E.R.: Combinatorial surface integration. In: Proc. 18th ICPR 2006, vol. 1, pp. 59–62 (2006)
6. Frankot, R.T., Chellappa, R.: A method for enforcing integrability in shape from shading algorithms. IEEE TPAMI 10(4), 439–451 (1988)
7. Georghiades, Belhumeur, Kriegman: Illumination cone models for face recognition under variable lighting and pose. IEEE TPAMI 23, 643–660 (2001)
8. Horn, B.K.P.: Height and gradient from shading. IJCV 5(1), 37–75 (1990)
9. Horn, B.K.P., Brooks, M.J.: Shape from Shading. MIT Press (1989)
10. Horn, B.K.P., Woodham, R.J., Silver, W.M.: Determining shape and reflectance using multiple images. Technical Report AI Memo 490. MIT (1978)
11. Kampel, M., Sablatnig, R.: 3D puzzling of archeological fragments. In: Skocaj, D. (ed.) Proc. of 9th Computer Vision Winter Workshop, pp. 31–40 (2004)
12. Kirkpatrick, D.G.: Optimal search in planar subdivisions. SIAM J. on Computing 12, 28–35 (1983)
13. Smith, L.N., Hansen, M.F., Atkinson, G.A., Smith, M.L.: 3D face reconstructions from photometric stereo using near infrared and visible light. Computer Vision and Image Understanding 114, 942–951 (2010)
14. Ng, Wu, Tang: Surface-from-gradients without discrete integrability enforcement: A Gaussian kernel approach. IEEE TPAMI 32 (November 2010)
15. Robles-Kelly, A., Hancock, E.R.: Surface height recovery from surface normals using manifold embedding. In: Proc. ICIP (October 2004)
16. Saracchini, Stolfi, Leitao, Atkinson, Smith: Multi-scale depth from slope with weights. In: Proceedings of the BMVC, pp. 40.1–40.12. BMVA Press (2010)
17. Smith, G.D.J., Bors, A.G.: Height estimation from vector fields of surface normals. In: Proc. IEEE DSP, pp. 1031–1034 (2002)
18. Smith, M.L., Smith, L.N.: Polished Stone Surface Inspection using Machine Vision, page 33. OSNET (2004)
19. Terzopoulos, D.: The computation of visible-surface representations. IEEE TPAMI 10(4), 417–438 (1988)
20. Terzopoulos, D.: Image analysis using multigrid relaxation methods. IEEE TPAMI PAMI 8(2), 129–139 (1986)
21. Wei, T., Klette, R.: Height from gradient using surface curvature and area constraints. In: Proc. 3rd ICVGIP (2002)
22. Woodham, R.J.: Photometric method for determining suface orientation from multiple images. Optical Engineering 19(1), 139–144 (1980)