CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Elsevier - Publisher Connector

Computer Vision and Image Understanding 116 (2012) 882–895

Contents lists available at SciVerse ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

A robust multi-scale integration method to obtain the depth from gradient maps ☆,☆☆

Rafael F.V. Saracchini [a,*], Jorge Stolfi [a], Helena C.G. Leitão [b], Gary A. Atkinson [c], Melvyn L. Smith [c]

[a] Institute of Computing, State University of Campinas, Brazil
[b] Institute of Computing, Federal Fluminense University, Brazil
[c] Machine Vision Laboratory, University of the West England, Bristol, United Kingdom

ARTICLE INFO

Article history:
Received 7 December 2011
Accepted 19 March 2012
Available online 27 March 2012

Keywords:
Computer vision
Multi-scale methods
Gradient map integration
Surface reconstruction

ABSTRACT

We describe a robust method for the recovery of the depth map (or height map) from a gradient map (or normal map) of a scene, 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, and also for sharp discontinuities in the scene's depth, e.g. along object silhouette edges. By using a multi-scale approach, our integration algorithm achieves linear time and memory costs. A key feature of our method is the allowance for a given weight map that flags unreliable or missing gradient samples. We also describe several integration methods from the literature that are commonly used for this task. Based on theoretical analysis and tests with various synthetic and measured gradient maps, we argue that our algorithm is as accurate as the best existing methods, handling incomplete data and discontinuities, and is more efficient in time and memory usage, especially for large gradient maps.

© 2012 Elsevier Inc. Open access under CC BY-NC-ND license.

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 [1,2] and multiple-light photometric stereo [3,4]. These methods usually determine a mean normal direction within each image pixel, from which one can obtain the surface gradient. Although this information alone does not determine the absolute heights, it can yield height differences between parts of the same surface. This relative height information is sufficient for many applications, such as industrial quality control [5], pottery fragment reassembly [6], surveillance and customs inspections [7], face recognition [8], and many others.

1.1. The mathematical problem

Abstractly, our goal is to determine an unknown function Z of two variables x and y defined on some region D of $\mathbb{R}^2$, given its gradient $\nabla Z = (\partial Z/\partial x, \partial Z/\partial y)$. That is, we wish to find Z such that

$$\frac{\partial Z}{\partial x}(x,y) = F(x,y) \quad \frac{\partial Z}{\partial y}(x,y) = G(x,y) \tag{1}$$

at every point (x, y) within the region D, where F and G are two given real functions defined on D. It is well known that this problem has a differentiable solution if and only if

$$\frac{\partial F}{\partial y}(x,y) - \frac{\partial G}{\partial x}(x,y) = 0 \tag{2}$$

for all (x, y) in D. The left-hand side of formula (2) is the curl (rotational) of the vector field (F, G), so this requirement is often called the *zero curl condition*.

If Eq. (2) holds, the solution Z can be expressed in many ways. For a rectangular domain with corner at $(0, 0)$, for example, it can be written as

$$Z(x,y) = C + \int_0^y G(0,v)\, dv + \int_0^x F(u,y)\, du \tag{3}$$

where C is an arbitrary constant. Note that the degree of freedom represented by C is an inherent feature of the original problem, not a limitation of the method.

1.2. Computational difficulties

In practical contexts, there are at least three difficulties with this approach. First, the gradient functions F and G are usually *discretized*, i.e. known only at certain *gradient sampling points* $p[u, v]$, which usually form a regular orthogonal grid. Each sample value $f[u, v]$ (resp. $g[u, v]$) is some weighted average of F (resp. G) over some neighborhood of the point $p[u, v]$. Note that photometric

☆ This work was partly supported by Grants from Brazil's CNPq (Grant 301016/92-5 and 550905/07-3), FAPESP (Grants 2007/59509-9 and 2007/52015-0), and CAPES (Grant 342109-0); and by the UK EPSRC (Grant EP/E028659/1).
☆☆ This paper has been recommended for acceptance by Edwin Hancock.
* Corresponding author.
E-mail address: saracchini@gmail.com (R.F.V. Saracchini).

1077-3142 © 2012 Elsevier Inc. Open access under CC BY-NC-ND license.
http://dx.doi.org/10.1016/j.cviu.2012.03.006

stereo methods will typically estimate the x-gradient and the y-gradient at the same points, e.g. at the center of each image pixel.

Second, the 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. In the case of photometric stereo, these regions include all pixels where the scene's surface is too dark, shadowed, specular, or outside the range of the illumination source.

Third, the height function Z is usually *discontinuous*: It almost always has step-like discontinuities, or *cliffs*, at the edges of solid objects. At any pixel of a real scene that straddles a cliff, photometric stereo and other gradient acquisition techniques usually fail to detect cliffs, and return a grossly incorrect gradient sample that gives no clue as to the height of the cliff. This happens because the computation of $f[u, v]$, and $g[u, v]$ from raw photometric stereo data is a highly non-linear process, especially when the surface normal is nearly perpendicular to the viewing direction. Therefore, the samples $f[u, v]$ or $g[u, v]$ will be proper local averages of F and G only when the scene has nearly constant slope within the pixel. See Fig. 1.

Finally, the scene may also have regions where the height and gradient functions are poorly defined, e.g. where the scene is highly porous, covered with hair-like structures, or transparent.

1.3. Reliability weight maps

As Fig. 1 shows, one cannot always deduce the position of cliffs and invalid gradient data from the gradient map alone. To work around this problem, practical integration algorithms (including ours) require the user to provide a *weight map*, that specifies the reliability of each gradient sample. (In this paper we consider a single weight map for both f and g, but our algorithm can be trivially adapted to use a separate mask for each map.)

The weight map may be a simple binary mask which is 0 at any pixel whose gradient data is unknown or unreliable (e.g. it is suspected to contain a cliff), and 1 elsewhere. See Fig. 1c. More generally, the weight should be inversely proportional to the estimated variance of the measurement noise affecting the corresponding gradient sample. If all data samples are equally reliable, one may set all weights to 1. Fractional values between 0 and 1 may be used to indicate varying degrees of data reliability.

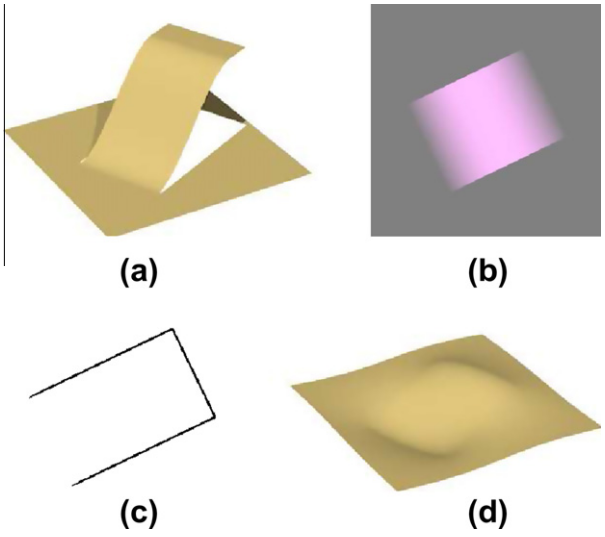

Fig. 1. A height map with cliff-like discontinuities (a), its color-coded gradient map (b), as could be obtained by photometric stereo methods, and a binary mask (c) showing the location of the cliffs. Note that the gradient map is oblivious to the cliffs, and gives no clue as to which end of the ramp (if any) is at ground level. Image (d) shows a distorted height map computed from (b) by the Frankot–Chellapa integrator [9], which is oblivious to cliffs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The weight map can be obtained in many ways, either from external information or by error detection algorithms applied to the gradient data [10–15]. The weights are often generated as a by-product of the measurement process that yielded the gradient data.

1.4. Our solution

Several integration methods that have been described in the literature (see Section 2) are unsuitable for gradient maps obtained by photometric stereo; either because they are too sensitive to noise, or because they cannot cope with cliffs and missing data samples, or because they are too costly for use with high-resolution maps.

In this paper we describe a new multi-scale iterative procedure for gradient map integration (see Section 5), that is as accurate and robust as the state-of-art methods, but substantially more efficient. Except for some extreme cases, its memory and time cost scale linearly with the number of data pixels, making it quite practical even for multi-megapixel maps.

2. Related work

2.1. Gradient map integration algorithms

In their survey [16], Kettle and Schlüns classified integration methods in two categories: *local path algorithms*, that compute the height values incrementally by line integrals along selected paths, and *global algorithms* that compute all height values simultaneously by an error minimization procedure. We will extend their classification into four broad groups: *path integration*, *Fourier filtering*, *local iteration*, and *direct system solving*.

Path-integration methods assign a height to one reference pixel p and then compute the height of every other pixel q by performing a numerical line integral of the gradient field along a path from p to q. This group includes the naive row-by-row integration, which is the discrete version of Eq. (3) [17] as well as other methods that choose the paths so as to avoid low-quality or missing data—e.g. by finding an optimum spanning tree and integrating along it, as done by Fraile and Hancock [18,10]. These methods are generally quite fast, since they require only $O(N)$ operations for an image with N pixels. However, as pointed out by Kettle and Schlüns [16], they are very sensitive to noise and discontinuities: if the heights of two adjacent pixels p',p'' are computed by distinct paths, the integration of the noise component of the gradient will result in a spurious height difference between them. See Fig. 2.

This problem can be alleviated, but not solved, by averaging the integral along many distinct paths between the two pixels [19]. Robles-Kelly and Hancock have proposed a variant where the gradient is integrated along a straight line between every pair of pixels, and these pairwise increments are then averaged to obtain

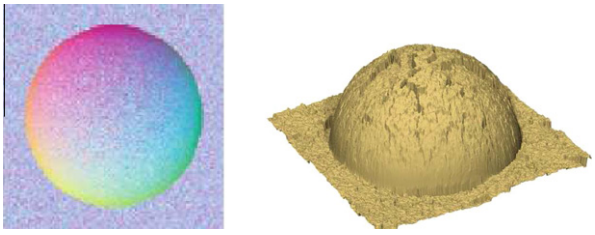

Fig. 2. Reconstruction of a noisy gradient map by the path-based integration method of Fraile and Hancock [18].

the height of each pixel [19]. While this approach gets rid of spurious steps due to noise, its cost is prohibitive (proportional to $N^{2.5}$ for an image with N pixels) and its results are still inferior to those of the other methods described below.

Fourier filtering methods rely on the fact that integrating a function corresponds to dividing each component of its Fourier transform by 2π times its frequency. This approach was pioneered by Frankot and Chellapa [9]. In the frequency domain one can easily filter out the curl component of the gradient data and apply other smoothing filters [20].

Through the use of fast Fourier transform algorithms (FFT or DCT) these methods obtain the height field for N pixels using only $O(N)$ memory and $O(N\log N)$ operations. However, this approach does not allow the use of a weight map, because the FFT always gives the same weight to all data samples. As a result, these methods will flatten out any invisible cliffs and deform the surface over a wide area surrounding them. See Fig. 1d.

Moreover, the Fourier integrator assumes that the Z function is periodic, meaning that the integral along each row of F and each column of G must be zero. This assumption can be met by properly mirroring and negating the gradient maps, which makes them four times as large. We note that popular implementations of the Frankot–Chellapa algorithm ignore this detail, and therefore fail if the Z values along each edge of D differ from those along the opposite edge.

Direct system solving methods reduce the gradient integration problem to a system of N equations whose unknowns are the N heights, and where each equation relates one height value and its neighbors to the given derivatives in that neighborhood. They then solve the system by a direct (non-iterative) method, such as Gaussian LU or Cholesky factorization. (If the equations are non-linear, they must be linearised and the process must be iterated over, as in the Newton-Raphson method. This outer loop usually has second-order convergence speed, so a few iterations are usually sufficient.) This approach is used, in particular, by several of Agrawal's "Poisson based" methods, including M-Estimators, Energy Minimization, α-Surface, and Affine Transform [11].

The local equations can be derived in several ways, e. g. as an energy minimization problem [11], or as the least-squares solution of an overdetermined system [2], or by averaging height estimates obtained from surrounding heights and gradients [21]. However, all these local criteria generally yield some discrete (and possibly non-linear) version of Poisson's equation $\nabla^2 Z(x, y) = H(x, y)$. Namely, each equation states the identity between two numerical estimates of the Laplacian of the height field: one ($\nabla^2 Z$) computed from the unknown heights, and one (H) computed from the given gradient data. The latter is, typically, a numerical estimate of the divergence $\nabla \cdot (F, G)$ of the given gradient field.

Since each equation refers to a small number of height values, the whole system uses only $O(N)$ storage. Unlike path-integration, the Poisson approach uses all the information that is present in the gradient map, automatically discarding its curl component, and does not generate spurious steps in the presence of noise. Indeed, the solution computed by these methods is theoretically equal to that of Fourier filtering if all the weights are equal. In this case, fast Fourier transforms can be used to efficiently solve the Poisson equation, as shown by Georghiades et al. [22].

The main advantage of the Poisson approach, as pointed out by Agrawal et al. in 2006 [11], is that each equation of the linear system can be individually adjusted so as to ignore bad data samples and suspected cliffs, as specified by the weight map. On the other hand, the direct solution of the Poisson system is generally slower than Fourier filtering and uses substantially more memory. Although the system's matrix is quite sparse, its Gaussian or Cholesky factors are substantially denser, so that memory usage scales more than linearly with the number of pixels N. According our tests, Agrawal's M-Estimator method, for example, needs about $5N$ nonzero elements in the system's matrix, but about $5N^{1.15}$ in its Cholesky factor, even with optimum row and column ordering and well-tuned sparse matrix algorithms. For $N = 1024 \times 1024 = 2^{20}$, that would be about $5 \times 2^{23} \approx 8{,}000{,}000$ nonzero elements, or about 128 MB of memory. The number of operations also grows proportionally to $N^{1.5}$. In our tests, the memory required by Agrawal's M-Estimator method grows like $N^{1.15}$, and its running time like $N^{1.5}$. For these reasons, direct system solving methods are impractical for multi-megapixel gradient maps.

Local iteration methods also set up an $N \times N$ system of equations from local constraints, but then solve the system iteratively, as in the Gauss–Seidel algorithm. Namely, they start with some initial guess for the height map, and then repeatedly use each equation in turn to recompute one height value, assuming the neighbors are fixed—until all the heights appear to stabilise. This approach was first described by Horn and Brooks [23,2].

Local iteration methods can use the same Poisson-like equations used by direct methods, including locally tuned formulas that take weight map into account. In addition, they can also handle moderately non-linear equations without the need for the Newton-Raphson linearization and its additional outer loop.

Local iteration methods require little memory space, which grows proportional to N rather than $N^{1.5}$. Their main drawback is excessive processing time. Although each iteration requires only $O(N)$ operations, the number of iterations needed to reduce the initial error E below a specified tolerance ε is usually proportional to $\log(E/\varepsilon)$ times the square of the image's diameter, that is to $N\log(E/\varepsilon)$; so the total running time is proportional to $N^2\log(E/\varepsilon)$.

This inefficiency can be explained by considering the effect of the Gauss–Seidel iteration on each Fourier component of the current error map δ (the difference between the current height map and the correct one). Each iteration essentially reduces the amplitude of ϕ by a factor $\alpha = \cos(2\pi/\omega)$, where ω is the wavelength of ϕ. In a square mesh with N pixels, the lowest-frequency component has $\omega = \sqrt{2N}$, so $\alpha = \cos(2\pi/\sqrt{2N}) \approx 1 - \pi^2/N$. Therefore, the number of iterations needed to reduce its initial amplitude from E to ε is proportional to $\log(E/\varepsilon)/\log(1 - \pi^2/N) \approx N/\pi^2\log(E/\varepsilon)$. On the other hand, any components of the error map δ whose wavelength is only a few pixels will be reduced to insignificance with $O(1)$ iterations. Indeed, the correction of the initial guess error propagates across the height map by a diffusion-wave process, mathematically similar to the spreading of heat in a solid plate.

One way to accelerate the convergence of local iterative methods is to use *multi-scale techniques* as suggested by Terzopoulos [24,25] in 1986. This is the approach we use in our algorithm; see Section 5.

Other methods. The *Kernel method* introduced by Ng et al. [26] assumes a sparse gradient field, and reduces gradient integration to a high-dimensional data fitting problem using certain kernel (basis) functions. This approach can accomodate irregularly spaced gradient sampling points and is claimed to provide 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 much more expensive in terms of time and memory usage.

The "pyramid-based" method of by Chen, Wang and Wang [27] (which does not accept weights) uses the idea of working at several scales of resolution, but is not truly "multi-scale" since the problem is not reduced and expanded between successive scales. Instead, their algorithm solves a sequence of $N \times N$ Poisson systems. System k related each height $z[u, v]$ to heights $z[u \pm 2^k, v \pm 2^k]$. Expensive line integrals are used to compute the right-hand side for each equation. While the use of longer strides substantially improved convergence, the speed and accuracy reported by the authors were still quite inferior to those of Fourier-based algorithms.

2.2. Computing weights from the gradients

Several papers on path- or Poisson-based integrators that use a weight map assume that the latter is not given by the user, but is to be computed from the given gradient data. Typically, these methods assume that errors in measured gradient samples can be detected by checking the curl-free condition (2). They therefore assign the weight of each data sample as some decreasing function of the local curl, estimated by some discrete version of (2) [10,11].

Other methods adjust the weights iteratively by analyzing the integrated height map [28,15]. Other methods adjust the weights iteratively by analyzing the integrated height map. An example is the method described in 2009 by Durou *et al.* [28]. Another example is the "ℓ_1 error correcting" approach described by Reddy and others [15]. They observed that the curl filtering performed by Fourier or Poisson integrators is equivalent to approximating the given gradient data by the nearest curl-free field in the ℓ_2 error metric ("root of sum of squares"). Since ℓ_2 approximations are notoriously sensitive to outliers, they propose instead to use the curl-free map that is nearest to the data in the ℓ_1 metric ("sum of absolute values"). However, the ℓ_1 measure of a vector v is equal to a weighted version of the ℓ_2 metric, for properly chosen weights (that depend on v). Therefore, these ℓ_1-based integrators can be viewed as variants of the weighted ℓ_2-based methods described in Section 2.1, where the weight map is computed iteratively by specific algorithms.

Most of these weight acquisition and adjustment techniques can be used with our integrator as well. However, as we discussed in Section 1, the gradient map may not contain enough information to determine the location of cliffs and other data problems. In particular, there may be large cliffs in regions where the curl is zero. For reliable results, the weight map should be obtained by some other means, such as analysis of shadows and specular highlights under varied light conditions [14,12], polarimetry [13], geometric stereo hints [29], and prior knowledge about the scene's geometry. Accordingly, in this paper, we consider only the central integration problem, assuming that the weight map is fixed and given as input to the integrator.

3. The weighted Poisson system

Our integrator builds the linear equation system for a weighted variant of the discrete Poisson problem, and solves it by a multiscale version of the Gauss–Seidel (or Gauss–Jacobi) iterative algorithm. In this section we formulate the continuous version of the Poisson method for error-free gradient fields, and then obtain a discrete version that takes weights into account.

3.1. The continuous Poisson formulation

Conceptually, in the Poisson approach, we differentiate both sides of the defining Eq. (1) and add them to obtain a single functional equation

$$\mathbf{L}(Z)(x,y) = \mathbf{H}(F,G)(x,y) \tag{4}$$

where

$$\mathbf{L}(Z) = \frac{\partial^2 Z}{\partial x^2} + \frac{\partial^2 Z}{\partial y^2} \quad \mathbf{H}(F,G) = \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} \tag{5}$$

Eq. (4) says that the Laplacian $\mathbf{L}(Z)$ of the height field Z computed from Z itself should be equal to the Laplacian $\mathbf{H}(F,G)$ computed from the given gradient fields F, G [25,23]. It turns out that Eq. (4) has a solution Z for *any* differentiable functions F, G.

On the other hand, being a second-order differential equation, it allows for spurious solutions that do not satisfy formula (1). Namely, the homogeneous version $\mathbf{L}(Z) = 0$ is satisfied by an arbitrary linear function of position $Z(x, y) = A\,x + B\,y + C$, which when added to any solution of Eq. (4) will yield infinitely many additional solutions. In contrast, the original equations (1) have only one degree of freedom, the offset C. The two degrees of freedom corresponding to A and B can be fixed by suitable global or boundary constraints derived from the gradient data f and g. We will defer this issue until after we describe our algorithm.

3.2. Discretizing the heights

In order to discretize Eq. (4), the height function Z is represented by a *height map*, an array of height samples $z[u, v]$, nominally placed at *height sampling points* $q[u, v]$; and the derivative operators in Eq. (4) are replaced finite difference operators applied to the arrays z, f, and g. The differential Eq. (4) then becomes a system of linear equations, whose unknowns are the elements of z array.

In order to obtain comparable derivative estimates for both sides of Eq. (4), we assume that the height sampling points $q[u, v]$ are displaced from the gradient sampling points $p[u, v]$ by half a step in each direction. Specifically, we assume that $p[u, v]$ is the point $(u + 1/2, v + 1/2)$ in $\mathbb{R}^2$, while $q[u, v]$ is the point (u, v). This convention is illustrated in Fig. 3.

Therefore, if the gradient map has n_x samples in x and n_y samples in y, we can assume that the nominal domain D of the problem is the rectangle $[0\_\ n_x] \times [0\_\ n_y]$ of $\mathbb{R}^2$. Each pixel $[u, v]$ of the derivative maps f and g can be identified with the unit square centered at $p[u, v]$, with $q[u, v]$ and $q[u + 1, v + 1]$ at opposite corners. Note that the height map z, on the other hand, has $n_x + 1$ columns and $n_y + 1$ rows, and its pixels are centered at the corners of the gradient pixels.

In practice, the derivative values $f[u, v]g[u, v]$ are almost always averages of the derivatives $\partial Z/\partial x$ and $\partial Z/\partial y$ over a neighborhood of the point $p[u, v]$, with some *gradient sampling kernel*, that is assumed to be symetric about $p[u, v]$ and do overlap in part the adjacent kernels. Likewise, the computed height $z[u, v]$ is an estimate for an average of the height Z around the point $q[u, v]$, taken with some other *height sampling kernel*.

We assume that the weight map is given as an array w of nonnegative numbers, with the same dimensions as f and g. In what follows, we assume that $w[u, v]$ is zero whenever the pixel $[u, v]$ is outside the domain D. As we shall see, only the relative magnitudes of the weights are important; multiplying all weights by a positive scale factor will not affect the result.

3.3. The discrete equations

In the discrete version, the differential operators **L** and **H** of Eq. (4) are replaced by finite difference estimators $\mathcal{L}$ and $\mathcal{H}$ (which are often called "Poisson kernels" in the literature.)

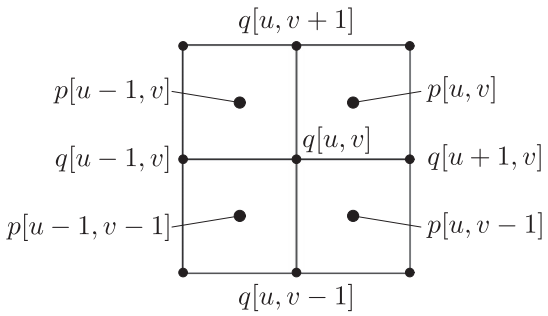

Fig. 3. The gradient and height sampling points around point $q[u, v] = (u, v)$.

To make the formulas more readable, we will use $z_{\circ\circ}$ for a generic height sample $z[u,v]$, and the following notations for its four neighboring height values:

$$z_{-\circ} = z[u-1, v] \quad z_{\circ-} = z[u, v-1]$$
$$z_{+\circ} = z[u+1, v] \quad z_{\circ+} = z[u, v+1]$$

Moreover, we use the following symbols for derivative values that are interpolated by our algorithm, from the maps f and g, at the midpoints of the edges between $q[u, v]$ and its four neighbors:

$$f_{-\circ} \approx \tfrac{\partial Z}{\partial x}(u-\tfrac{1}{2}, v) \quad g_{\circ-} \approx \tfrac{\partial Z}{\partial y}(u, v-\tfrac{1}{2})$$
$$f_{+\circ} \approx \tfrac{\partial Z}{\partial x}(u+\tfrac{1}{2}, v) \quad g_{\circ+} \approx \tfrac{\partial Z}{\partial y}(u, v+\tfrac{1}{2})$$

We will also use $w_{-\circ}$, $w_{+\circ}$, $w_{\circ-}$, and $w_{\circ+}$ for the *edge reliability weights*, assigned by our algorithm to the interpolated slopes $f_{-\circ}$, $f_{+\circ}$, $g_{\circ-}$, and $g_{\circ+}$, respectively. See Fig. 4. The interpolation formulas for these derivatives and weights are described in Section 4.

If all the weights are 1, we could use the discrete operators $\widetilde{\mathcal{L}}(z)$ and $\widetilde{\mathcal{H}}(f,g)$ to approximate $\mathbf{L}(Z)$ and $\mathbf{H}(F,G)$, where

$$\widetilde{\mathcal{L}}(z)[u,v] = +(z_{\circ+} - z_{\circ\circ}) - (z_{\circ\circ} - z_{\circ-}) + (z_{+\circ} - z_{\circ\circ}) - (z_{\circ\circ} - z_{-\circ}) \tag{6}$$

$$\widetilde{\mathcal{H}}(f,g)[u,v] = f_{+\circ} - f_{-\circ} + g_{\circ+} - g_{\circ-} \tag{7}$$

The discrete version of Eq. (4) is then the set of equations

$$\widetilde{\mathcal{L}}(Z)[u,v] = \widetilde{\mathcal{H}}(f,g)[u,v] \tag{8}$$

for all $[u, v]$ in the domain of z. Note that the first term $z_{\circ+} - z_{\circ\circ}$ in formula (6) is another estimate for the derivative $\partial Z/\partial x$ on the midpoint of the horizontal grid edge between $q[u, v]$ and $q[u+1, v]$, derived from the height map z, that may be compared to the interpolated derivative datum $f_{+\circ}$. Similarly, the other terms of formula (6) are numerical height derivatives estimates that are compared to the derivative data $f_{-\circ}$, $g_{\circ-}$, and $g_{\circ+}$. Inspired by this observation, we split each Eq. (8) into four equations: For each edge incident to $q[u, v]$, we write a finite difference equation:

$$\begin{aligned} z_{+\circ} - z_{\circ\circ} &= +f_{+\circ} \quad z_{\circ+} - z_{\circ\circ} = +g_{\circ+} \\ z_{-\circ} - z_{\circ\circ} &= -f_{-\circ} \quad z_{\circ-} - z_{\circ\circ} = -g_{\circ-} \end{aligned} \tag{9}$$

In general, the system (9) is overdetermined, as it has $\approx 2N$ equations (after eliminating duplicates) on $\approx N$ unknowns. We look for the weighted least-squares near-solution, that minimizes the weighted sum of squared differences between the two sides of those equations. For this sum we use the weights $w_{-\circ}$, $w_{+\circ}$, $w_{\circ-}$, and $w_{\circ+}$ of the interpolated edge data. The least-squares solution turns out to be given by another system of N linear equations of the N unknown heights. Each equation says that some height value $z[u, v]$ is equal to the weighted average of its four neighbors, displaced by the interpolated edge derivatives and weighted by the edge weights:

$$\begin{aligned} z_{\circ\circ} = &+\frac{w_{-\circ}}{w_{\circ\circ}}(z_{-\circ} + f_{-\circ}) + \frac{w_{+\circ}}{w_{\circ\circ}}(z_{+\circ} - f_{+\circ}) + \frac{w_{\circ-}}{w_{\circ\circ}}(z_{\circ-} + g_{\circ-}) \\ &+ \frac{w_{\circ+}}{w_{\circ\circ}}(z_{\circ+} - g_{\circ+}) \end{aligned} \tag{10}$$

where $w_{\circ\circ}$ is the *total vertex weight*,

$$w_{\circ\circ} = w_{\circ-} + w_{\circ+} + w_{-\circ} + w_{+\circ} \tag{11}$$

Note that only the relative values of the edge weights are relevant. Rearranging Eq. (10) to separate unknown and known terms, we get

$$-\mathcal{L}(z)[u,v] = -\mathcal{H}(f,g)[u,v] \tag{12}$$

where

$$-\mathcal{L}(z)[u,v] = +z_{\circ\circ} - \frac{w_{\circ-}}{w_{\circ\circ}} z_{\circ-} - \frac{w_{\circ+}}{w_{\circ\circ}} z_{\circ+} - \frac{w_{-\circ}}{w_{\circ\circ}} z_{-\circ} - \frac{w_{+\circ}}{w_{\circ\circ}} z_{+\circ} \tag{13}$$

and

$$-\mathcal{H}(f,g)[u,v] = -\frac{w_{\circ-}}{w_{\circ\circ}} g_{\circ-} + \frac{w_{\circ+}}{w_{\circ\circ}} g_{\circ+} - \frac{w_{-\circ}}{w_{\circ\circ}} f_{-\circ} + \frac{w_{+\circ}}{w_{\circ\circ}} f_{+\circ} \tag{14}$$

Note that when an edge has zero weight, the corresponding term drops out in Eqs. (10), (13), and (14). Along the lower margin ($v = 0$), for example, Eq. (12) will relate $z[u, v]$ to its three neighbors $z_{\circ+}$, $z_{+\circ}$ and $z_{-\circ}$, ignoring the non-existing term $z_{\circ-}$. We refer to formula (12) as the *axial equation* for $z[u, v]$.

The value $w_{\circ\circ}$ can be interpreted as a local reliability weight for the computed height $z[u, v]$. Values of $z[u,v]$ that have $w_{\circ\circ} = 0$ should be considered meaningless in subsequent computations.

Failover to diagonal equations: When many data values around $q[u, v]$ are missing, all four edge weights $w_{\circ+}$, $w_{\circ-}$, $w_{+\circ}$, $w_{-\circ}$ may be zero. In that case we replace the axial Eq. (12) by a *diagonal equation*, that relates $z_{\circ\circ}$ to its four diagonal neighbors

$$z_{--} = z[u-1, v-1] \quad z_{+-} = z[u+1, v-1]$$
$$z_{-+} = z[u-1, v+1] \quad z_{++} = z[u+1, v+1]$$

Namely, we use the Eq. (12) with

$$-\mathcal{L}(z)[u,v] = z_{\circ\circ} - \frac{w_{--}}{w_{\circ\circ}} z_{--} - \frac{w_{-+}}{w_{\circ\circ}} z_{-+} - \frac{w_{+-}}{w_{\circ\circ}} z_{+-} - \frac{w_{++}}{w_{\circ\circ}} z_{++} \tag{15}$$

and

$$\begin{aligned} -\mathcal{H}(f,g)[u,v] = &+\frac{w_{--}}{w_{\circ\circ}}(+f_{--} + g_{--}) + \frac{w_{-+}}{w_{\circ\circ}}(+f_{-+} - g_{-+}) \\ &+ \frac{w_{+-}}{w_{\circ\circ}}(-f_{+-} + g_{-+}) + \frac{w_{++}}{w_{\circ\circ}}(-f_{++} - g_{++}) \end{aligned} \tag{16}$$

where the total vertex weight $w_{\circ\circ}$ is redefined as

$$w_{\circ\circ} = w_{--} + w_{-+} + w_{+-} + w_{++} \tag{17}$$

Uncoupled height values: When both the axial and the diagonal equations fail (that is, when formulas (11) and (17) are zero), we consider $z[u, v]$ to be *uncoupled*. We handle this special case by setting the height $z[u, v]$ to zero and removing that unknown and its corresponding equation from the system. In any case, the values assigned to uncoupled pixels $z[u, v]$ by these fixes will not affect any neighboring z value, unless that neighbor too is uncoupled.

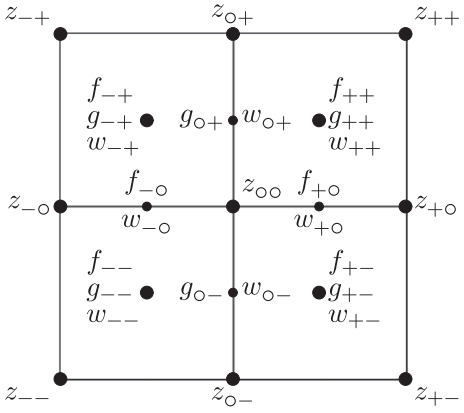

Fig. 4. Notations for the interpolated height, gradient and weight values around point $q[u, v]$.

3.4. Assembling the system

After excluding the uncoupled heights, there is a one-to-one correspondence between the discrete Poisson equations (12) (axial or diagonal) and the height sampling points $q[u, v]$; so the number

of equations is equal to the number of unknowns. We gather all those equations in a linear equation system

$$\mathbf{M}\hat{\mathbf{z}} = \mathbf{b} \tag{18}$$

where $\hat{\mathbf{z}}$ is a vector with $N = (n_x + 1)(n_y + 1)$ elements, which are the unknown heights $z[u, v]$; $\mathbf{M}$ is the $N \times N$ coefficient matrix of the left-hand sides (13) of those of equations; and $\mathbf{b}$ is the N-vector containing the corresponding right-hand sides (14).

Note that $\mathbf{M}$ depends on the weights $w[u,v]$ but not on the given gradients f and g; while $\mathbf{b}$ depends on w, f, and g. Note also that any constant height map $z[x, y] = C$ for all x, y and any c satisfies the homogeneous system $\mathbf{M}\hat{\mathbf{z}} = 0$. As a consequence, the solution of system (18) is determined only up to an additive constant.

It may happen that the missing data partition the height map into two or more unrelated components, such that no equation relates height values belonging to different components. In that case, the system (18) splits into two or more independent systems, with one indeterminate offset for each component.

4. Estimating the edge derivatives

To compute the right-hand side of Eq. (14) we need the estimate $g_{\circ+}$ for the derivative $\partial Z/\partial y$ at the edge midpoint $r = (u, v + \frac{1}{2})$ between $q[u, v]$ and $q[u, v+1]$, taking weights into account. For that purpose, we use the four data values $g_a = g[u - 2, v]$, $g_b = g[u - 1, v]$, $g_c = g[u, v]$, and $g_d = g[u + 1, v]$, which are assumed to be the derivatives sampled around the points $(u - \frac{3}{2}, v + \frac{1}{2})$, $(u - \frac{1}{2}, v + \frac{1}{2})$, $(u + \frac{1}{2}, v + \frac{1}{2})$, and $(u + \frac{3}{2}, v + \frac{1}{2})$, respectively. Note that the signed horizontal distances from these points to r are $-\frac{3}{2}, -\frac{1}{2}, +\frac{1}{2}$, and $+\frac{3}{2}$, respectively. We also use the four reliability weights w_a, w_b, w_c, and w_d of these data values. See Fig. 5.

Considering consecutive pairs of these four values, by linear interpolation or extrapolation we obtain three estimates for the derivative $\partial Z/\partial y$ at r:

$$\begin{aligned} g_- &= (3g_b - g_a)/2 \\ g_\circ &= (g_b + g_c)/2 \\ g_+ &= (3g_c - g_d)/2 \end{aligned} \tag{19}$$

Given the interpretation of $w[u, v]$ as the reciprocal of the variance of $g[u, v]$ (see Section 1.3), the weights of these estimates are

$$\begin{aligned} w_- &= 4/(9/w_b + 1/w_a) \\ w_\circ &= 4/(1/w_b + 1/w_c) \\ w_+ &= 4/(9/w_c + 1/w_d) \end{aligned} \tag{20}$$

We then take a weighted average of the three estimates, and assign it the appropriate weight:

$$\begin{aligned} w_{\circ+} &= w_- + w_\circ + w_+ \\ g_{\circ+} &= \frac{w_- g_- + w_\circ g_\circ + w_+ g_+}{w_{\circ+}} \end{aligned} \tag{21}$$

As usual, any samples that lie outside the domain of the g-map are assigned zero weight, so their value will ignored in this computation. Note that any estimate of g_-, $g_\circ$, or g_+ that depends on a zero-weight sample will itself have zero weight, and therefore will not contribute to the final result. If all three weights $w_-, w_\circ$, and w_+ are zero, the final weight $w_{\circ+}$ will be zero, and therefore $g_{\circ+}$ will be excluded from formula (14). In this case, we set arbitrarily $g_{\circ+} = g_\circ$.

We denote this computation by

$$(g_{\circ+}, w_{\circ+}) = \mathit{Interpolate}(g_a, w_a, g_b, w_b, g_c, w_c, g_d, w_d) \tag{22}$$

The same algorithm is used to estimate $g_{\circ-}$. To estimate $\partial Z/\partial x$ at the midpoint $s = (u + \frac{1}{2}, v)$ of a horizontal edge, we use the same function, applied to the four vertically adjacent samples $f_a = f[u, v - 2]$, $f_b = f[u, v - 1]$, $f_c = f[u, v]$, and $f_d = f[u, v + 1]$ and their respective weights., displaced and/or rotated, are used to estimate $g_{\circ-}$, $f_{+\circ}$, and $f_{-\circ}$.

5. Multiscale solver

The Poisson-like system (18) is usually too large to solve by direct elimination methods, therefore we use the iterative Gauss–Seidel method. Note that the matrix M is large but quite sparse, with at most five non-zero terms in each row. We use the same ordering for the heights in $\hat{\mathbf{z}}$ and for the rows of $\mathbf{M}$, so that the diagonal elements of $\mathbf{M}$ are all 1, while the off-diagonal coefficients are non-positive and add up to -1. These features ensure that the iteration converges. Furthermore, each equation has at most five non-zero coefficients in the left-hand side, so the system's coefficient matrix M uses only $O(N)$ storage, and the product $M\hat{\mathbf{z}}$ can be computed in $O(N)$ time.

As discussed in Section 2, for faster convergence we use the Terzopoulos's multiscale approach [24]. Namely, to obtain the initial guess for the solver, we reduce the gradient arrays f, g to half their original width and height, compute from this reduced-scale data a reduced-scale height map z, and expand the latter to the full scale. The reduced problem is solved recursively in the same way.

In other words, we construct a pyramid of horizontal derivative maps $f^{(k)}$, where $f^{(0)} = f$ and each map $f^{(k+1)}$ is a reduced copy of the previous one $f^{(k)}$. The same procedure yields a pyramid $g^{(k)}$ of vertical gradient maps from g. The reduction stops at a level m such that $f^{(m)}$ and $g^{(m)}$ are small enough to be integrated efficiently (e.g. by Gauss–Seidel or even Gaussian elimination), resulting in a height map $z^{(m)}$ at the same scale. Then we integrate the maps $f^{(k)}$ and $g^{(k)}$, in order of decreasing k, each time with an iterative method using the solution $z^{(k+1)}$ as the initial guess, and obtaining a more detailed solution $z^{(k)}$. The height map $z^{(0)}$ is the result. See Fig. 6. If the reduction and expansion steps are done properly—in particular, if the connectivity is preserved within regions that are separated by height discontinuities—the recursively computed initial guess will be close to the correct solution in its broad features. The iterative solver will then converge quickly, since it has only to adjust details at a scale of one or two pixels. Terzopoulos [24] uses trivial subsampling for reduction and bilinear interpolation for expansion. Although those choices are adequate for smooth continuous surfaces, they do not perform well next to domain edges and discontinuities. To get the benefit of multiscale in those cases too, we use more elaborate reduction and expansion formulas, described in Sections 5.2 and 5.3, that take the weight map into account.

5.1. The algorithm

The central part of our multiscale integration algorithm is the recursive procedure *ComputeHeights* whose pseudocode is given in Fig. 7. As discussed in Section 3.2, the input maps f, g, w must have the same dimensions, namely n_x columns by n_y rows. while the output height map z will have $n_x + 1$ columns and $n_y + 1$ rows. The parameter κ is the maximum number of Gauss–Seidel iterations allowed, and τ is a numeric convergence criterion, both for scale *0*. The constants λ and *smallsize* are internal parameters.

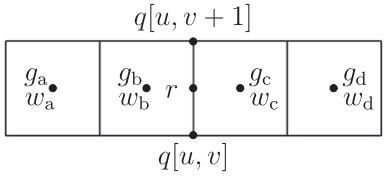

Fig. 5. Interpolating the derivative $\partial Z/\partial y$.

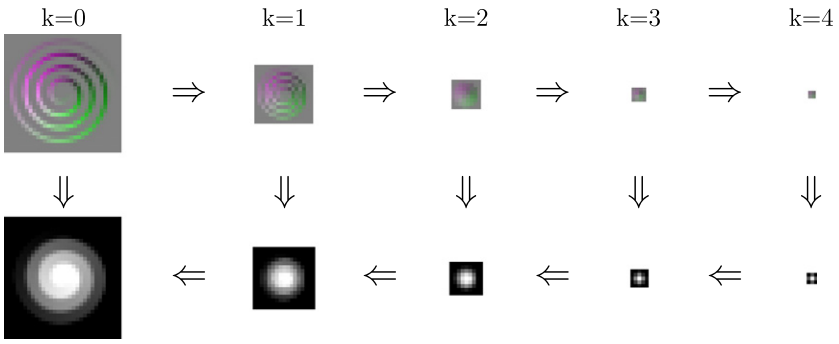

Fig. 6. Schematic of multiscale integration method with maximum scale $m = 4$.

The procedure BuildSystem constructs the system's matrix $M = \mathbf{M}$ and the right-hand side vector $b = \mathbf{b}$ as described in Section 3. The procedure *SolveSystem*(M, b, z, τ) is the Gauss–Seidel algorithm, that repeats $\hat{\mathbf{z}} \leftarrow \mathbf{b} - -(\mathbf{M} - -\mathbf{I})\hat{\mathbf{z}}$ at most κ times, stopping when every height changes by less than τ from its previous iteration. The procedures ReduceGradientMap, ReduceWeightMap, and ExpandHeightMap are explained in Sections 5.2 and 5.3 below.

We implemented our algorithm as a set of library procedures and a main program (`slope_to_height`) written in the GNU dialect of C. The code is available at the URL http://www.ic.unicamp.br/stolfi/EXPORT/projects/photo-stereo/.

5.2. Reducing the slope and weight maps

At each level in the recursion, the procedures *ReduceGradientMap* and *ReduceWeightMap* are used to compute the reduced-scale data f', g' and w'. By assumption, every sample $f'[u,v]$ should be the value of the derivative $\partial Z/\partial x$ averaged over the pixel of the reduced map f' with indices $[u, v]$; which corresponds to the 2×2 block of pixels of the original map f centered at the point $(2u + 1, 2v + 1)$. Therefore, the value of $f'[u,v]$ should be the unweighted average of the original samples $f_{--} = f[2u, 2v]$, $f_{+-} = f[2u + 1, 2v]$, $f_{-+} = f[2u, 2v + 1]$, and $f_{++} = f[2u + 1, 2v + 1]$. Then the weight $w'[u, v]$ of the reduced sample should be 16 divided by the sum of the reciprocals of the four original weights.

However, this approach would give $w'[u, v] = 0$ (or a very low value) if any of the four original weights is zero (or very low). We use a more robust formula, based on the observation that the average of any two diagonally opposite samples in the block is an acceptable estimate for the derivative at the center of the block. Namely, we first compute the two estimates

$$f'_a = (f_{--} + f_{++})/2 \quad f'_b = (f_{-+} + f_{+-})/2$$

with the respective weights

$$w'_a = \frac{4}{\frac{1}{w_{--}} + \frac{1}{w_{++}}} \quad w'_b = \frac{4}{\frac{1}{w_{-+}} + \frac{1}{w_{+-}}}$$

We then take a weighted average of the two estimates, and assign to it the appropriate weight:

$$w'[u, v] = w'_a + w'_b$$
$$f'[u, v] = \frac{w'_a f'_a + w'_b f'_b}{w'_a + w'_b}$$

As usual, samples that lie outside the domain of the f-map are assigned an arbitrary value with zero weight. Note that $w'[u, v]$ will be zero only if both diagonals of the block include at least one sample with zero weight. The same formulas are used for the vertical derivatives map g.

Observe that the integration operator is linear when the inputs are the derivatives f, g, but non-linear when the inputs are the surface normals $\vec{n}$. Therefore, when reducing the problem to a smaller scale, one must average the gradients f and g within each 2×2 block, rather than the normal directions $\vec{n}$.

ComputeHeights(f, g, w, κ, τ)
1. If $f.nx < smallsize$ and $f.ny < smallsize$ then
2. $z \leftarrow (0, 0, \ldots, 0)$;
3. else
4. $f' \leftarrow ReduceGradientMap(f, w)$;
5. $g' \leftarrow ReduceGradientMap(g, w)$;
6. $w' \leftarrow ReduceWeightMap(w)$;
7. $z' \leftarrow ComputeHeights(f', g', w', \lambda\kappa, \tau/\lambda)$;
8. $z \leftarrow ExpandHeightMap(z')$;
9. $M, b \leftarrow BuildSystem(f, g, w)$;
10. $z \leftarrow SolveSystem(M, b, z, \kappa, \tau)$;
11. Return z.

Fig. 7. The main procedure of the integrator.

5.3. Expanding the height map

The procedure ExpandHeightMap computes an estimate z for the heights at some scale k, given the heights z' computed for scale $k + 1$. Let $z'_a = z'[u-1, v], z'_b = z'[u, v], z'_c = z'[u+1, v], z'_d = z'[u+2, v]$ and w'_a, w'_b, w'_c, w'_d be the respective vertex weights as defined in Section 3.3. We use the Interpolate function derived in Section 4 to estimate the height z'_x at the midpoint of $q[u, v]$ and $q[u + 1, v]$:

$$(z'_x, w'_x) = Interpolate(z'_a, w'_a, z'_b, w'_b, z'_c, w'_c, z'_d, w'_d)$$

In the same way we compute height estimates for z'_y at the midpoint of $q[u, v]$ and $q[u, v + 1]$, and the respective weight w'_y. Then we set

$$z[2u, 2v] = 2z'[u, v]$$
$$z[2u + 1, 2v] = 2z_x$$
$$z[2u, 2v + 1] = 2z_y$$
$$z[2u + 1, 2v + 1] = 2\frac{w'_x z'_x + w'_y z'_y}{w'_x + w'_y}$$

for all u, v where the left-hand side is defined. The factor of 2 accounts for the change in pixel spacing from scale $k + 1$ to scale k. As a special case, if both w'_x and w'_y are zero, we set $z[2u + 1, 2v + 1]$ to the simple average of z'_x and z'_y.

5.4. Ordering the equations

In the Gauss–Seidel iterative method, the order of the variables may have a substantial impact on the speed of convergence.

Ideally, variables whose neighbours are closest the correct values should be recomputed first, so that good values will tend to replace bad estimates, rather than the other way around. In general, points $q[u, v]$ that have higher total vertex weights by formula (11) or (17) are expected to be more correct than those with lower weights. Therefore, we reorder the equations so that height values with higher total weight are computed first. For that purpose we create a directed graph whose vertices are the unknowns $z[u, v]$, with an edge from $z[u, v]$ to $z[u', v']$ when both occur in same equation and the vertex weight of $z[u, v]$ is higher than that of $z[u', v']$. The equation ordering is then obtained by a topological sort [30] of the vertices of this graph.

5.5. The indeterminate linear term

Unlike Eq. (1), the second-order continuous Poisson Eq. (4) has a solution for *any* functions F,G. Actually Eq. (4) is under-determined because the homogeneous version $\mathbf{L}(Z) = 0$ is satisfied by any affine function of position $Z(x,y) = Ax + By + C$, which when added to any solution of Eq. (4) will yield infinitely many solutions. The additive constant C is truly indeterminate, but the linear part $Ax + By$ is an artifact of the method.

In a single-scale solution, the term $Ax + By$ becomes determined in part by the gradient values at the effective domain boundaries (the boundaries of D as well as the points adjacent to zero-weight data samples). At these points the weighted Laplacians $\mathcal{L}(z)$ and $\mathcal{H}(f, z)$ become a combination of first- and second-order derivatives, so the equations become sensitive to gradients rather than just curvature. If the iteration is stopped early, the linear term is also determined in part by the average linear term contained in the starting guess.

Note that this overall linear term is largely preserved by the expansion of the height maps. This spurious linear term is often conspicuous in height maps integrated by single-scale iterative methods, since the iteration usually has to be terminated well before it converges. In those methods one could remove that term by computing the average x-gradient over D from the data $f[u, v]$ and from the height field z, and then subtract the appropriate linear term Ax from the latter to make the two values equal. A similar correction would take care of the term linear in y. However, this fix will not correct localized errors in the mean gradients that may occur due to premature termination.

The height map computed by the multiscale integration algorithm is usually free from such spurious linear terms, and therefore does not need such post-processing. To understand why, observe that the gradient map reduction method largely preserves the average gradient over the domain. Therefore, at the coarsest scale, the computed height map, which is an exact solution, will have an approximately correct average gradient too.

5.6. Analysis

To analyse the efficiency of this algorithm, we consider how it operates in the frequency domain. When the gradient maps are reduced, the higher-frequency components of the data are lost, while the 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. Thus, each Fourier component of the height map will be computed at the scale where its wavelength is only a few pixels; which requires only a small number of iterations by the Gauss–Seidel solver.

In any case, the algorithm limits the number of iterations at each scale k to $\kappa\lambda^k$. Therefore, the time spent in the Gauss–Seidel solver at that scale will be proportional to $N\kappa(\lambda/4)^k$. The total system-solving time for all scales will be $1 + \lambda/4 + \lambda^2/4^2 + \cdots \lambda^m/4^m)$ times the work of scale 0. As long as $\lambda < 4$, this sum is less than $1/(1 - \lambda/4)$, therefore that cost will be $O(N)$. The cost of building the linear system at each scale k, including the topological sort, is proportional to $N/4^k$; therefore that part of the cost too is $O(N)$.

5.7. Limitations

The multiscale approach is not valid in situations like Fig. 8, when the effective domain (the region where the weights are non-zero) includes a long and narrow corridor.

Note that the ReduceWeightMap procedure must ensure that any pixels that might be crossed by a cliff have their weight set to zero. It follows that if the corridor is t pixels wide and T pixels long, it will be disconnected and/or obliterated at the scale $k \approx \log_2 t$. Then the intermediate map $z^{(k)}$ will be useless, and at level $k - 1$ the Gauss–Seidel algorithm would have to compute the heights along the corridor from a incorrect guess—which will require

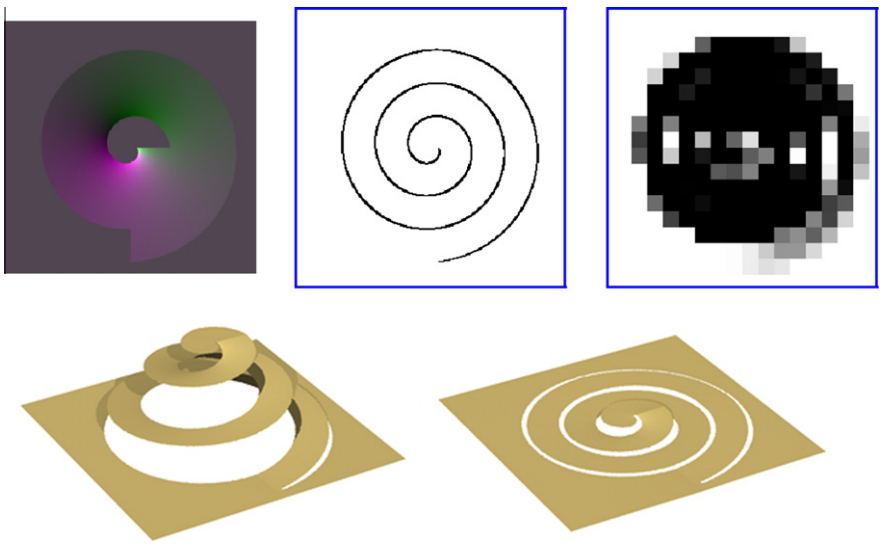

Fig. 8. A pathological case for multiscale iterative integration. On top: maps $f^{(0)}$ and $w^{(0)}$ (256×256), and the reduced map $w^{(4)}$ (16×16). Bottom: heights z obtained by direct solution of the Poisson System [11] (correct) and by our algorithm with 200 iterations per level (quite incorrect).

$\Omega((T/t)^2)$ iterations, i.e. $\Omega(N^2)$ in the worst case possible. This problem occurs also when the weight map has zero elements scattered with mean spacing t in a region with diameter T. For these pathological cases, direct solution of the linear system may be much faster than our iterative method.

6. Test datasets

In this section we describe our test datasets, which are similar to those proposed by Kettle and Schlüns in their survey [16], but include reference height maps for all cases. They comprise four virtual 3D scenes, and one real scene. See Table 1 and Fig. 9.

Virtual scenes: In these datasets each height field $Z(x, y)$ was defined by an algebraic formula. The `spdome` field is a hemispherical dome on a flat groundplane, C smooth except for a gradient discontinuity around its rim. The `sbabel` field is a spiral ramp connecting a flat floor with a horizontal platform at the center. The ramp is flanked by steep (but not vertical) walls with rounded shoulders. It is C smooth everywhere except at the ends of the ramp. The `cbramp` field is C-smooth everywhere except for the vertical cliffs on three sides. Finally, `mixwav` is the sum of two sinuisodal waves with different frequencies.

For these virtual scenes, each data sample $f[u, v]$ or $g[u, v]$ was obtained by evaluating the analytic derivative of $Z(x, y)$ on a 11×11 subsampling grid within a 2×2 pixel square centered at $p[u, v]$, and averaging those sub-samples with a 2D Hann window.

Table 1
Datasets used in our tests.

Set	Source	Description
`spdome`	Synth.	Spherical dome
`sbabel`	Synth.	Conical tower with spiral ramp.
`cbramp`	Synth.	Cubic ramp with cliffs
`mixwav`	Synth.	Low and high frequency waves.
`dtface`	Real	Face captured by 3D scanning

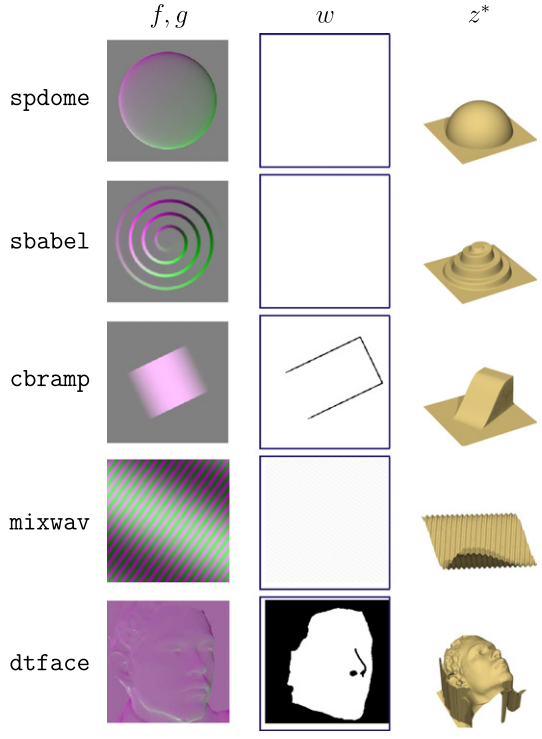

Fig. 9. Test datasets, showing the gradient maps f,g,w and the height map z^*.

The reference height map $z^*[u, v]$ was generated by averaging $Z(u, v)$ on a subsampling grid within the pixel centered at $q[u, v]$. The weights were set to 1 for all pixels and all datasets, except for `cbramp` where any pixel containing cliffs where masked out by hand.

The sampling procedure above resembles the photometric stereo gradient acquisition process in that it is oblivious to cliffs of the Z function, and is subject to sampling noise wherever the gradient ∇Z changes abruptly. Note that taking finite differences of the reference height map z^* will *not* yield adequate test data, since such gradients would be correct everywhere—even across cliffs. In that case the heights could be recovered by trivial path integration, with zero error.

Real scene: The `dtface` data set is derived from a digital mesh model of a human face consisting of *84,590* triangles, acquired from a live subject with a commercial 3D acquisition camera that combines structured lighting and geometric stereo [31]. We used the POV-Ray [32] raytracer to extract the gradient and height maps. The heights were obtained by rendering the scene with orthogonal projection and proper Z-dependent texture. The gradients were obtained by rendering the object using the in-built texturing function `slope` α which is defined as $P[u, v] = 0.5 + \arcsin(\vec{n}_\alpha[u, v])/\pi$, where α is 'x', 'y', or 'z', and $P \in [0, 1]$ is the scaled pixel intensity, and $\vec{n}_\alpha$ is the α component of the average surface normal at this pixel. The binary weight mask was manually created with an image editor, and excludes regions where the data is known to be unreliable or missing, such as background, hair, and under-chin.

7. Reference algorithms

We compared our integrator against the best methods available for each category, which are listed in Table 2.

For path integration (PI), we used the Fraile-Hancock [18] Combinatorial Surface Integration algorithm implemented in Matlab [36], using the *Norm* method to compute the MST. We modified the latter to accept a weight map and exclude from the MST any samples with zero weight.

For Fourier filtering, we used the Frankot–Chellappa (FC) algorithm [9] implemented in Matlab/Octave by Peter Kovesi [33]. It uses Matlab's FFT functions for the transform, and does not accept a weight map. We did not use Wei's reformulation [20] since that merely modified the frequency-domain filter coefficients to add extra smoothing.

The Poisson-based methods are represented by the unweighted Poison–Solver (UP) of Agrawal et al. [11], and its weighted variants M-Estimators (ME) and Diffusion with Affinely-Transformed kernels (AT), all three implemented in Matlab by their authors [34]. Algorithm UP uses a discrete cosine transform to solve the Poisson system, while AT and ME solve using Matlab's linear system solver. The Laplacian estimator used by AT depends on a variable number of z values, averaging to about 6 in our tests, while UP and ME use a 5-point estimator like ours. The original versions of AT and ME

Table 2
Integration methods used in the tests.

Code	Description	Type	Takes w
PI	Combinatorial Surface Integration [18]	Path integral	Yes
FC	Frankot-Chellappa [9,33]	Fourier integral	No
UP	Unweighted Poisson [11,34]	Poisson by DCT	No
AT	Affine-Transform Diffusion [11,34]	Poisson direct sol.	Yes
ME	M-Estimators [11,34]	Poisson direct sol.	Yes
MS	Our multiscale integration method [35]	Poisson multi-scale iter.	Yes

Table 3
Relative RMS errors of each method for gradient maps without noise.

Meth.	spdome e/R (%)	sbabel e/R (%)	cbramp e/R (%)	mixwav e/R (%)	dtface e/R (%)
PI	5.5	0.2	0.1	7.2	1.5
FC	0.2	0.2	120.4	42.9	48.8
UP	1.6	2.1	107.3	1.2	32.2
AT	5.2	12.4	0.4	2.3	4.0
ME	1.9	2.1	0.4	1.2	4.1
MS	0.5	0.6	2.6	0.9	2.3

attempt to identify noisy and outlier samples in the gradient maps by an iterative weight deduction loop; we modified the Matlab implementations to start this loop with the given weight map.

Method MS is our proposed multi-scale iterative integrator. For all runs, we set the maximum number κ of Gauss–Seidel iterations to 50, and the convergence threshold τ to *0.0005* (at scale 0), with internal parameters $\lambda = 2$ and *smallsize* = 2.

We did not test Ng's kernel method [26] since its time and memory requirements are much higher than other methods available, and its advantage seems to be mainly its ability to fill in gaps in the data. We considered testing Agrawal's Alpha-Surface and Energy Minimization algorithms [11] but we could not see how to modify them to accept our weight map. For datasets which do not have a weight map, we verified that those methods were about as accurate as FC, but considerably slower.

8. Results and discussion

We evaluated the accuracy and robustness of all integrators on the five datasets, comparing the integrated height map with the reference one. We also evaluated the execution time and memory usage of those algorithms that accept input weight maps.

8.1. Robustness and accuracy

To test the robustness and accuracy of our method, we compared its output with each of the algorithms in the Table 2, on all datasets. The results are shown in Table 3 and Figs. 10 and 11.

In order to analyse the behavior of the integration algorithms in presence of noise, we also tested the algorithms on the same gradient maps mixed with 30% of Gaussian noise with zero mean and unit variance. The result of these runs are shown in Table 4 and Figs. 12 and 13.

Evaluation: Ideally, the accuracy of a gradient integrator should be evaluated by comparing the computed z values with the "true" height field. However, even the synthetic datasets are affected by gradient sampling noise in regions of high curvature. The discretization of the gradient map implies loss of high-frequency detail, especially in regions of high curvature (such as around the rim of the spdome set and along the shoulders of the sbabel set). Therefore, due to sampling effects it is mathematically impossible to reconstruct the function Z exactly; the best one could hope for is to recover its Fourier components up to a certain cutoff frequency—which depends on the gradient sampling kernel. Thus, some errors are unavoidable in the discrete surface integration.

Another issue that arises is the fact that our method produces height maps with one pixel more in each axis, due its interpretation of gradient maps described in Section 3.2, while most other methods return height maps with the same size. To make a proper comparison, we generated ground-truth height maps with proper size for each integrator output.

In each case, we computed the RMS error e between the two integrated height fields, and the relative RMS error e/R, where R is the RMS value of the two height fields. The values of e and R are computed after shifting both depth maps to have zero mean, so as to compensate for the arbitrary integration constant C; and

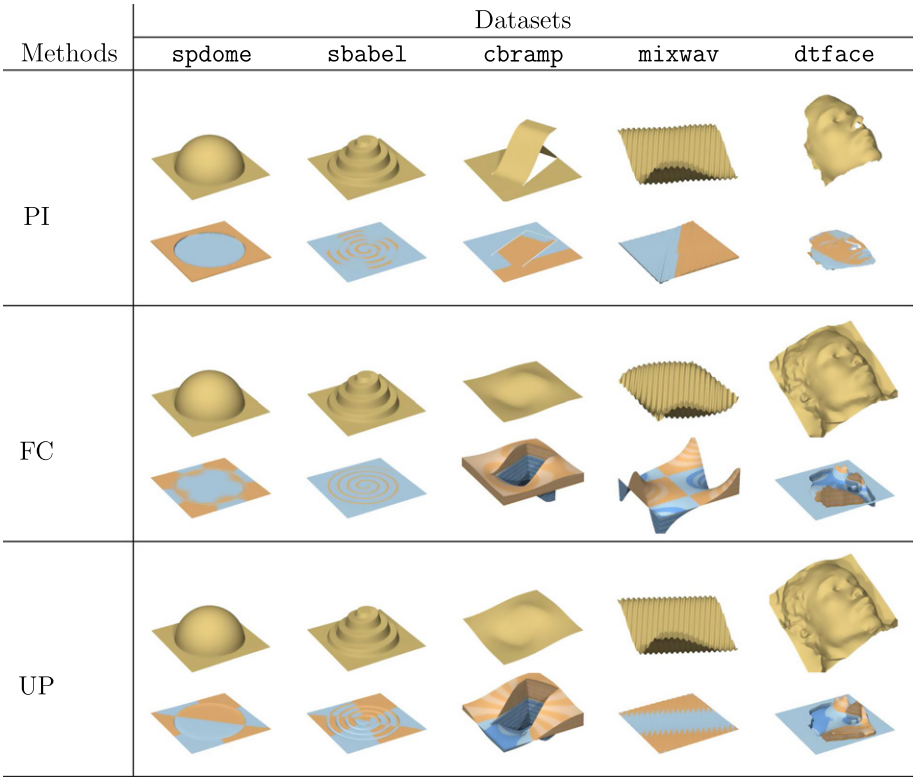

Fig. 10. Computed height maps and error maps for the tests of the PI, FC and UP methods. Blue and orange indicate that the computed height is respectively below or above the true height.

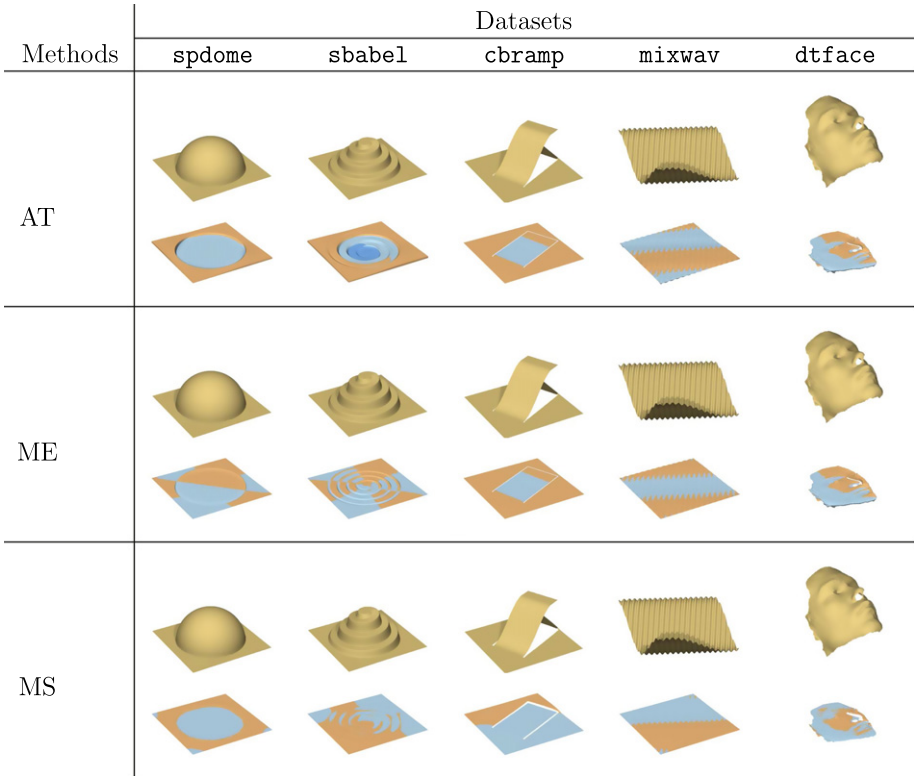

Fig. 11. Computed height maps and error maps for the tests of the AT, ME and MS methods.

Table 4
Relative RMS errors of each method for gradient maps with noise added.

Meth.	spdome e/R (%)	sbabel e/R (%)	cbramp e/R (%)	mixwav e/R (%)	dtface e/R (%)
PI	10.8	19.3	8.9	10.8	19.8
FC	0.9	1.1	120.5	42.9	49.1
UP	1.8	2.4	107.4	1.3	31.8
AT	9.8	15.5	4.5	13.0	4.9
ME	2.0	2.4	1.2	1.3	4.4
MS	1.1	1.5	2.4	1.1	3.7

weighting the difference at each vertex by its total vertex weight (formula (11) or (17)) at scale 0.

Discussion: As shown by Tables 3 and 4, the only methods that obtained usable results for all data sets were Agrawal's Diffusion by Affine Transforms (AT) and M-Estimators (ME) methods, and our multiscale method (MS). The sensitivity of Path-Integration (PI) to noise is evident in Fig. 12. As expected, the unweighted methods FC and UP failed completely on the datasets with cliffs and invalid data. The FC method failed on the mixwave dataset as well, even in absence of discontinuities, due the edge effect explained in Section 2.1.

8.2. Time and memory

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 compared our method against two weighted Poisson integrators provided by Agrawal et al. [34], namely the Diffusion by Affine Transforms method (AT) and the weighted Poisson system builder and solver (PC) that is the innermost loop of their M-Estimator, Energy Minimisation, and α-Surface methods. Those are the only methods in the literature that accept a weight map (thus solving the same problem as ours) and are fast enough for practical use. We removed the outermost loop of these three methods since we are concerned only with the gradient map integration problem, not with the generation of weight map.

Evaluation: All the tests were run on a laptop Dell XPS 1340, with a 2.4 Ghz Intel P8600 Core Duo processor and 4 GB of available memory with 64-bit versions of all software involved. However, the absolute running times are not directly comparable since Agrawal's integrators were implemented in Matlab 2008b under Windows Vista, while our code was compiled with GNU GCC and executed under Mandriva Linux 2010. Therefore, we analysed how the run time and memory usage scale with increasing input map sizes.

We timed only the construction and solution of the linear system, excluding input and output overhead. For the memory figures, we counted only the memory used by the linear system itself and by the working storage used in its solution. In the case of our algorithm, we counted also the time and memory used to prepare the pyramid of gradient and weight maps. The results of these tests are shown in Figs. 14 and 15, and Table 5.

Discussion: The plots in Fig. 14 show that the running times scale quite differently: like $O(N)$ for our algorithm (solid line) and apparently like $O(N^{1.5})$ for the direct Poisson solvers (dashed lines).

Our multiscale integrator also uses less memory than the direct solvers; see Fig. 15. Its memory usage is dominated by the Poisson system's matrix $\mathbf{M}$ which has at most $5N$ nonzero entries and is stored in a specialized sparse data structure that uses $60N$ bytes. The reduced-scale gradient and weight maps use an additional $5N$ bytes in all.

The direct solving methods need to store the matrix $\mathbf{M}$ and also Gauss's triangular factor $\mathbf{U}$ (or Cholesky's $\mathbf{R}$). Note that Gauss's lower factor $\mathbf{L}$ need not be stored. For these methods, we counted the nonzero entries $N_{\mathbf{M}}$ in $\mathbf{M}$ and $N_{\mathbf{U}}$ in $\mathbf{U}$, and estimated the memory usage conservatively as $12N_{\mathbf{M}} + 16N_{\mathbf{U}}$ bytes assuming a general

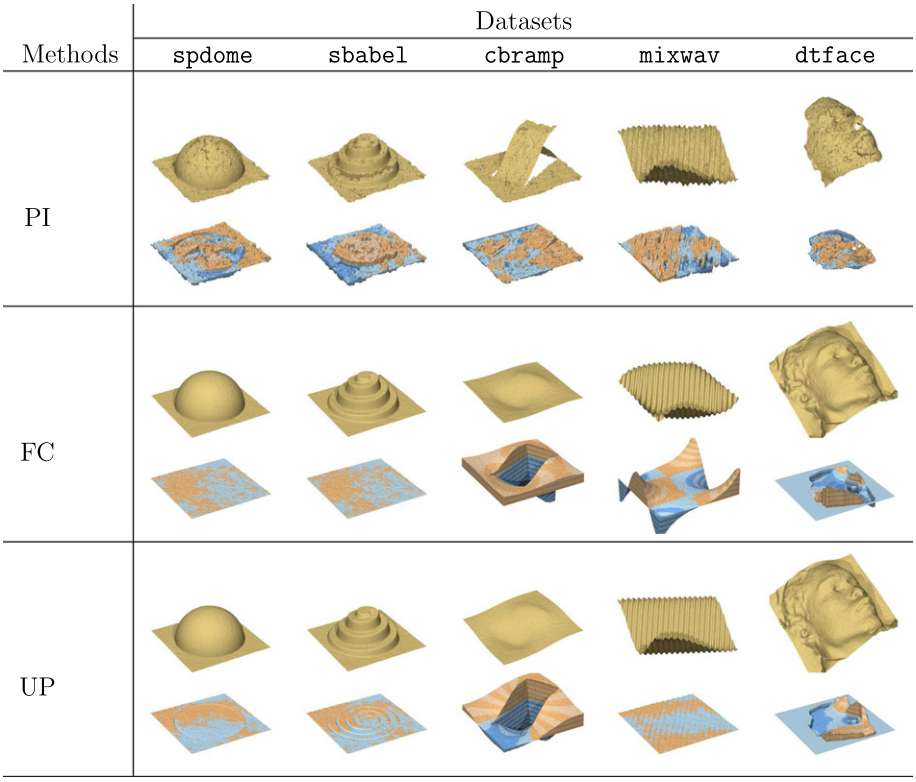

Fig. 12. Computed height maps and error maps obtained from PI, FC and UP methods with 30% of Gaussian noise added to the gradient maps.

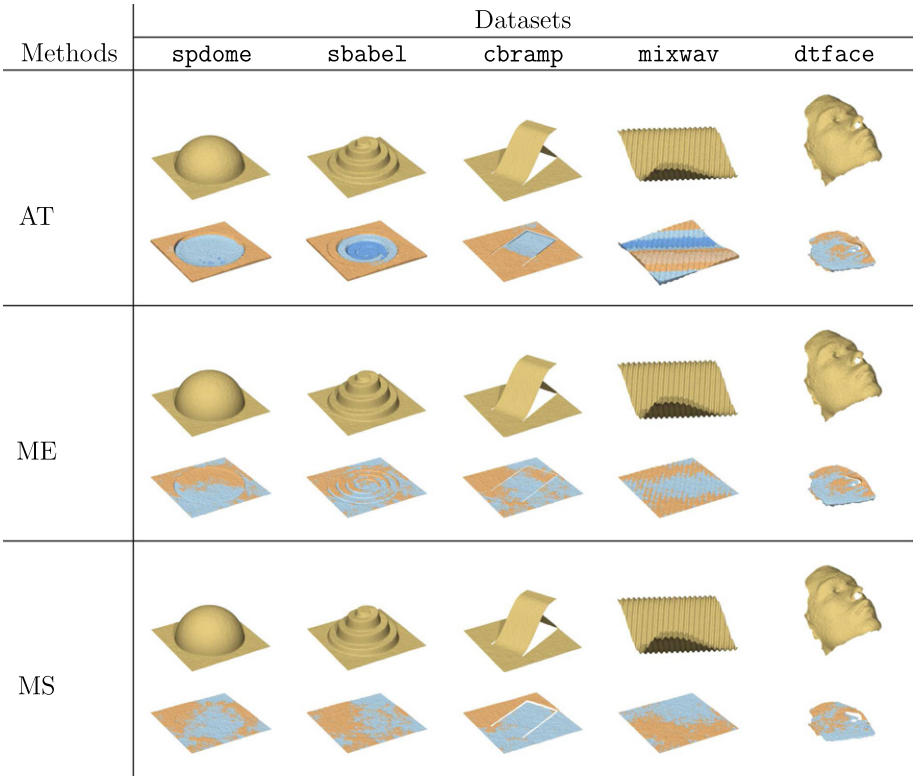

Fig. 13. Computed height maps and error maps obtained from AT, ME and MS methods with 30% of Gaussian noise added to the gradient maps.

sparse matrix representation for **U**. We observed that $N_{\mathbf{M}}$ does not exceed $5N$ for PC and $7N$ for AT, but $N_{\mathbf{U}}$ is much larger and seems to grow like $O(N^{1.15})$ (dotted lines). The actual memory usage of the Matlab implementation is much larger than this estimate: both PC and AT run out of memory, on an unloaded machine with 4 GB of RAM, given a 1024×1024 map.

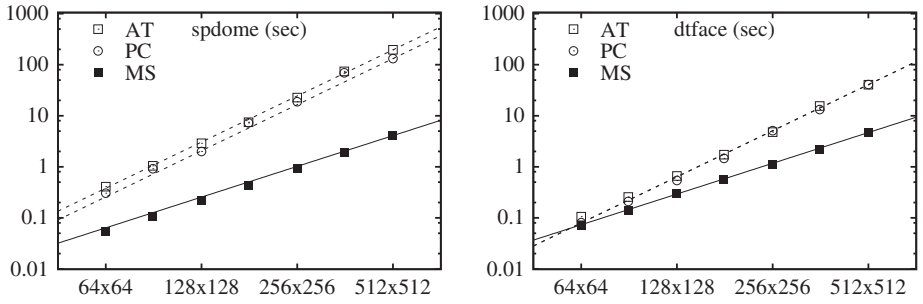

Fig. 14. Log–log plots of the running time of two direct solving methods (PC, AT) and of our multiscale method (MS), in seconds, on two representative datasets and various map resolutions. The straight lines are fitted power laws, $O(N)$ (solid) and $O(N^{1.5})$ (dotted).

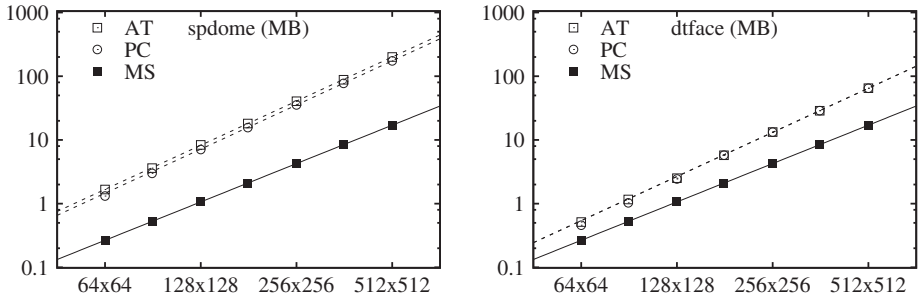

Fig. 15. Log-log plots of memory usage for the system's matrix **M** and its **U** factor (if any), in MBytes. The straight lines are fitted power laws, $O(N)$ (solid) and $O(N^{1.15})$ (dotted).

Table 5
Running times and memory usage of methods MS, PC, and AT on two representative datasets.

N	Time			Memory		
	AT	PC	MS	AT	PC	MS
`spdome`						
$64 \times 64 = 4096$	0.4	0.3	0.1	1.7	1.3	0.3
$90 \times 90 = 8100$	1.1	0.9	0.1	3.6	3.0	0.5
$128 \times 128 = 16384$	2.9	2.0	0.2	8.4	7.1	1.1
$180 \times 180 = 32400$	7.5	7.3	0.4	18.1	15.6	2.1
$256 \times 256 = 65536$	22.7	18.8	0.9	40.5	35.5	4.3
$360 \times 360 = 129600$	74.0	69.2	1.9	88.1	77.2	8.4
$512 \times 512 = 262144$	195.4	131.8	4.1	200.2	174.1	17.0
`dtface`						
$64 \times 64 = 4096$	0.1	0.1	0.1	0.5	0.5	0.3
$90 \times 90 = 8100$	0.3	0.2	0.1	1.2	1.0	0.5
$128 \times 128 = 16384$	0.7	0.5	0.3	2.5	2.4	1.1
$180 \times 180 = 32400$	1.7	1.5	0.6	5.7	5.8	2.1
$256 \times 256 = 65536$	4.8	5.1	1.1	13.3	13.3	4.3
$360 \times 360 = 129600$	15.5	13.2	2.2	28.7	28.8	8.4
$512 \times 512 = 262144$	40.6	40.8	4.7	64.8	64.7	17.0

Note that time and memory costs are much lower for the `dtface` dataset than `spdome`. This is because the three algoritms exclude from the Poisson system the height values which have no gradient data ($w[u, v] = 0$) and set them to zero directly.

9. Conclusions

We have described in this paper an algorithm that integrates a gradient map to yield a height (depth) map. It accepts reliability weights for individual gradient samples, thus allowing reliable reconstruction of discontinuous height fields. Thanks to the use of multiscale integration, our method is as accurate as competing algorithms but considerabily faster in most inputs. The exceptions are gradient maps that have narrow corridors or isthmuses bounded by cliffs or missing data (such as the example in Fig. 8).

Our algorithm could be used also as the inner loop of nonlinear iterative outlier detection methods such as described by Agrawal *et al.* [11]. It also can be easily parallelized for SIMD platforms such as GPU and FPU for real-time processing.

Acknowledgments

We thank Antonio Robles-Kelly, Jean-Denis Durou, Jean-François Aujol, and Amit Agrawal, who kindly provided the code of their integrators for our tests. We thank also the support given from CnPQ, CAPES, FAPESP and EPSRC funding agencies for this work.

References

[1] B.K.P. Horn, M.J. Brooks, Shape from Shading, MIT Press, Cambridge, Mass, 1989.
[2] B.K.P. Horn, Height and gradient from shading, Intl. J. Computer Vis. 5 (1) (1990) 37–75.
[3] B.K.P. Horn, R.J. Woodham, W.M. Silver, Determining shape and reflectance using multiple images, Tech. Rep. AI Memo 490, MIT Artificial Intelligence Laboratory, 1978.
[4] R.J. Woodham, Photometric method for determining surface orientation from multiple images, Opt. Eng. 19 (1) (1980) 139–144.
[5] M.L. Smith, L.N. Smith, Polished Stone Surface Inspection using Machine Vision, OSNET, 2004, p. 33.
[6] M. Kampel, R. Sablatnig, 3D puzzling of archeological fragments, in: D. Skocaj (Ed.), Proc. of 9th Computer Vision Winter Workshop, 2004, pp. 31–40.
[7] J. Sun, M.L. Smith, A.R. Farooq, L.N. Smith, Concealed object perception and recognition using a photometric stereo strategy, in: ACIVS'09, vol. 5807, pp. 445–455. doi:10.1007/978-3-642-04697-1.
[8] M.F. Hansen, G.A. Atkinson, L.N. Smith, M.L. Smith, 3d face reconstructions from photometric stereo using near infrared and visible light, Computer Vis. Image Understand. (in press), doi:10.1016/j.cviu.2010.03.001.
[9] R.T. Frankot, R. Chellappa, A method for enforcing integrability in shape from shading algorithms, IEEE Trans. Pattern Anal. Mach. Intell. 10 (4) (1988) 439–451.
[10] A. Agrawal, R. Chellappa, R. Raskar, An algebraic approach to surface reconstruction from gradient fields, in: ICCV 2005, pp. 174–181.
[11] A. Agrawal, R. Raskar, R. Chellappa, What is the range of surface reconstructions from a gradient field? in: ECCV 2006, vol. 3951, pp. 578–591, doi:10.1007/11744023.
[12] M. Chandraker, S. Agarwal, D. Kriegman, ShadowCuts: Photometric stereo with shadows, in: CVPR07, 2007, pp. 1–8.

[13] G.A. Atkinson, E.R. Hancock, Surface reconstruction using polarization and photometric stereo, in: CAIP 2007, vol. 4673, pp. 466–473.
[14] H.C.G. Leitão, R.F.V. Saracchini, J. Stolfi, Matching photometric observation vectors with shadows and variable albedo, in: SIBGRAPI'08, pp. 179–186.
[15] D. Reddy, A. Agrawal, R. Chellappa, Enforcing integrability by error correction using ℓ_1-minimization, in: CVPR' 09, pp. 2350–2357.
[16] R. Klette, K. Schlüns, Height data from gradient fields, in: Proc. 1996 SPIE Conf. on Machine Vision Applications, Architectures, and Systems Integration, 1996, pp. 204–215.
[17] Z. Wu, L. Li, A line-integration based method for depth recovery from surface normals, Computer Vis. Graph. Image Process. 43 (1) (1988) 53–66.
[18] R. Fraile, E.R. Hancock, Combinatorial surface integration, in: Proc. 18th Intl. Conf. on Pattern Recognition (ICPR'06), vol. 1, 2006, pp. 59–62, doi:10.1109/ICPR.2006.379.
[19] A. Robles-Kelly, E.R. Hancock, Surface height recovery from surface normals using manifold embedding, in: Proc. Intl. Conf. on Image Processing (ICIP), 2004.
[20] T. Wei, R. Klette, Height from gradient using surface curvature and area constraints, in: Proc. 3rd Indian Conf. on Computer Vision, Graphics and Image Processing, 2002.
[21] G.D.J. Smith, A.G. Bors, Height estimation from vector fields of surface normals, in: Proc. IEEE Intl. Conf. on Digital Signal Processing (DSP), 2002, pp. 1031–1034.
[22] A.S. Georghiades, P.N. Belhumeur, D.J. Kriegman, From few to many: illumination cone models for face recognition under variable lighting and pose, TPAMI 23 (2001) 643–660.
[23] B.K.P.Horn, Height and gradient from shading, Tech. Rep. AI Memo 1105, Massachusetts Institute of Technology, 1989.
[24] D. Terzopoulos, Image analysis using multigrid relaxation methods, TPAMI (2) (1986) 129–139.
[25] D. Terzopoulos, The computation of visible-surface representations, TPAMI 10 (4) (1988) 417–438.
[26] H.-S. Ng, T.-P. Wu, C.-K. Tang, Surface-from-gradients without discrete integrability enforcement: a Gaussian kernel approach, TPAMI 32 (11) (2010) 2085–2099, http://dx.doi.org/10.1109/TPAMI.2009.183.
[27] Y. Chen, H. Wang, Y. Wang, Pyramid surface reconstruction from normal, in: ICIG 2004, pp. 464–467.
[28] D.J. Durou, F.J. Aujol, F. Courteille, Integrating the normal field of a surface in the presence of discontinuities, in: EMMCVPR '09, pp. 261–273.
[29] T.S.F. Haines, R.C. Wilson, Integrating stereo with shape-from-shading derived orientation information, in: BMVC 2007.
[30] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, McGraw-Hill, 1990.
[31] 3dMD, 3dMDFace system, 2010, Electronic document at <http://www.3dmd.com/3dmdface.html, edit date 2010/10/26> (accessed 26.10.2010).
[32] C. Cason, POV-Team, Persistence of Vision Raytracer, 2008, Electronic document at <http://www.povray.org/, edit date 2008/07/30> (accessed 25.10.08).
[33] P.D. Kovesi, frankotchellappa.m: The Frankot–Chellappa gradient integrator in MATLAB/Octave, Univ. of Western Australia, 2000 <http://www.csse.uwa.edu.au/pk/Research/MatlabFns/Shapelet/>.
[34] A. Agrawal, Matlab/Octave code for robust surface reconstruction from 2d gradient fields, 2006 <http://www.umiacs.umd.edu/aagrawal/software.html> (accessed 01.05.10) [11].
[35] H.C.G. Leitão, R.F.V. Saracchini, J. Stolfi, Depth from slope by weighted multi-scale integration, Tech. Rep. IC-10-09, Institute of Computing, State University of Campinas, 2010.
[36] A. Robles-Kelly, R. Fraile, E.R. Hancock, Surface Integration in Matlab, BitBucket.org Free source code hosting, 2010 <http://bitbucket.org/jaxze/surface-integration-in-matlab/overview>.