A robust multi-scale method for two-dimensional gradient integration using irregular difference meshes

Rafael Felipe Veiga Saracchini[a], Jorge Stolfi[b], Helena Cristina G. Leitão[c]

[a] *Technological Institute of Castilla y León, Burgos, Spain*
[b] *State University of Campinas, Campinas, Brazil*
[c] *Federal Fluminense University, Niterói, Brazil*

Abstract

We describe a particular variant of the algebraic multi-grid method for gradient integration on meshes with arbitrary topology and geometry, on the plane or other surface of low genus. This problem is an essential task in photometric stereo, and can be seen as a least-squares approach to solving more general Poisson-type problems.

Unlike some geometric multi-grid methods, our algorithm – which we call two-dimensional topological multi-grid (`TMG2`) – does not assume that the mesh nodes have specific positions. Our novel mesh coarsening algorithm runs in linear time and uses only the topology of the mesh, not the node positions and distances. It produces a two-dimensional mesh with the same underlying manifold topology but a guaranteed fractional reduction of the number of variables and equations. The reduced mesh remains connected even if the original one had narrow bridges.

We also describe robust formulas for converting gradient data sampled in a regular grid, with local uncertainties and missing samples, into a topological mesh that can be the input of our integration method.

We show that our algorithm outperforms other surface gradient integrator algorithms on typical data sets, especially on instances with poorly connected meshes that cause previous multi-scale methods to fail.

Keywords: Image processing, Algebraic multi-grid, Surface gradient integration, Photometric stereo

Email addresses: `saracchini@gmail.com` (Rafael Felipe Veiga Saracchini), `stolfi@ic.unicamp.br` (Jorge Stolfi), `hcgl@ic.uff.br` (Helena Cristina G. Leitão)

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]. In these applications, one usually obtains the mean normal vector of the object's surface that is visible within each image pixel; which can be converted to the height gradient, that is, the partial derivatives (*slopes*) of the surface's height Z with respect to the image coordinates X and Y.

Although the gradient 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 [5], pottery fragment reassembly [6], surveillance and customs inspections [7], face recognition [8], and many others.

Abstractly, the *gradient integration* problem consists in the determination of a unknown real-valued function $Z(x, y)$ defined in a domain $D \in \mathbb{R}^2$, given its gradient $\nabla Z = (\partial Z/\partial x, \partial Z/\partial y)$ in that region. That is, we wish to compute Z such that

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

for each point (x, y) within D, where F and G are two known functions defined in D. 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 each $(x, y) \in D$. The left side of equation (2) is the curl (rotational) of the vector field (F, G), thus the equation is called the *zero curl condition*.

If condition (2) is satisfied, the solution Z can be obtained in various ways. For a rectangular domain D with lower corner placed at $(0, 0)$, for example, it can be obtained by the formula

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

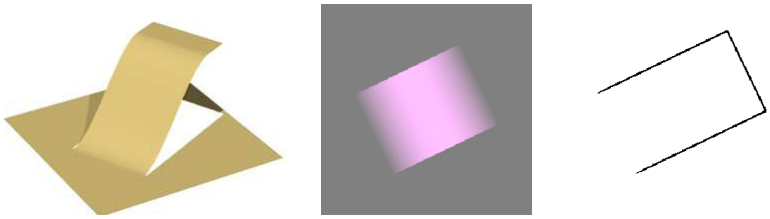
where C is an arbitrary constant. Note that the degree of freedom represented by C is an inherent characteristic of the original problem, not a limitation of the method.

1.1. Computational difficulties

In practical contexts, the computation of heights from given slopes runs into three major difficulties. First, the gradient data F, G 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*. Therefore, the height function cannot be precisely determined. It can only be approximated by a member of some finite-dimensional space of approximating functions. The approximating function can be (and usually is) uniquely represented by a finite set of discrete *height samples*, each being the estimated average of the height Z over some neighborhood of a *height sampling point*. Note that the height sampling points may not coincide with the gradient sampling points.

Second, the gradient data is usually contaminated with *noise* arising from unavoidable measurement, quantization, and computation errors. In some parts of the domain D, the expected magnitude of the error may be so high that the gradient is essentially unknown. In the case of photometric stereo and shape-from-shading, for example, it is usually impossible to determine the gradient where the scene's surface is affected by shadows or specular highlights, is too dark, or is poorly illuminated. Gaps (or large errors) in the data will also arise wherever the actual height or gradient functions are inherently indeterminate, e.g. where the scene is highly porous, covered with hair-like structures, or transparent.

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



(a) (b) (c)

Figure 1: A height map with cliff-like discontinuities (a), its color-coded gradient map (b), as could be obtained by photometric stereo methods, and a binary image (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.

Some gradient integration algorithms try to detect data gaps and height discontinuities by analyzing the given gradient data. Usually, the violation of the zero-curl condition (2) is taken as an indication that the gradient data is unreliable at that point. However, as shown by the the example of figure 1, the gradient data simply does not contain the necessary information to reliably detect cliffs and bad data. Therefore, in this paper we assume that these flaws are identified by a separate algorithm (possibly using other data capture techniques), and they are given to the integrator separately from the gradient data, as a *weight* image, as shown in figure 1(c). See section 4.2 for the definition of that image.

1.2. The region disconnection problem

The presence of cliffs and missing data creates a major problem for methods that try to use the multi-scale approach to speed up the integration [9, 10]: the possible *disconnection* of parts of the domain as the data is reduced to coarser and coarser scales [11]. When a rectangular grid of data is sub-sampled, any gaps or indeterminacies will persist in the smaller grid, and will occupy a proportionately larger area in it. Eventually these growing data gaps may completely separate parts of the domain, rendering the multi-scale integration ineffective.

Specifically, a band of well-defined samples that is t grid steps wide and surrounded by data gaps will disappear after $k \approx \log_2(t)$ coarsening steps. See figure 2.

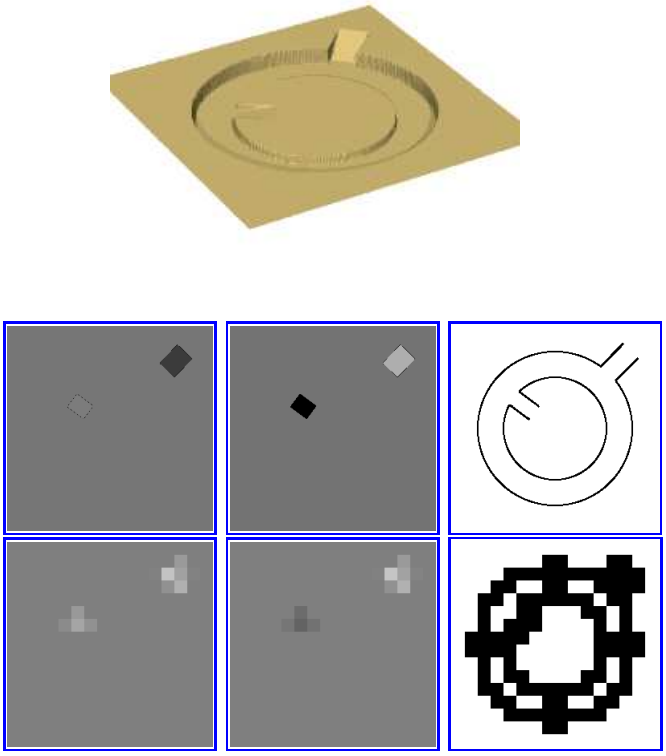


Figure 2: Example of region disconnection in grid-based multi-scale integration. Top: perspective view of a height map $Z = Z(X, Y)$ with discontinuities. Middle: derivatives $F^{(0)} = \partial Z/\partial X$ and $G^{(0)} = \partial Z/\partial Y$ sampled on a 256×256 grid, and 256×256 binary mask $W^{(0)}$ showing the location of the height discontinuities in the domain (in black). Bottom: the maps $F^{(4)}$, $G^{(4)}$, and $W^{(4)}$ resulting from 4 steps of filtering and sub-sampling. Note that the central disk has become disconnected from the surrounding areas.

When two regions R and S become disconnected at some scale k, the height of R relative to S cannot be determined from the reduced data, not even approximately. Therefore, the height map $Z^{(k)}$ computed at that scale will not be a good starting guess for the iteration at the next finer scale $k - 1$, and it may take many iterations at that scale for the regions R and S to be displaced to the proper relative height. In fact, while the relative height of the two regions is being readjusted, the information computed inside them at the coarser scales is lost – and has to be recomputed, much more slowly,

at scale $k - 1$.

1.3. Contributions

The main contribution of this article is a flexible integration method for discretized gradient data, suitable for photometric stereo and other applications, that can cope with non-uniform errors and gaps in the input gradient data, as well as discontinuities in the height field. We will denote it by the acronym `TMG2`, short for *two-dimensional topological multi-grid*.

The general approach of the new method is similar to the multi-scale integrator described previously by Saracchini *et al.* [11], but using an irregular and purely topological mesh in place of the rectangular grid of gradient samples used by most methods, including that one. See section 3. It is a special case of the *algebraic multi-grid* (AMG) approach, that preserves the connectivity even in poorly connected cases. The method is designed for gradient integration problem on the plane (or two-dimensional manifolds of low genus), but it can be used to solve other Poisson-type (second-order) integration and optimization problems as well.

The irregular mesh representation allows `TMG2` to overcome the disconnection problem of previous multi-scale methods, described in section 1.2. By representing the given gradient data as an arbitrary planar mesh. instead of a regular grid, we can preserve the connectivity of the gradient data while reducing the spatial resolution. Moreover, the mesh allows the integration of data that is irregularly spaced by nature, such as gradient maps that have been subjected to optical rectification, perspective correction and mosaic composition.

Another contribution of this article is a robust method for the conversion of gradient data sampled in a regular rectangular grid to the weighted differences mesh representation that is the input to our integrator. See section 5

These two methods yield a gradient integrator algorithm that generally outperforms other integrators described in the literature, in running time, robustness against data errors and gaps, and accuracy of the computed height maps.

2. Related Work

A brief review and classification of gradient integration methods for photometric stereo was provided by Saracchini *et al.* [11]. To summarize, most algorithms for this problem use one of four main techniques: *path integration* [12, 13, 14, 15]; integration via *Fourier transform* [16, 17]; *direct*

solving of a system of linear equations that discretize the Poisson equation [2, 18, 19, 20, 21], by Gaussian LU or Cholesky factorization; and *local iteration* [2, 11, 22, 23] by Gauss-Seidel or similar methods.

Another classification was provided by Durou *et al.* [24, 25], according to six properties: (1) *computational speed*; (2) *robustness* in face of Gaussian noise; (3) allowance for *free boundary* conditions; (4) handling of *discontinuities* such as caused by occlusions; (5) ability to process data with *arbitrary domains*, not just rectangular; and (6) need for *fine tuning* of the algorithm parameters for each instance of the problem. He found only one method that meets criteria (4) and (6), namely the path-based integrator of Fraile and Hancock [14]; which however fails at criterion (2), since it is extremely sensitive to measurement noise.

Local iteration methods seem to be the most adequate for photometric stereo, because they are able to use Poisson or Krylov equations, even with Laplacian estimators that take into account missing or unreliable gradient data at each sampling point, satisfying most of the aforementioned criteria. Additionally, these methods can deal with moderately non-linear equations in the same iterative loop, without explicit linearization (as in the Newton-Raphson method). They also demand less space: their memory consumption grows proportionally to N whereas direct solving methods seem to require space proportional to $N^{1.15}$ even with good sparse system solvers [11].

The main disadvantage of local iteration methods is the potentially slow rate of convergence [11]. Each iteration requires only $O(N)$ operations, but the number of iterations needed to reduce the initial error E bellow a tolerance ε is on the order of $N \log(E/\varepsilon)$, which implies a total processing time of $N^2 \log(E/\varepsilon)$.

This convergence can be accelerated by the *multi-scale* approach, as suggested by Terzopoulos [9, 10]. In this approach, the original gradient data F, G is properly sub-sampled to give F', G' a lower resolution scale. This data is integrated recursively to yield a height field Z' at the same resolution. This coarse solution is then interpolated to give the initial guess for the iterative computation of the desired solution Z.

The multi-scale iterative method developed by Saracchini *et al.* [11] is competitive in speed with Fourier based integration, while still being able to cope with missing data, non-uniform error, and height discontinuities. It expects to receive, as part of the input data, a *weight* image that specifies the regions of the domain where the gradient is unreliable (such as along the discontinuities). However, the multi-scale approach fails to retrieve proper height map estimates within an acceptable number of iterations if the domain becomes disconnected at the coarsest scales of the multi-scale pyramid.

In 2015, Quéau *et al.* [23] described an integrator based on the energy minimization principle. They considered three minimization strategies inspired by image denoising techniques: weighted least-squares, total variation, and L^1 optimization. Like many other integrators, his algorithm tries automatically detect data gaps and cliffs by looking for inconsistencies in the gradient data.

In 2016, Breuß *et al.* [26] proposed the usage of a fast-marching method in order to provide a first approximation for iterative Poisson-based and Krylov-based solvers. Their FM integrator is very fast and with very little memory overhead compared with direct solvers. However, it is reportedly less accurate than the other methods.

3. Weighted Differences Mesh

Our main contribution is the representation of the integration problem as a *weighted differences mesh*, which consists in a directed graph G with vertices $\mathcal{V}\,G$ and directed edges $\mathcal{E}\,G$, where each vertex v is associated to a unknown *height* value $z[v]$, and each edge e has two numerical attributes: the *difference* $d[e]$ and the *weight* $w[e]$. See figure 3. By definition, for each directed edge e in the weighted mesh, the reverse edge $\textsc{sym}(e)$ is also present with $d[\textsc{sym}(e)] = -d[e]$ e $w[\textsc{sym}(e)] = w[e]$. However when drawing the mesh, only one of those edges is shown.

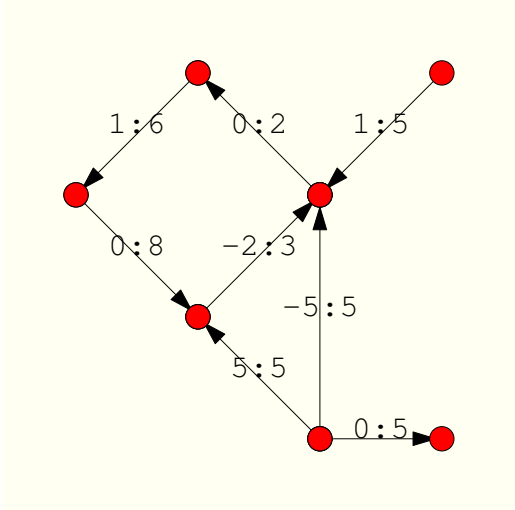


Figure 3: A small weighted differences mesh. The edge labels are the pairs $d[e] : w[e]$. Note that only one of the edges $e, \textsc{sym}(e)$ is shown.

The attribute $d[e]$ is to be interpreted as an estimate of the difference $z[v] - z[u]$ between the height values of the destination vertex $v = \text{DST}(e)$ and the source vertex $u = \text{ORG}(e)$. In our algorithm, this estimated is obtained by interpolation, extrapolation, and scale reduction of the given gradient data. Thus, the differences mesh can be seen as a position independent discretization and abstraction of the partial differential equations (1).

The weight $w[e]$ of an edge e is a non-negative real number which expresses the reliability of the difference $d[e]$. More specifically, we assume that the edge difference $d[e]$ includes a Gaussian measurement error, with expected value zero and variance proportional to $1/w[e]$. In particular, $w[e] = 0$ signifies that the difference $d[e]$ is totally random and contains no useful information. Such edges can be removed from the mesh without effect on the computed solution.

We say that a mesh is *simple* if it does not have *parallel edges*, that is, two or more edges with same source and destination. In a simple mesh, we can identify each edge e with the ordered pair (u, v) of its source and destination vertices. In this case, we can denote $d[e]$ also as $d[u, v]$ and $w[e]$ by $w[u, v]$.

3.1. Edge equations

A weighted mesh G can be interpreted as a system of *edge equations*, the linear equations

$$z[\text{DST}(e)] - z[\text{ORG}(e)] = d[e] \tag{4}$$

for each directed edge e. The problem of *mesh integration* is to compute the most probable value $z[v]$ for each vertex v, given the parameters $d[e], w[e]$ of each edge e.

It is evident that each connected component of G can be treated as a separate instance of this problem. Thus we will assume that the graph G is always connected. Moreover, since the equations (4) depend only on the value differences, the solution for a connected mesh will have one and only one degree of freedom: an additive constant C.

A set of height values z is said to be *tension free* if all equations (4) are satisfied exactly. This is the case if and only if the sum of differences of edges along any directed cycle is zero. This property is analogous to the the zero curl condition for the continuous integration problem of section 1. In particular, if the mesh has only one simple path between two vertices (that is, a tree), it always has a tension free solution.

3.1.1. Path integration on a mesh

If a weighted differences mesh admits a solution $z[v]$ free of tension, that solution can be computed by choosing an arbitrary spanning tree T for G, associating an arbitrary equation for a given vertex v_0 and using equation (4) to compute the heights of other vertices in order of increasing graph distance from v_0 along T. Note that the weights of the edges are irrelevant in this case. This algorithm is the irregular-mesh version of the path integration formula (3).

3.2. Vertex equilibrium equation

If a weighed differences mesh G has cycles, however, the system of equations (4) is over-determined. In this case, the existence of a tension free solution is unlikely. If we apply the path integration algorithm to this case, the computed height $z[u]$ will depends on the choice of the path from v_0 to u.

If the system (4) is over-determined, the assumption of independent Gaussian measurement errors in the edges and Bayesian analysis imply that the most probable set of values z is the solution by least squares of that system; that is, the values z that minimize the *quadratic discrepancy function*

$$Q(z) = \sum_{e \in \mathcal{E}\, G} w[e]\, (z[\textsc{dst}(e)] - z[\textsc{org}(e)] - d[e])^2 \tag{5}$$

Suppose that the mesh G is simple and let $G[u]$ be the set of vertices adjacent to the vertex $u \in G$. The function Q is minimized when each vertex u is in *equilibrium*; that is, if only and only if we have

$$\sum_{v \in G[u]} w[u,v]\, (z[v] - z[u] - d[u,v]) = 0 \tag{6}$$

We can write the equation (6) as

$$z[u] = \frac{1}{w_{\text{tot}}[u]} \sum_{v \in G[u]} w[u,v](z[v] - d[u,v]) \tag{7}$$

where

$$w_{\text{tot}}[u] = \sum_{v \in G[u]} w[u,v] \tag{8}$$

In other words, equilibrium occurs when $z[u]$ is the weighted average of $z[v] - d[u,v]$ of all neighbors v of u, where each of these terms is weighted

by $w[u,v]$. The solution z is tension-free if all the terms $z[v] - d[u,v]$ have the same value. Equation (7) can also be written as

$$z[u] - \sum_{v \in G[u]} \lambda[u,v] z[v] = \sum_{v \in G[u]} \lambda[u,v] d[u,v] \tag{9}$$

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

The vertex equilibrium equation (9) can be seen as an abstraction of the Poisson formulation of the integration problem [11]. Namely, the left-hand side of (9) can be seen as an estimate of the Laplacian of Z, obtained from the (unknown) height values by a second-order finite difference formula; whereas the right-hand side would be another (known) estimate of the Laplacian, obtained by differentiating the gradient data once. Equating these two estimates gives a linear system that has a unique solution (apart from a constant).

3.3. Physical analogies

The following mechanical analogy may help understand the mesh integration problem. Each vertex v is modeled as a mass-less horizontal plate, which is free to move vertically, but cannot move horizontally or rotate. The value $z[v]$ is the vertical position of said plate. Each edge $e = (u,v)$ is modeled as an ideal vertical spring with rigidity coefficient $w[e]$, connected to the plates u and v so as to apply to the bottom plate a vertical force with value $w[e](z[v] - z[u] - d[e])$, and an equal but opposite force to the top plate. That is, the spring tries to pull or push the plates apart, trying to make the distance $z[v] - z[u]$ to be equal to $d[e]$.

A set of values $z[v]$ which minimize $Q(z)$ is a situation of mechanical equilibrium, in which the total force acting on each plate is zero. Note that the potential energy of the springs is $\frac{1}{2}Q(z)$, and the system will be in equilibrium (no forces actuating in each vertex) when its potential energy is minimum. In particular, the solution z is free of tension if $Q(z)$ is zero, that is, if each spring length is equal to its relaxed length.

Another analogy for this mathematical problem is an electrical circuit where each vertex v is a conductive node, the variable $z[v]$ the electrical potential of the node (in volts) and each edge (u,v) a battery with driving voltage $d[u,v]$ and internal resistance $1/w[u,v]$ (in ohms) connected to the nodes u and v. Thus, the current (u,v) (in amperes) that arrives in u by the edge $[u,v]$ is $-w[u,v](z[v] - z[u] - d[u,v])$. The functional $Q(z)$ is the electrical power dissipated by the circuit. Then equation (6) is Kirchoff's

law, which is satisfied when the circuit is electrical equilibrium and the net total current entering and exiting each node is zero.

3.4. Matrix formulation

In order to express the problem in matrix form, let $v_1, v_2, \ldots, v_n$, the vertices of G, in arbitrary order, and $e_1, e_2, \ldots, e_m$ a list of directed edges from G, also in arbitrary order, which has only one directed version of each pair e and $\text{SYM}(e)$. Let also

- $\mathbf{A}$ be the *incidence matrix* of G, that is, an $m \times n$ matrix such that $\mathbf{A}_{kj}$ is $+1$ if the vertex $v_j = \text{DST}(e_k)$, -1 if $v_j = \text{ORG}(e_k)$, and zero otherwise;
- $\mathbf{W}$ be a diagonal $m \times m$ matrix such that $\mathbf{W}_{jj} = w[e_j]$;
- $\mathbf{d}$ be a column vector of m elements such that $\mathbf{d}_j = d[e_j]$;
- $\mathbf{z}$ be a column vector of n elements such that $\mathbf{z}_k = z[v_k]$;

for each $i, j \in \{1, \ldots, m\}$ and each $k \in \{1, \ldots, n\}$. Then the functional Q can be expressed as a matrix product:

$$Q(z) \;=\; (\mathbf{Az} - \mathbf{d})^\top \mathbf{W} (\mathbf{Az} - \mathbf{d}) \tag{10}$$

where $\mathbf{M}^\top$ denotes the transpose of the matrix $\mathbf{M}$. The vector $\mathbf{z}$ that minimizes Q can be computed by differentiating this formula with respect to each $\mathbf{z}_k$ and equating it to zero. In matrix form, the equations are

$$\mathbf{Mz} = \mathbf{b} \tag{11}$$

where

$$\mathbf{M} \;=\; \mathbf{A}^\top \mathbf{WA} \tag{12}$$

$$\mathbf{b} \;=\; \mathbf{AWd} \tag{13}$$

The solution of the linear system (11) can be then computed by the Gauss-Seidel or direct solving methods.

4. Converting a gradient image to a difference mesh

In this section we describe the conversion of gradient data from the image format (arrays of discrete gradient samples, taken on an uniform rectangular grid of points) into a weighted differences mesh G.

4.1. Discretizing F and G

We assume that the gradient data is given as two *derivative maps*, that is, two matrices $f[u,v]$ and $g[u,v]$, with n_x columns and n_y rows, where the indices u, v range in $\{0,1,\ldots,n_x-1\}$ and $\{0,1,\ldots,n_y-1\}$, respectively. By definition, the domain D of the problem is the rectangle $[0,n_x]\times[0,n_y] \in \mathbb{R}^2$. Each element $[u,v]$ from the derivative maps can be identified with the unit side square centered on the *gradient sampling point* $p[u,v] = (u + 1/2, v + 1/2)$, with (u,v) and $(u+1,v+1)$ in opposite corners. See figure 4.

4.2. The weight map

We assume that, together with the gradient data, we are also given a *weight* image: an array w of non-negative values, with the same dimensions as f and g. Our algorithms assume that the samples $f[u,v]$ and $g[u,v]$ at each sampling point $p[u,v]$ are contaminated by additive Gaussian noise with variance proportional to $1/w[u,v]$.

In particular, if $w[u,v]$ is zero, the gradient is assumed to be completely indeterminate at $p[u,v]$. We assume that $w[u,v]$ is zero, and therefore $f[u,v]$ and $g[u,v]$ are indeterminate, when the point $p[u,v]$ is outside the domain D. Note that only the *relative* weight magnitudes are significant, that is, the results wont be affected if we multiply all the weights by a positive scale factor.

4.3. Discretizing Z

The function Z is represented by a *height map*, a matrix $z[u,v]$ with n_x+1 columns and n_y+1 rows that represents the estimated height Z at the *height sampling point* $q[u,v] = (u,v)$. See figure 4. Thus, note that the gradient sample points $p[u,v]$ are assumed to be shifted by one half pixel, along each axis, relative to the height sampling points $q[u,v]$.

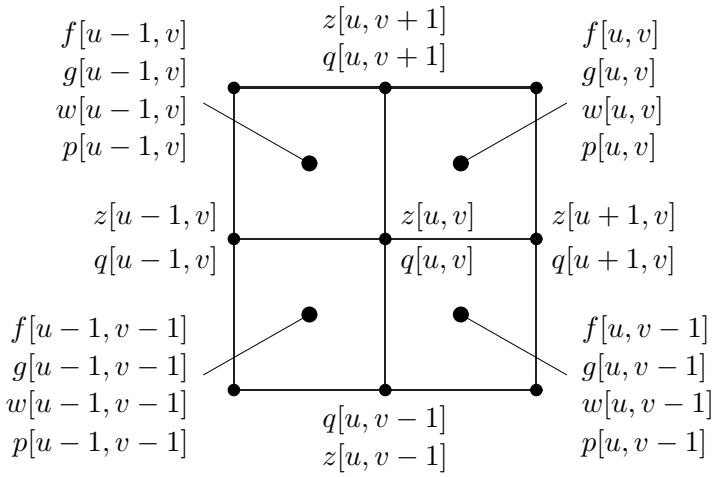


Figure 4: Gradient sampling points $p[x, y]$ and height sampling points $q[x, y]$ around the point $q[u, v] = (u, v)$.

The *grid edges* are the line segments that connect two height sampling points that are adjacent horizontally or vertically; that is, from each $q[u, v]$ to either $q[u+1, v]$ or $q[u, v+1]$, provided that both endpoints are inside D or on its border. The edges of the difference mesh G will correspond to a subset of the grid edges, each oriented in both ways, with appropriate difference and weight attributes.

In practice, the derivative maps $f[u, v]$ and $g[u, v]$ are almost always an average of the continuous derivatives $F = \partial Z/\partial x$ and $\partial Z/\partial y$ in the neighborhood of the point $p[u, v]$, obtained by convolution with a *gradient sampling kernel*. This kernel should be symmetric relative to $p[u, v]$ and should overlap partially the neighboring kernels. Similarly, the computed height $z[u, v]$ will be an estimate of the average Z around the point $q[u, v]$, obtained by some *height sampling kernel*. The relationship between those two kernels is outside of the scope of this paper.

4.4. Interpolated edge gradients

In order to improve the legibility of the following formulas, when u and v are fixed by the context, we will use the notation $z_{\circ\circ}$ for the desired height sample $z[u, v]$, and the following notations for the adjacent samples:

$$\begin{aligned} z_{-\circ} &= z[u-1, v] \qquad & z_{\circ-} &= z[u, v-1] \\ z_{+\circ} &= z[u+1, v] \qquad & z_{\circ+} &= z[u, v+1] \end{aligned} \tag{14}$$

In order to build the mesh, our algorithm first estimates the derivative F or G at the midpoint of the grid edge between $q_{\circ\circ} = q[u, v]$ and each of its four neighbors. We denote those values as follows:

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

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

Our algorithm also assigns weights to those derivative estimates, which we will denote $w_{-\circ}$, $w_{+\circ}$, $w_{\circ-}$, and $w_{\circ+}$. See figure 5. The computation of these values and weights is described in section 4.6.

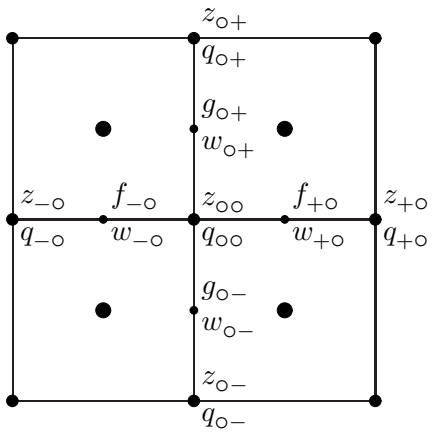


Figure 5: Notation for the interpolated gradient values f and g, the respective weights w, and the desired height values z around the point $q_{\circ\circ} = q[u, v]$.

4.5. The mesh edge equations

We can relate the desired height values to those interpolated derivatives, by equating the difference of the two values with the appropriate derivative at the midpoint of the grid edge connecting them. Namely, for each height sampling point $q_{\circ\circ} = q[u, v]$, we have the following equations, with the respective weights in parentheses:

$$\begin{aligned} z_{+\circ} - z_{\circ\circ} &= +f_{+\circ} \quad (w_{+\circ}) & \qquad z_{\circ+} - z_{\circ\circ} &= +g_{\circ+} \quad (w_{\circ+}) \\ z_{-\circ} - z_{\circ\circ} &= -f_{-\circ} \quad (w_{-\circ}) & \qquad z_{\circ-} - z_{\circ\circ} &= -g_{\circ-} \quad (w_{\circ-}) \end{aligned} \tag{15}$$

Each equation in this set that has a positive weight is represented in the mesh G by a directed edge e from vertex $z_{\circ\circ}$ to the corresponding adjacent vertex. The difference attribute $d[e]$ of e is the interpolated derivative on the right-hand side of the equation, and the weight attribute $w[e]$ is the weight associated to that interpolated value.

Note that, by applying these rules to every height sampling point $q[u, v]$, every undirected edge of the grid with positive interpolated weight will give rise to two oppositely directed edges e', e'' of the mesh, with $d[e'] = -d[e'']$ and $w[e'] = w[e'']$.

As detailed in section 4.6, if $q_{\circ\circ}$ lies along the border of the domain rectangle D, any edge e to a neighbor that lies outside D will get zero weight $w[e]$, and thus will be omitted from the mesh G.

4.6. Interpolating the derivatives

In this section we describe how we obtain the derivative estimates $f_{+\circ}$, $f_{-\circ}$, $g_{\circ+}$, and $g_{\circ-}$ at the edge midpoints, needed for the equations (15), and the corresponding weights $w_{+\circ}$, $w_{-\circ}$, $w_{\circ+}$, and $w_{\circ-}$. Each slope estimate is computed by interpolation and/or extrapolation of up to four adjacent slope samples, two on each side of the edge.

For example, the estimate $g_\mathrm{m} = g_{\circ+}$ for the derivative $\partial Z/\partial y$, at the mid point $p_\mathrm{m} = (u, v + \frac{1}{2})$ of the edge between $q[u, v]$ e $q[u, v + 1]$, is computed from the four given derivative samples $g_\mathrm{a} = g[u - 2, v]$, $g_\mathrm{b} = g[u - 1, v]$, $g_\mathrm{c} = g[u, v]$ and $g_\mathrm{d} = g[u + 1, v]$, which are the derivatives sampled at the points $p_\mathrm{a} = p[u - 2, v] = (u - \frac{3}{2}, v + \frac{1}{2})$, $p_\mathrm{b} = p[u - 1, v] = (u - \frac{1}{2}, v + \frac{1}{2})$, $p_\mathrm{c} = p[u, v] = (u + \frac{1}{2}, v + \frac{1}{2})$ and $p_\mathrm{d} = p[u + 1, v] = (u + \frac{3}{2}, v + \frac{1}{2})$, respectively. Note that the distances (signed) of those points to p_m are $-\frac{3}{2}$, $-\frac{1}{2}$, $+\frac{1}{2}$ and $+\frac{3}{2}$ respectively. The reliability weight $w_\mathrm{m} = w_{\circ+}$ of the result g_m is computed from the weights w_a, w_b, w_c and w_d of the input derivative samples. See figure 6.

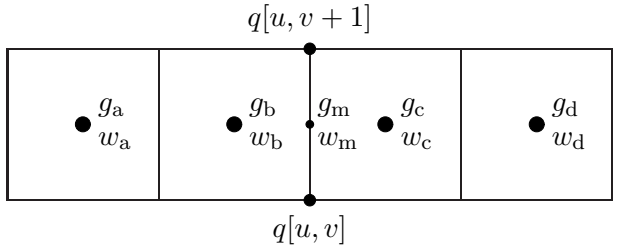


Figure 6: Data used to compute the interpolated derivative $g_\mathrm{m} = \partial Z/\partial y$ at the point $p_\mathrm{m} = (u, v + 1/2)$, and its weight w_m.

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

$$\begin{aligned} g_- &= (-g_\mathrm{a} + 3g_\mathrm{b})/2 \\ g_\circ &= (g_\mathrm{b} + g_\mathrm{c})/2 \\ g_+ &= (+3g_\mathrm{c} - g_\mathrm{d})/2 \end{aligned} \tag{16}$$

Given the interpretation of $w[u,v]$ as the reciprocal of the variance of the noise in $g[u,v]$, the weights of said estimates will be

$$\begin{aligned} w_- &= 4/(1/w_\mathrm{a} + 9/w_\mathrm{b}) \\ w_\circ &= 4/(1/w_\mathrm{b} + 1/w_\mathrm{c}) \\ w_+ &= 4/(9/w_\mathrm{c} + 1/w_\mathrm{d}) \end{aligned} \tag{17}$$

We then take the weighted average g_m of those three estimates and compute the corresponding weight w_m, by the formulas

$$\begin{aligned} g_\mathrm{m} &= \frac{w_- g_- + w_\circ g_\circ + w_+ g_+}{w_- + w_\circ + w_+} \\ w_\mathrm{m} &= w_- + w_\circ + w_+ \end{aligned} \tag{18}$$

Note that each of the weights w_-, $w_\circ$ or w_+ will be zero if its formula depends on a weight that is zero. In particular, derivative samples that lie outside the index range of the arrays f and g are automatically excluded from the estimate g_m. If one or more of the weights w_a, w_b, w_c and w_d are zero, the formulas (**??**)–(16) get simplified as follows:

Table 1: Corresponding average gradients and weights for samples with zero weight.

Case	Interpolated derivative g_m	Weight w_m
$w_\mathrm{a} = 0$	$(w_\circ g_\circ + w_+ g_+)/(w_\circ + w_+)$	$w_\circ + w_+$
$w_\mathrm{b} = 0$	g_+	w_+
$w_\mathrm{a} = w_\mathrm{b} = 0$	g_+	w_+
$w_\mathrm{a} = w_\mathrm{c} = 0$	—	0
$w_\mathrm{a} = w_\mathrm{d} = 0$	$g_\circ$	$w_\circ$

The remaining cases ($w_\mathrm{c} = 0$, $w_\mathrm{b} = w_\mathrm{d} = 0$, etc.) are symmetric to the above.

Note that we do not try to interpolate the derivative g_m by combining non-consecutive samples, such as $(g_\mathrm{a} + 3g_\mathrm{c})/4$. If the intermediary samples (g_b, in this case) have low or zero weight, those two samples may straddle a discontinuity of the height Z; in which case the interpolated derivative would be meaningless, even if the two samples have large weights w_a and w_c.

If all three weights w_-, $w_\circ$ and w_+ are zero, the final weight w_m will be zero by the formula (18). In this case, the estimate g_m is irrelevant. We will denote the computation described by formulas (16–18) as the procedure

$$(g_\mathrm{m}, w_\mathrm{m}) \leftarrow \textsc{Interpolate}(g_\mathrm{a}, w_\mathrm{a}, g_\mathrm{b}, w_\mathrm{b}, g_\mathrm{c}, w_\mathrm{c}, g_\mathrm{d}, w_\mathrm{d}) \tag{19}$$

In order to obtain $f_{+\circ}$, the estimated value of $\partial Z/\partial x$ at the mid point $s = (u + \frac{1}{2}, v)$ of an horizontal edge, we apply this same Interpolate procedure to the four samples above and below the edge, two on each side; that is, $f_\mathrm{a} = f[u, v-2]$, $f_\mathrm{b} = f[u, v-1]$, $f_\mathrm{c} = f[u, v]$ and $f_\mathrm{d} = f[u, v+1]$ with their respective weights. The estimates $g_{\circ-}$ and $f_{-\circ}$ are computed in the sane way.

5. Topological multi-grid integration

The core of our algorithm is a *decimation* procedure which removes a certain fraction of the vertices of the input mesh G, and adds some bridging edges, producing a smaller mesh G'. The vertices of G' are a subset of $\mathcal{V}\,G$, and the edges of G' are constructed so that they summarize the weight and difference information contained in the corresponding edges of G. The integration problem is then solved recursively for the mesh G', generating height estimates z' for its vertices. By interpolation of those heights we obtain a initial guess z of the heights in the original mesh G. The heights z are then adjusted by the iterative Gauss-Seidel method. The recursion is interrupted when the mesh z is reduced to a single vertex v, at which point we can set $z[v]$ to zero.

In other words, we build 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 mesh $G^{(k)}$. We compute then the solutions $z^{(m)}, z^{(m-1)}, \ldots, z^{(0)}$, in this order, where $z^{(m)}[v]$ is zero, and each $z^{(k)}$ is obtained from $z^{(k+1)}$ by interpolation and Gauss-Seidel iterations. The output of the algorithm is then the map $z^{(0)}$. See figure 7.

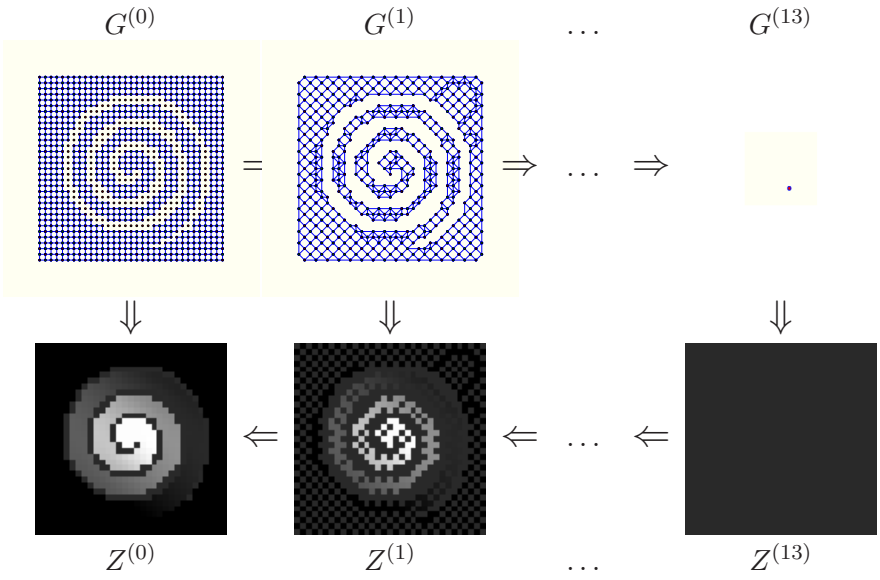


Figure 7: Multi-scale mesh integration.

Formally, the algorithm is the recursive procedure MSMeshIntegrate, whose the pseudo-code is given in figure 8. It receives as input the weighted differences mesh G, the maximum number of iterations q, and a tolerance ε, and returns a vector of values z for $\mathcal{V}\,G$.

Procedure MSMeshIntegrate(G, q, ε)

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

Figure 8: Main procedure of the multi-scale integrator for weighted difference meshes

5.1. Mesh decimation

The procedure Decimate called in the step 4 receives a simple connected and planar mesh G and returns a smaller mesh G', which is itself simple, planar and connected.

The DECIMATE procedure first partitions $\mathcal{V}G$ in a set R of vertices to be removed, and a complementary set K of vertices to be kept. The set R is a maximal subset of independent (pairwise non-adjacent) vertices with maximum degree 6. To achieve this goal, the procedure uses an attribute *mark* for each vertex, which can have three possible states: REMOVE, KEEP and BLANK. Initially every vertex is marked as BLANK. For each degree k, from 1 to 6, the procedure scans sequentially all the vertices that are in BLANK state. When a vertex of degree k is found, it is marked as REMOVE and all its neighbors which still BLANK are marked as KEEP. In the end, the set R consists of all vertices marked as REMOVE, and K has all the vertices marked as BLANK or KEEP.

After defining the sets K and R, the vertices in R are removed from G. Every time that a vertex u is removed, the edges connected to u are also removed. If u is of degree 1, no additional action is needed. If u has degree greater or equal than 2, new edges are added to G', connecting the neighbors of u, as described in section 5.4. Note that all those vertices are in K and therefore they will be always vertices of G'.

5.2. Interpolation of heights

Once a solution z' is obtained for the reduced mesh G' (step 6 of procedure MSMESHINTEGRATE), it is converted into an initial guess of z to the complete mesh G by the procedure INTERPOLATE (step 7). Initially, for each vertex v in K (which exists in both meshes), we set $z[v] \leftarrow z'[v]$. Then, for each vertex u in R (which exists only in G), we compute $z[u]$ by the equation of vertex equilibrium (7). Note that each neighbor $v \in G[u]$ belongs to K, and therefore its height $z[u]$ is already defined.

5.3. Iterative adjustment

The initial estimate z computed by INTERPOLATE satisfies the vertex equilibrium equation (7) for the vertices in R, but generally not for the vertices in K. That estimate used as initial guess for the Gauss-Seidel SOLVESYSTEM procedure (step 8). Each iteration of SOLVESYSTEM examines each vertex $u \in \mathcal{V}G$ and uses the equation (7) to re-compute its value $z[u]$ from the current values $z[v]$ of its neighbors.

The SOLVESYSTEM procedure stops after a specific number of iterations κ, or when the variation between a value $z[u]$ from one iteration to another is smaller than a given tolerance ε, for each vertex u, whichever happens first.

Note that, at each level of recursion, the limit κ to the number of iterations is increased by a factor $1/\sqrt{\beta}$, and the tolerance reduced by $\sqrt{\beta}$

(step 6); where $\beta = \#\,\mathcal{E}\,G'/\#\,\mathcal{E}\,G$ is the *mesh reduction factor* achieved by DECIMATE (step 4).

5.4. Adding the new edges

We now describe the new edges that are added by the DECIMATE procedure, between the neighbors of a removed vertex u. Basically, the endpoints, weights and differences of these new edges are chosen so that the solution $z'[v]$ for the mesh G' is close to the solution $z[v]$, for each vertex $v \in K$.

More precisely, let k be the degree of u in G; let $e_0, e_1, \ldots, e_{k-1}$ the edges connected to u in counter-clockwise order; and let $v_0, v_1, \ldots, v_{k-1}$ the corresponding destination vertices. Let $w_i = w[e_i]$ be the weight of the edge e_i and $d_i = d[e_i]$ its difference. It is easy to show that the solution z' for each G' is a subset of the solution z of G, if, for each pair i, j, we add an edge $e'_{i,j}$ from v_i to v_j with the following attributes:

$$d'_{ij} = d_j - d_i \qquad w'_{ij} = \frac{w_i w_j}{w_{\text{tot}}} \tag{20}$$

where w_{tot} is the sum of weights $w_0, w_1, \ldots, w_{k-1}$. We call this operation — removal of u and every edge e_i connected to it, and creation of edges e'_{ij} between all pairs of neighbors of u — the *star-clique swap* for u.

In particular, if the vertex u has degree $k = 2$, the swap will only add a new pair of opposite edges $e'_{0,1}$ e $e'_{1,0}$. If the degree k is 3, only the edges $e_{0,1}$, $e'_{1,2}$, $e'_{0,2}$ and their reverses will be added. In both cases the planarity of the mesh is preserved.

However, if k is greater than or equal to 4, the star-clique swap would make G' non-planar, which would severely affect the algorithm's efficiency. Therefore, in this case we add only the edges $e'_{i,i+1}$ which connect the successive vertices in a cycle; namely, between v_i and v_{i+1} for $i \in \{0, 1, \ldots, k-1\}$, with indices reduced modulo k. We call this operation the *star-cycle swap* for u. The differences $d'_{i,i+1}$ between such edges are assigned by formula (20), that is:

$$d'_{i,i+1} = d_{i+1} - d_i \tag{21}$$

whereas the weights $w'_{i,i+1}$ are given by distinct formulas depending on the degree k, as given in the table 2.

Table 2: Formulas for the weight w'_{01} of the new edge $e'_{01} = (v_0, v_1)$ created by the star-cycle swap, for each degree k. The same formulas are valid for every other edge $e'_{i,i+1}$, except that the indices are incremented by i modulo k.

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

In contrast with the star-clique swap, the star-cycle swap does not ensure that the height values determined from the mesh G' are exactly the same as those determined by the mesh G. In fact, it is not possible to ensure this condition by adding only a subset of edges e'_{ij} of the clique, whose differences d'_i and weights w'_i are computed by local formulas (that is, dependent only on the attributes of the edges incident to u). To ensure that condition, one would have to analyze the entire G mesh, and essentially solve the integration problem – which would make the multi-scale approach pointless.

However, our experiments shows that, by adding only the k edges from the cycle with the weights shown in table 2, the solution z' of the mesh G' has the correct low frequency terms of the solution z of G. This implies that the corrections that need to be made to the mesh z are highly localized, and can be removed with a few Gauss-Seidel iterations.

5.5. Removing parallel edges

The star-cycle swaps in the DECIMATE procedure may create parallel edges, which can be either new edges added when applying the swap to distinct R vertices, or edges from the original mesh G that have both extremities within the set K and thus were not removed.

Therefore, after all the star-cycle swaps have been applied, the procedure DECIMATE simplifies the mesh G', replacing every group of two or more edges with same source and destination for a single equivalent edge. In particular, if the edges e' and e'' have the same source and destination, they are replaced by a single edge e with the following attributes

$$w[e] = w[e'] + w[e''] \qquad d[e] = \frac{w[e']d[e'] + w[e']d[e']}{w[e'] + w[e'']} \tag{22}$$

This process is repeated until there are no more parallel edges. It is easy to see that this process does not change the solution z defined by the equations (7).

5.6. Analysis of the algorithm

Correctness: It is easy to see that, if the input mesh G is connected, planar, and simple, the mesh G' returned by DECIMATE will be connected, planar, and simple too. Therefore, the same will hold for all levels of the multi-scale mesh pyramid. The algorithm MSMESHINTEGRATE is therefore immune to problems caused by premature loss of connectivity, even if the original mesh has parts that are connected to each other only by a single edge or path.

Moreover, the vertex equations (7) are dominated by the diagonal. Therefore, when the Gauss-Seidel algorithm is applied at scale 0, with proper values of κ and ε, will converge to the unique solution $z = z^{(0)}$ of those equations, independently of the initial estimate obtained by the decimated mesh $G^{(1)}$.

Time and space: The efficiency of the algorithm depends on the reduction factor β obtained by the procedure DECIMATE, and on the number of Gauss-Seidel iterations needed in each level. Let $N = \#\,\mathcal{V}\,G$, $N_k = \#\,\mathcal{V}\,G^{(k)}$, $M = \#\,\mathcal{E}\,G$, and $M_k = \#\,\mathcal{E}\,G^{(k)}$. Let β_k be the mesh reduction factor at step k, that is, $\#\,\mathcal{E}\,G^{(k+1)}/\#\,\mathcal{E}\,G^{(k)}$; and let $\hat{\beta}$ be the largest of those numbers. If $\hat{\beta} < 1$ then the maximum scale m will be $\log_{1/\hat{\beta}} N = O(\log N)$, and the total number N_{tot} of vertices of all meshes will be at most $N/(1-\hat{\beta}) = O(N)$.

We now show that the upper limit for $\hat{\beta}$ is less than 1. Since G is simple and planar, by Euler's formula we conclude that $M \leq 6N$, and that G has at most $N/7$ vertices with degree less than or equal to 6 [27]. The same conclusions are valid about N_k and M_k, for all reduced meshes $G^{(k)}$. From those facts, it is possible conclude that $\hat{\beta} \leq 41/42 \approx 0.976$ [27]. Therefore M is at most $20 \log_2 N$ and N_{tot} is at most $42N$. However, these are worst-case theoretical bounds; in typical meshes, such as those that are obtained from rectangular grid data, the reduction factor $\hat{\beta}$ turns out to be close to 0.6, which gives $M \approx 1.4 \log_2 N$ and $N_{\text{tot}} \approx 2.5N$.

The amount of memory required by the algorithm is dominated by the data structure that represents each mesh $G^{(k)}$. A simple representation, which suffices for our purposes, consists of an *edge table* with $2M_k$ entries, each one containing the destination, weight, and difference of each directed edge $G^{(k)}$, ordered by the source vertex; and a *vertex table* with N_k entries, which stores, for each vertex v, the index of the first entry in the edge table

with source v. The total storage space is therefore at most $N_k + 2 \times 3M_k \leq 19N_k$ words for the mesh $G^{(k)}$, and at most $19N_{\text{tot}} \approx 47.5N$ words for all meshes in the pyramid.

The planarity condition also ensures that the decimation algorithm runs in time $O(N_k + M_k) = O(N_k)$ for each level k. Therefore all levels of the pyramid are built in total time $O(N_{\text{tot}}) = O(N)$.

The time necessary for a single iteration of the Gauss-Seidel at the level k is $\Theta(N_k+2M_k) = \Theta(N_k)$, and the number of iterations executed at this level is $(N\hat{\beta}^k)(\kappa/\hat{\beta}^{k/2}) = N\hat{\beta}^{k/2}$; which implies total time of $O(N/(1 - \sqrt{\hat{\beta}})) = O(N)$.

Convergence speed: As in the work of Saracchini *et al.* [11], this algorithm exploits the fact that a Gauss-Seidel iteration converges rapidly if the error consists mostly of high frequency spatial errors. When the mesh is decimated, the high frequency components are mostly suppressed, while the low frequency components have their wavelength reduced by a factor approximately $\sqrt{\beta_k}$. Thus, the recursively computed solution $z^{(k+1)}$, after expanded to the previous scale $z^{(k)}$, will be correct mostly in the low frequency components, with small-scale details missing. Those missing details will be recovered after a small number of Gauss-Seidel iterations, which is largely independent of N_k. The whole recursive process is fast because each spectral component of the height map is computed in the scale where its spatial frequency is low.

Unfortunately, this intuitive explanation is not easy to formalize, much less easy to demonstrate theoretically, since it needs a formulation of "frequency" for an irregular topological mesh, which is outside the scope of this work. However, the experimental tests show that convergence is achieved after few iterations, even with instances where other multi-scale methods fail.

6. Experiments and discussion

In this section we compare the cost and precision of our topological multi-grid integrator (`TMG2`, here abbreviated to `MG`) with other published methods.

We consider only methods which are able to deal with discontinuities and missing data. They comprise: the Poisson-based integrators presented by Agrawal *et al.* [19], specifically the M-Estimators (`AM`) and Diffusive Affine Transform (`AT`) methods; the multi-scale grid integrator of Saracchini *et al.* [11] (`MS`); and the iterative methods developed by Quéau and Durou [23],

namely Isotropic Total Variation (DT) and L^1 Functional (DL). See table 3. Note that the popular Fourier-based integrator of Frankot and Chellapa [16], in particular, cannot cope with missing data or non-rectangular domains.

We could not test the integrator proposed by Breuß *et al.* [26] since its source code was not available at the moment of writing this paper. Anyway, it is a pre-processing step for some Poisson or Krylov-based integrator, which could be one of the methods listed above.

Table 3: Tested integration methods.

Tag	Description
AT	Diffusive Affine Transform [19, 28]
AM	M-Estimators [19, 28]
DT	Isotropic Total Variation [23]
DL	L^1 Functional [23]
MS	Weighted multi-scale on regular grid [11]
MG	Weighted multi-scale on differences mesh (TMG2)

The algorithms provided by Agrawal and Durou (AT, AM, DT, and DL) were implemented by the authors in MATLAB [29]. In order to make the comparisons more meaningful, we modified Agrawal's algorithms to use a weight image given as an additional input, instead of letting them estimate the weights from the f and g images. Our own algorithms (MS and MG) were implemented in C.

All tests were executed on an Intel I5-3470 at 3.2 GHz with 16GB of RAM memory. The MATLAB scripts were executed in MATLAB version 2009b under Windows 7, whereas the compiled C programs were compiled with the Gnu C compiler [30] and executed under Linux Debian 7.0. In order to avoid inconsistent results due to processor swapping and the in-built parallelization of MATLAB, the running times were measured with each process bound to only one core of the CPU.

6.1. Test datasets

We used six test datasets, each originally obtained as regular height sample arrays z^* with 2048×2048 samples. Four of the datasets (spdome, mixwav, cbabel, and cpiece) were obtained by sampling mathematical functions. The bebust dataset was obtained from a laser-scanned bust of Beethoven, a standard benchmark in 3D modeling [31]. The dtbust dataset was obtained from a live subject by a 3DMT scanner [32], which resulted in

a triangular mesh surface with 76,601 faces. The mesh was converted to a height map and numerically differentiated.

Note that the datasets `cbabel`, `cpiece`, and `dtbust` have cliffs where the gradient is undefined; and both `bebust` and `dtbust` have extended regions where the gradient data is not available. The `dtbust` set also has weakly connected regions due occlusions, and several small regions with missing data. For all models, the weight image was created by hand with an image editor, as a gray-scale image where the cliffs and undefined regions were marked in black (weight 0) over a background of white (weight 1). See figure 9 and 10.

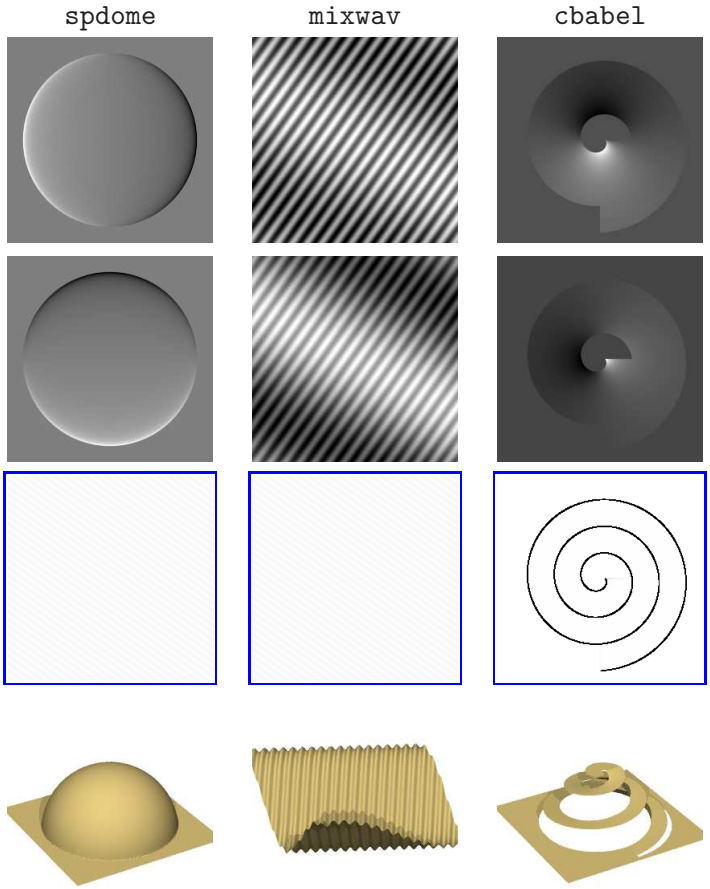


Figure 9: The test datasets `spdome`, `mixwav` and `cbabel`, showing the gradient maps f and g (topmost two rows), the weight maps w (third row), and the correct height map z^* in perspective (bottom row).

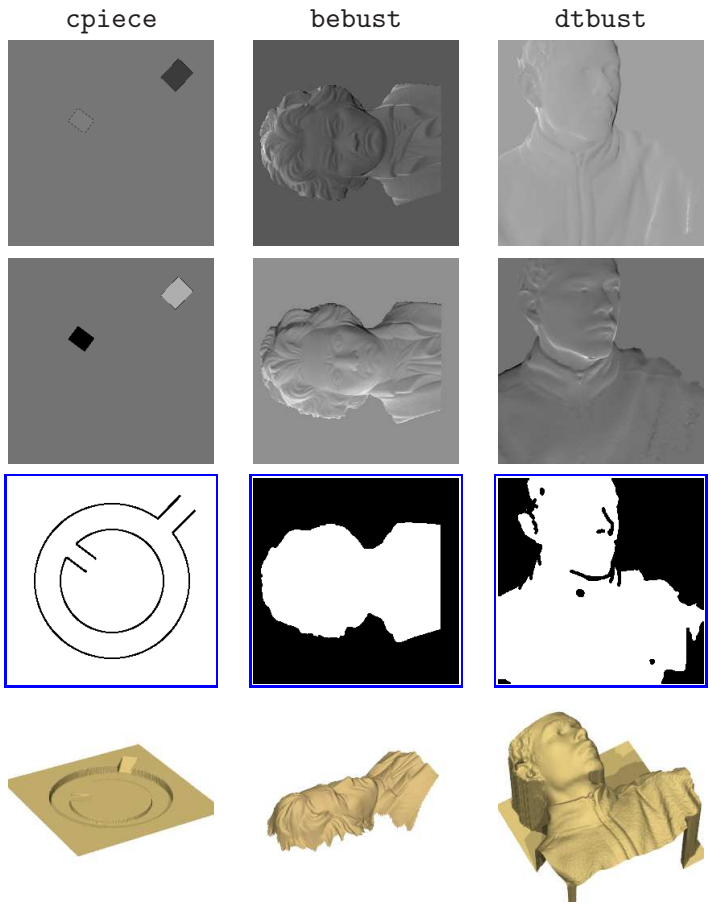


Figure 10: The test datasets `cpiece`, `bebust` and `dtbust`.

We processed each dataset as given, and also after perturbing the gradient maps f and g by mixing them with 30% Gaussian white noise.

6.2. Accuracy Tests

In the accuracy tests, we verified the quality of the computed height maps by comparing them with the known reference height map ("ground truth") z^*.

For these tests, the images f, g, w, and z^* were reduced to 256×256 samples. The maximum number of iterations κ at scale 0 was set respectively at 50 and 20 for the `MS` and `MG` integrators, and to the default 50 and 500 iterations for `DT` and `DL`. The *absolute* accuracy of each computed map z was quantified as the standard deviation of the difference $\eta = z[w] - z^*[v]$, weighted by $w[v]$. The *relative* accuracy was quantified as η/R, where R

667 is the standard deviation of the reference height map z^*, also weighted by
668 $w[v]$. These values are summarized in tables 4–5.

669

Table 4: Absolute and relative root-mean-square errors η and η/R of each method for the test datasets, without noise.

Method	η	η/R	η	η/R	η	η/R
	`spdome`		`mixwav`		`cbabel`	
AT	1.82	5.2%	0.89	2.3%	0.02	0.1%
AM	0.58	1.6%	0.46	1.2%	0.02	0.1%
DT	0.05	0.2%	0.02	0.0%	4.51	18.55%
DL	0.04	0.1%	0.67	0.0%	19.90	102.2%
MS	0.19	0.5%	0.36	0.9%	25.31	134.8%
MG	0.04	0.1%	0.02	0.0%	0.03	0.0%
	`cpiece`		`bebust`		`dtbust`	
AT	0.15	0.3%	1.59	11.07%	0.64	2.5%
AM	0.15	0.3%	0.30	2.0%	0.71	2.8%
DT	0.89	17.8%	1.62	10.9%	0.46	1.8%
DL	4.32	104.3%	1.28	8.8%	5.46	23.6%
MS	5.26	138.4%	1.02	6.4%	2.99	12.4%
MG	0.00	0.0%	0.87	5.4%	0.39	1.5%

Table 5: Absolute and relative root-mean-square errors η and η/R of each method for the test datasets, with 30% of Gaussian noise added.

Method	η	η/R	η	η/R	η	η/R
	`spdome`		`mixwav`		`cbabel`	
AT	3.30	9.8%	4.75	13.0%	0.80	3.0%
AM	0.64	1.8%	0.51	1.3%	0.86	3.3%
DT	0.46	1.3%	0.37	0.9%	12.47	58.6%
DL	0.48	1.4%	0.92	2.4%	24.36	129.7%
MS	0.34	0.9%	0.44	1.1%	25.36	135.1%
MG	0.39	1.1%	0.34	0.9%	0.76	2.9%
	`cpiece`		`bebust`		`dtbust`	
AT	0.55	10.0%	1.94	13.9%	1.22	4.9%
AM	0.54	9.9%	0.40	2.7%	0.71	2.8%
DT	1.46	30.2%	1.21	8.2%	1.21	4.9%
DL	4.46	114.3%	2.26	15.5%	9.86	45.1%
MS	5.25	137.9%	0.90	5.6%	2.98	12.3%
MG	0.46	8.7%	0.93	5.8%	0.59	2.3%

Note that the topological multi-scale method described in this article (`MG` = `TMG2`) is generally more accurate than the other five methods tested, except that `AM` got a smaller relative error (2.0% versus 5.4%) on the `bebust` dataset.

We observed also that our previous uniform-grid multi-scale method `MS` failed on the datasets `cbabel`, `cpiece`, and `dtbust`, because of loss of connectivity at the smallest levels of the pyramid. The iterative method `DL` of Quéau and Durou failed on the same datasets, too; while their `DT` method did badly on `cbabel` and `cpiece`. With all other data and method combinations, the solution obtained was fairly accurate. The results of the most significant failure cases are shown in figure 11–13.

All methods, including ours, were fairly sensitive to noise on the `cpiece` dataset, presumably because any gradient noise on the narrow bridges between the three main regions implied a significant change on the relative heights of those regions. Noise also had a notable impact on the accuracy of `AT` and `AM` with the `mixwav` set, and of `DT` with the `cbabel` set.

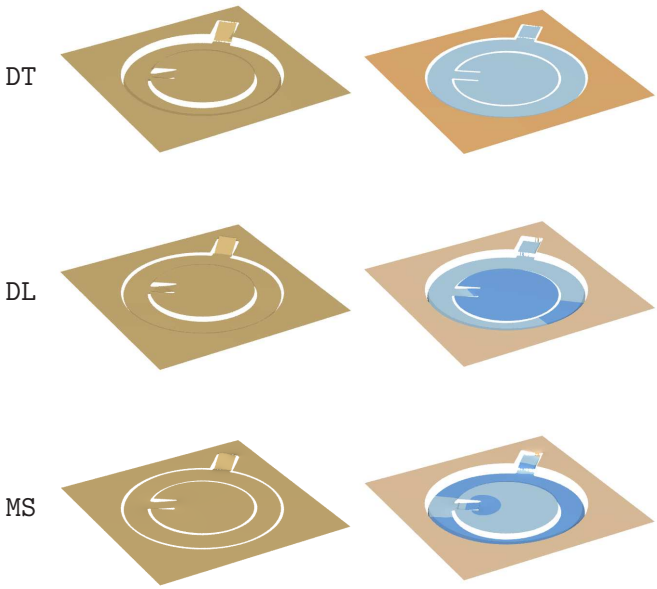


Figure 11: Failure examples: At left, the height maps z computed by algorithms `DL`, `DT`, and `MS` on the dataset `cpiece` without noise. At right, the absolute error maps $z - z^*$. Blue and orange hues in the latter indicate that the computed height was below or above the correct height, respectively.

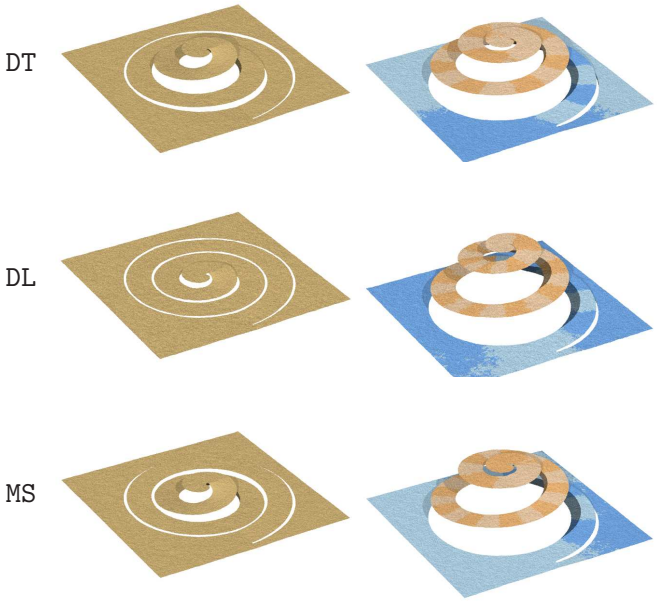


Figure 12: Failure examples: Height maps z (left) and absolute error maps $z - z^*$ (right) obtained with methods `DL`, `DT`, and `MS` on the dataset `cbabel` with 30% of Gaussian noise.

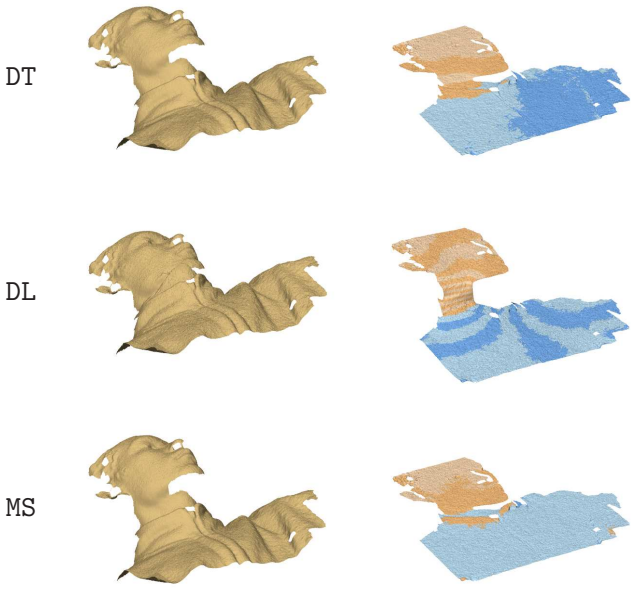


Figure 13: Failure examples: Height maps z (left) and absolute error maps $z - z^*$ (right) obtained with methods `DL`, `DT`, and `MS` on the dataset `dtbust` with 30% of Gaussian noise.

6.3. Running times

In order to evaluate the efficiency and scaleability of our algorithm, we measured the running time $t(N)$ for the integration of two datasets (`spdome` and `dtbust`), each reduced from the original size 2048×2048 to various sizes $N = n \times n$, with $n = 64$, 128, 256, 512, 1024, and 2048. For the multi-scale methods, we counted the computational time of all iterations performed in all scales.

For the single-scale methods of Agrawal and Durou (`AT`, `AM`) and Quéau and Durou (`DT`, and `DL`), we considered only the the time needed to solve the linear system, since the pre-processing stages have linear computational cost which may be higher than that of solving the system itself. For this evaluation, we specified a maximum of 2 iterations at scale 0 for the direct-solving methods `DT` and `DL`. As for the multi-scale methods based on Gauss-Seidel iteration, we specified 50 for our previous method `MS` on regular grids, and 20 for our new method `MG` with differences mesh.

The absolute times in seconds cannot be compared due substantial differences of programming language and libraries. Therefore, we focused on

how the computing costs scale with the size of the problem. The results are shown in figures 14 and 15 and table 6.

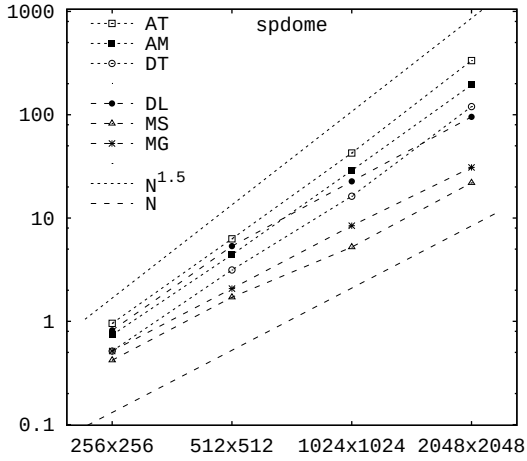


Figure 14: Plots of the system solving time $t(N)$, in seconds, as a function of the total number of samples $N = n \times n$, for the six methods (`AT`, `AM`, `DT`, `DL`, `MS` and `MG`), on the `spdome` dataset. For slope comparison, the plot also shows the functions αN (dashed) and $\gamma N^{1.5}$ (dotted), for arbitrary coefficients α and γ.

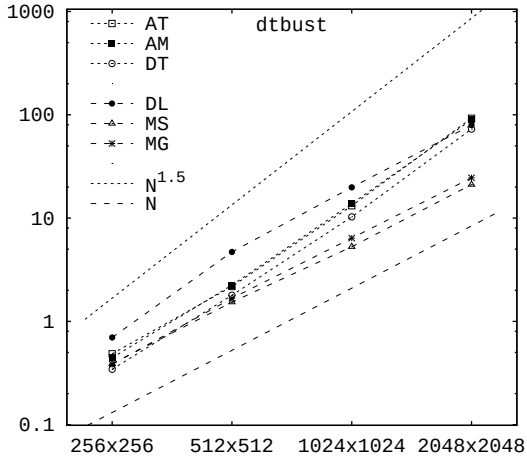


Figure 15: System solving time $t(N)$, in seconds, for the six methods on the `dtbust` dataset.

Table 6: System solving time $t(N)$, in seconds, for the six methods on the `spdome` and `dtbust` datasets.

`spdome`						
N	`AT`	`AM`	`DT`	`DL`	`MS`	`MG`
64×64	0.04	0.04	0.05	0.07	0.02	0.02
128×128	0.17	0.15	0.22	0.19	0.09	0.10
256×256	0.95	0.73	0.51	0.81	0.42	0.51
512×512	6.28	4.41	3.13	5.34	1.70	2.07
1024×1024	45.36	29.24	16.98	22.05	5.22	8.43
2048×2048	351.04	194.06	119.93	95.32	21.92	30.88
`dtbust`						
N	`AT`	`AM`	`DT`	`DL`	`MS`	`MG`
64×64	0.04	0.06	0.04	0.07	0.02	0.01
128×128	0.09	0.11	0.09	0.18	0.08	0.08
256×256	0.48	0.44	0.34	0.69	0.38	0.38
512×512	2.17	2.24	1.79	4.70	1.53	1.67
1024×1024	13.16	13.80	10.26	19.83	5.27	6.40
2048×2048	92.51	89.46	72.78	79.54	21.10	25.03

Figures 14 and 15 show that the running time $t(N)$ grows linearly with N for our multi-scale methods (`MS` and `MG`) and for the method `DL` of Durou *et al.*. For the other three methods (`AT`, `AM`, and `DT`), the running time grows proportional to $N^{1.5}$, mostly due their use of a Cholesky-based sparse linear systems solver.

7. Conclusions

The main contributions of this article are a novel multi-scale gradient integration method, based on irregular differences mesh (`TMG2`); and a method to convert a regular grid of discrete gradient data with localized uncertainties or missing data into such a mesh. The combination of the two methods can cope with non-uniform errors and gaps in the input gradient data, as well as (known) discontinuities in the height field, such as may be obtained through photometric stereo.

We compared our `TMG2` method with others found in the literature, through tests with synthetic and real data. In most tests, our method was

found to be asymptotically faster than state-of-the-art single-scale methods, and significantly more accurate. The use of meshes with arbitrary topology allows us to get the speed benefits of multi-scale Gauss-Seidel methods without the errors caused by region disconnection at coarser scales.

Specifically, the running time and memory requirements of the `TMG2` method scale linearly with the number N of input samples, while the running time of earlier methods, based on directly solving the Poisson equations, grow proportionally to $N^{1.5}$. The `TMG2` method can be easily parallelized to SIMD processing platforms (such as GPUs and FPUs) due the simplicity of the data structures and the use of local decimation and iteration methods.

We expect that `TMG2` will be attractive in any applications of gradient integration where speed, accuracy, and robustness are important; especially in the use of photometric stereo for industrial quality control, surveillance and biometric identification, remote sensing, stereo microscopy, and fast single-camera 3D scanning.

Acknowledgments

We would like to thank Jean-Denis Durou, Yvain Quéau and Amit Agrawal, who kindly provided the code of their integrators. We also express our gratitude to Melvin Smith for hosting Rafael Saracchini in his laboratory during his doctoral studies. This work was partly funded by CnPQ (grant 301016/92-5), CAPES, FAPESP (2013/07699-0), FAPERJ (E-26/111.389/2014), and EPSRC.

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. Journal of Computer Vision 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, Optical Engineering 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: Proc. 11th Intl. Conf. on Advanced Concepts for Intelligent Vision Systems (ACIVS), Vol. 5807, 2009, pp. 445–455.

[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 Vision and Image Understanding In Press. doi:10.1016/j.cviu.2010.03.001.

[9] D. Terzopoulos, Image analysis using multigrid relaxation methods, IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-8 (2) (1986) 129–139.

[10] D. Terzopoulos, The computation of visible-surface representations, IEEE Transactions on Pattern Analysis and Machine Intelligence 10 (4) (1988) 417–438.

[11] R. F. Saracchini, J. Stolfi, H. C. Leitão, G. A. Atkinson, M. L. Smith, A Robust Multi-Scale Integration Method to Obtain the Depth from Gradient Maps, Computer Vision and Image Understanding 116 (8) (2012) 882–895.

[12] Z. Wu, L. Li, A line-integration based method for depth recovery from surface normals, Computer Vision, Graphics and Image Processing 43 (1) (1988) 53–66.

[13] A. Robles-Kelly, E. R. Hancock, Surface height recovery from surface normals using manifold embedding, in: Proc. Intl. Conf. on Image Processing (ICIP), 2004.

[14] R. Fraile, E. R. Hancock, Combinatorial surface integration, in: Proc. 18th Intl. Conf. on Pattern Recognition (ICPR'06) Volume 1, 2006, pp. 59–62.

[15] A. Agrawal, R. Chellappa, R. Raskar, An algebraic approach to surface reconstruction from gradient fields, in: Proc. 2005 Intl. Conf. on Computer Vision (ICCV), 2005, pp. 174–181.

[16] R. T. Frankot, R. Chellappa, A method for enforcing integrability in shape from shading algorithms, IEEE Trans. Pattern Analysis and Machine Intelligence 10 (4) (1988) 439–451.

[17] 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.

[18] 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.

[19] A. Agrawal, R. Raskar, R. Chellappa, What is the range of surface reconstructions from a gradient field?, in: Proc. 9th European Conf. on Computer Vision (ECCV), Vol. 3951, 2006, pp. 578–591.

[20] D. Reddy, A. Agrawal, R. Chellappa, Enforcing integrability by error correction using ℓ_1-minimization, in: Proc. 2009 IEEE Conf. on Computer Vision and Pattern Recognition, 2009, pp. 2350–2357.

[21] A. S. Georghiades, P. N. Belhumeur, D. J. Kriegman, From few to many: Illumination cone models for face recognition under variable lighting and pose, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (2001) 643–660.

[22] B. K. P.Horn, Height and gradient from shading, Tech. Rep. AI Memo 1105, Massachusetts Institute of Technology (1989).

[23] Y. Queau, J.-D. Durou, Edge-Preserving Integration of a Normal Field: Weighted Least-Squares, TV and L1 Approaches, in: J.-F. Aujol, M. Nikolova, N. Papadakis (Eds.), Scale Space and Variational Methods in Computer Vision, Vol. 9087 of Lecture Notes in Computer Science, Springer International Publishing, 2015, p. 576–588.

[24] J.-D. Durou, Y. Quéau, J.-F. Aujol, Normal integration–part i: A survey.

[25] Y. Quéau, J.-D. Durou, J.-F. Aujol, Normal integration–part ii: New insights.

[26] M. Breuß, Y. Quéau, M. Bähr, J.-D. Durou, Highly efficient surface normal integration, in: Proceedings of the Conference Algoritmy, 2016, pp. 204–213.

[27] D. G. Kirkpatrick, Optimal search in planar subdivisions, SIAM J. on Computing 12 (1983) 28–35.

[28] A. Agrawal, Matlab/Octave code for robust surface reconstruction from 2d gradient fields, Available from `http://www.umiacs.umd.edu/ aagrawal/software.html`. Accessed on 2010-05-01, see [19] (2006).

[29] The Mathworks Inc., MATLAB, Online document at `https://www.mathworks.com/products/matlab.html`. Accessed on 2017-04-12. (2017).

[30] Free Software Foundation, GCC, the GNU compiler collection, Online document at `https://gcc.gnu.org/`. Accessed on 2017-04-12. (2017).

[31] Université de Tolouse, Codes for photometric stereo, Dataset at `http://ubee.enseeiht.fr/photometricstereo/` , accessed on 2016-06-22 (2010).

[32] 3dmdface system, Company page at `http://www.3dmd.com/3dmdface.html`, accessed on 2010-04-22 (2010).