A topological multi-grid method for integration of two-dimensional meshes

Rafael F. V. Saracchini, Carlos A. Catalina, Jorge Stolfi, and Helena C.G. Leitão

Abstract—We describe a particular variant of the algebraic multigrid (AMG) method, which we call two-dimensional topological multi-grid (TMG2), for solving Poisson-type problems such as surface gradient integration on meshes with arbitrary topology and geometry, on the plane or other surface of low genus. The coarsening algorithm runs in linear time and 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 show that our algorithm outperforms other surface gradient integrator algorithms for practically relevant cases, especially for poorly connected meshes.
Unlike Geometric multi-grid methods, we do not assume that the mesh nodes have specific positions (and therefore do not use their positions and distances when reducing the mesh. The reduction algorithm uses only the topology of the mesh, which must be specified explicitly (in the form of a circular ordering of the edges out of each node).

Index Terms—Photometric Stereo,Surface gradient integration, 3D Reconstruction, Computer Vision

◆

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 the mean surface normal vector within each image pixel, from which one can obtain the height gradient (the partial derivatives of the surface's height Z with respect to the spatial coordinates X and Y). Although this information alone does not determine the absolute surface heights, it can yield height differences between parts of the same surface. This relative height information is sufficient for many important applications, such as industrial quality control [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 function $Z(x,y) : \mathbb{R}^2 \rightarrow \mathbb{R}$ 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*. Once the condition 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 problem of computing the heights from given slopes faces at least three 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 neighbourhood of a *gradient sampling point*. Therefore, the height function cannot be precisely determined. It can only be approximated by a member of a finite-dimensional function space, defined by a finite function basis. Such a function can be (and usually is) uniquely represented by a finite set of discrete *height samples*, each being the average of the height Z over some neighbourhood of a *height sampling point*. 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 wherever the scene's surface is affected by shadows or specular highlights, is

---

- *Rafael Saracchni and Carlos Catalina are with the Department of Simulation and Control, Technological Institute of Castilla y León,Spain.*
  E-mail: rafael.saracchini@itcl.es, carlos.catalina@itcl.es
- *Jorge Stolfi is with the Institute of Computing, State University of Campinas,Brazil.*
  E-mail: stolfi@ic.unicamp.br
- *Helena C. G. Leitão is with the Institute of Computing, Federal Fluminense University,Brazil.*
  E-mail: hcgl@ic.uff.br

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 edges of solid objects. At any sampling point that straddles those cliffs, photometric stereo and other gradient acquisition techniques usually fail to detect the (very large) gradient across the edge, and return an incorrect gradient sample that gives no clue as to the height of the cliff. See figure 1.

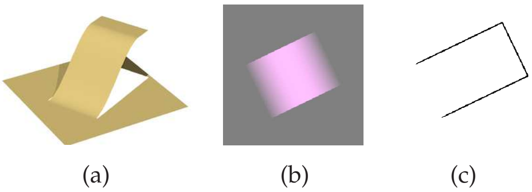

Fig. 1. A height map with cliff-like discontinuities (a), its color-coded gradient map (b), as could be obtained by photometric stereo methods, and a binary mask (c) showing the location of the cliffs. Note that the gradient map is oblivious to the cliffs, and gives no clue as to which end of the ramp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

1.2 Contributions

Here we describe the 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 (known) discontinuities in the height field. The general approach of the new method is similar to the multi-scale integrator described previously by Saracchini *et al*. [9], but using irregular and purely topological mesh in place of the rectangular grid of gradient samples used by most methods, including that one. It is a special case of the *algebraic multi-grid* (AMG) approach, optimized for planar meshes and the gradient integration problems, that preserves the connectivity even in poorly connected cases. We will denote it by the acronym TMG2, short for *two-dimensional topological multi-grid integrator*.

The irregular grid allows TMG2 to overcome the principal limitation of previous multi-scale methods, namely the possible disconnection of parts of the domain as the data is reduced to coarser and coarser scales. 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 surround some part R of the domain. 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.

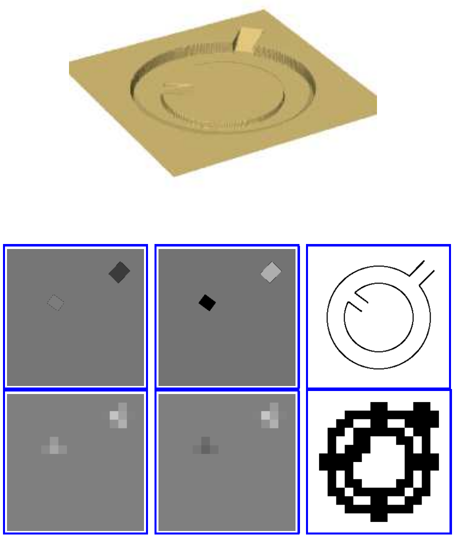

Fig. 2. Example of region disconnection in grid-based multi-scale integration. On top: perspective view of a height map $Z = Z(X, Y)$ with discontinuities. At middle: derivatives $F^{(0)} = \partial Z/\partial X$ and $G^{(0)} = \partial Z/\partial Y$ sampled on a 256×256 grid and 256×256 binary mask $W^{(0)}$ 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 a region R becomes disconnected at some scale k, the height of R relative to the rest of the surface cannot be determined from that data, not even approximately. Therefore, height map $Z^{(k)}$ computed at that scale will not be a good starting guess for the iteration at the next finer scale, and it may take many iterations to recompute at scale k to adjust the height R inside.

2 RELATED WORK

A brief review and classification of gradient integration methods for photometric stereo was provided by Saracchini *et al*. [9] and by Durou *et al*. [10], [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 system of linear equations that discretize the Poisson equation [2], [18], [19], [20], [21], by Gaussian LU or Cholesky factorization; and *local iteration* [2], [9], [22], [23] by Gauss-Seidel or similar methods. Durou *et al*. [10] proposed the classification of integrators accordingly with five main properties:

- *Fastness*: the integration process is aimed to be as fast as possible.
- *Robustness*: capability to recover an acceptable reconstruction by a normal map affected by Gaussian noise.
- *Free boundary*: capability to recover an accurate surface with unknown boundary conditions.
- *Discontinuity*: the method should be able to deal with depth discontinuities and occlusions without

segmenting the scene in separate parts without discontinuity.
- *Rectangular domain*: the integrator should be able to operate with a non-rectangular domain D.

A sixth lesser property defined by Durou is the capability to retrieve adequate results without fine tuning of the algorithm parameters (coefficients, number of iterations, etc.). So far, only one method is reported to satisfy this criteria and able to deal with discontinuities, which is the path-based integrator of Fraile and Hancock [14], which is extremely sensitive to measurement noise.

Local iteration methods had been demonstrated 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 linearisation (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 [9].

The main disadvantage of local iteration methods is the potentially slow rate of convergence. As explained in [9], 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 [24], [25]. 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.* [9] is competitive in speed with Fourier based integration, while still being able to cope with missing data, non-uniform error, and height discontinuities. However, the multi-scale approach fails to retrieve proper height map estimates within an acceptable number of iterations if the binary mask is susceptible to disconnection in smaller scales of the multi-scale pyramid.

Quéau *et al.* [23] follows a energy minimization approach, proposing the usage of 3 minimization strategies (weighted least-squares, Total Variation and L^1 optimization) inspired by image de-noising techniques. This approaches aims to detect non-differentiable elements of the surface without *a priori* detection of such regions computing automatically their weights, and is able to deal with certain cases of surface discontinuity.

More recently 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, its computed surface is reportedly less accurate than the results retrieved by the previously mentioned approaches.

3 WEIGHTED DIFFERENCES MESH

Our contribution is the representation of the Poisson-type 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 value $z[v]$, and each edge e is associated with two numerical values: the *difference* $d[e]$ and the *weight* $w[e]$.

By using a planar mesh instead of a regular grid, we can preserve the connectivity of the gradient data while reducing the spatial resolution. Moreover, even though the data is initially acquired in the form of a regular grid of pixels, the gradient sampling points may become irregularly spaced when the data is subjected to optical rectification, perspective correction and mosaic composition.

The $d[e]$ parameter is the estimated difference $z[v] - z[u]$ between the values of the destination vertex $v = \text{DST}(e)$ and the source vertex $u = \text{ORG}(e)$ which corresponds to the derivative between the sampling points u and v. The weight of the edge $w[e] \in \mathbb{R}^*$ which express the reliability of $d[e]$. More specifically, we assume that the edge difference includes a Gaussian measurement error, with expected value zero and variance proportional to $1/w[e]$. Note that edges with weight zero are not added to the graph, since they do not provide any information.

By definition, for each directed edge e in the weighted mesh, the reverse edge $\text{SYM}(e)$ is also present with $d[\text{SYM}(e)] = -d[e]$ e $w[\text{SYM}(e)] = w[e]$. However when drawing the mesh,only one of those edges are represented. See figure 3.

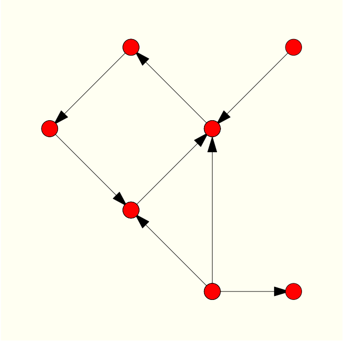

Fig. 3. A small weighted differences mesh. The edge labels are the pairs $d[e] : w[e]$. Note that only one of the edges $e, \text{SYM}(e)$ are illustrated.

We say that a mesh is *simple* if it does not have parallel edges (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 Mesh integration

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 different instance of this problem. Thus we will assume that G is always connected graph. Moreover, since the equations (4) depends only on the value differences, a solution for a connected mesh have one and only one degree of freedom: an additive constant C.

A set of values z is said *tension free* if all equations (4) are satisfied exactly. This is the case only and only if the sum of differences of edges along any directed cycle is zero. This property is equivalent to the the *zero curl* condition for the continuous integration problem of section **??**. 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

If the mesh admits a solution free of tension, $z[v]$ 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 increasing distance order from v_0 along T. Note that the weights of the edges are irrelevant in such case. This algorithm also is the irregular mesh version of the sample integration formula 3.

3.2 Vertex equilibrium equation

If G has cycles, however, the system of equations (4) is overdetermined. In such case, it is almost impossible that there is a tension free solution; that is, it is impossible to satisfy all the edge equations at same time. Given the assumption of independent Gaussian measurement errors in the edges, the Bayesian analysis says that the most probable set of values z is the solution by least squares of the system (4); that is, the values z that minimize the *quadratic discrepancy function*

$$Q(z) = \sum_{e \in \mathcal{E}\, G} w[e] \left(z[\mathrm{DST}(e)] - z[\mathrm{ORG}(e)] - d[e] \right)^2 \tag{5}$$

If we apply the path integration algorithm in this case, we will obtain an approximate solution which depends on the choise of the initially chosen vertex.

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

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

We can write the equation (6) as

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

where

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

In another words, the equilibrium occurs when $z[u]$ is the weighted average of $z[v] - d[u,v]$ of all neighbours v of u, where each neighbour 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. The equation (7) can also be writen 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 $w[u,v]/w_t[u]$ is the *relative weight* of v between the neighbours u.

3.3 Physical analysis

The following mechanical analogy can help to understand the problem: each vertex v is modelled as a horizontal plate without weight which is free to move vertically, but cannot move horizontally or rotate. The value $z[v]$ is the height value of said plate. Each edge $e = (u,v)$ is modelled as a ideal vertical spring with rigidity coefficient $w[e]$, connected to the plates u and v such that it apply a vertical force with magnitude $w[e](z[v] - z[u] - d[e])$; that is, the spring tries to force the distance $z[v] - z[u]$ to be equal to $d[e]$. The value $z[v]$ which minimizes $Q(z)$ is a situation of mechanical equilibrium, where the total force between each plate is zero. This can be seen, since 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 the $Q(z)$ is zero, that is, if each spring length is equal to its relaxed length.

Another analogy for this mathematical problem is an electric 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 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. Equation (6) becomes then the Kirchoff law, which is satisfied when the circuit is electrical equilibrium and the total current entering and exiting each circuit is zero.

3.4 Matrix form

In order to express the problem in a 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 in arbitrary order, which has only one directed version of each non directed edge. Let also,

- $\mathbf{A}_{m \times n}$ be an incidence matrix of G such that $\mathbf{A}_{kj}$ is $+1$ if the vertex v_i is the destination of the edge e_k, -1 if v_i is the source of e_k, and zero otherwise;
- $\mathbf{W}$ be a diagonal matrix $m \times m$ 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 $k \in \{1, \ldots, n\}$. The functional Q can be then expressed as the matrix product

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

Where $^{\mathrm{T}}$ denotes a matrix transposing operator. 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

$$\mathbf{M}\mathbf{z} = \mathbf{b} \tag{11}$$

where

$$\mathbf{M} = \mathbf{A}^{\mathrm{T}}\mathbf{W}\mathbf{A} \tag{12}$$

and

$$\mathbf{b} = \mathbf{A}\mathbf{W}\mathbf{d} \tag{13}$$

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

4 CONVERTING A DISCRETIZED GRADIENT GRID TO A WEIGHTED MESH

In this section we describe the conversion of the gradient data from a uniform rectangular grid to the graph format.

4.1 Discretizing Z

In the Poisson-based approaches, both sides of equation (1) are derived, giving us the equation

$$\mathbf{L}(Z)(x,y) = \mathbf{H}(F,G)(x,y) \tag{14}$$

where

$$\mathbf{L}(Z) = \frac{\partial^2 Z}{\partial x^2} + \frac{\partial^2 Z}{\partial y^2} \qquad \mathbf{H}(F,G) = \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} \tag{15}$$

In their discrete versions, the operators $\mathbf{L}$ and $\mathbf{H}$ of the equations (14) are replaced by the finite difference estimators $\mathcal{L}$ and $\mathcal{H}$ (referred in the literature often as "Poisson kernels").

In order to discretize the equation (14), the function Z is represented by a *Z map*, a matrix $z[u,v]$ placed in a *sampling point grid* $q[u,v]$. To obtain compatible estimates of the derivatives from both sides of the equation (1), we suppose that the sample points $q[u,v]$ are shifted by half-pixel in relation to the gradient sampling points $p[u,v]$ in each axis. Specifically we assume that $p[u,v]$ is the point $(u+1/2, v+1/2) \in \mathbb{R}^2$, whether $q[u,v]$ is the point (u,v). This convention is illustrated in the figure 4.

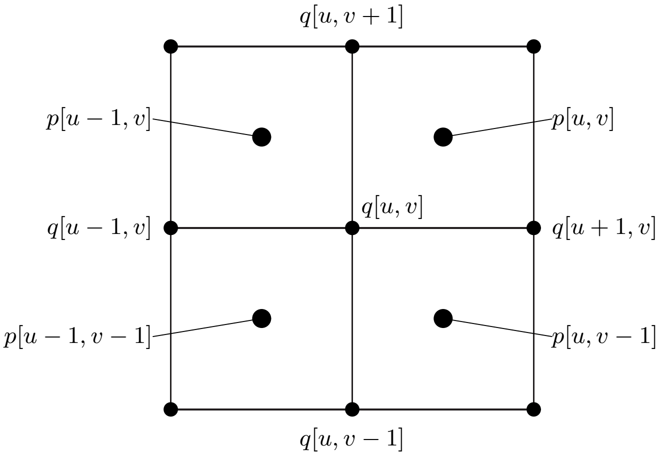

Fig. 4. Gradient and Z sampling points adjacent to the point $q[u,v] = (u,v)$.

Therefore, if the derivative maps f, g have n_x columns and n_y lines, we can assume that the domain D is the rectangle $[0,n_x] \times [0,n_y] \in \mathbb{R}^2$. Each pixel $[u,v]$ from the derivative maps can be identified with the unit side square centred on $p[u,v]$ with $q[u,v]$ and $q[u+1,v+1]$ in opposite corners. Note that the map z, on the hand, has n_x+1 columns and n_y+1 lines, and we can suppose that each pixel $z[u,v]$ is a unit side square centred on the point $q[u,v]$, which is a vertex of the gradient grid.

In practice, the derivative maps $f[u,v]$ and $g[u,v]$ are almost always the average of the derivatives $\partial Z/\partial x$ and $\partial Z/\partial y$ in the neighbourhood of the point $p[u,v]$, computed by a *gradient sampling kernel*, which should be symmetrical relative to to $p[u,v]$ and overlap partially the neighbour kernels. In similar way, the computed $z[u,v]$ will be an estimate to the average Z around the point $q[u,v]$, obtained by another *sampling kernel*. The relationship between those two kernels is outside of the scope of this paper.

We assume that the weight map is given in the form of a w matrix of non-negative values, with the same dimensions as f and g and in similar way, they are related to the points $p[u,v]$. We assume that $w[u,v]$ is zero always when the points $p[u,v]$ is outside the domain D. 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.

In order to improve the legibility of the formulas, we will use the notation $z_{\circ\circ}$ for a given sample $z[u,v]$, and the following notations for its four neighbours:

$$\begin{array}{ll} z_{-\circ} = z[u-1,v] & z_{\circ -} = z[u,v-1] \\ z_{+\circ} = z[u+1,v] & z_{\circ +} = z[u,v+1] \end{array}$$

We will use also the following symbols for the interpolated derivative values from our algorithm, from the maps f and g, in the average edge points between $q[u,v]$ and its four neighbours:

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

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

We will denote $w_{-\circ}$, $w_{+\circ}$, $w_{\circ-}$, and $w_{\circ+}$ *edge reliably weights*, assigned by our algorithm to the interpolated slopes $f_{-\circ}$, $f_{+\circ}$,$g_{\circ-}$ and $g_{\circ+}$, respectively. See figure 5.

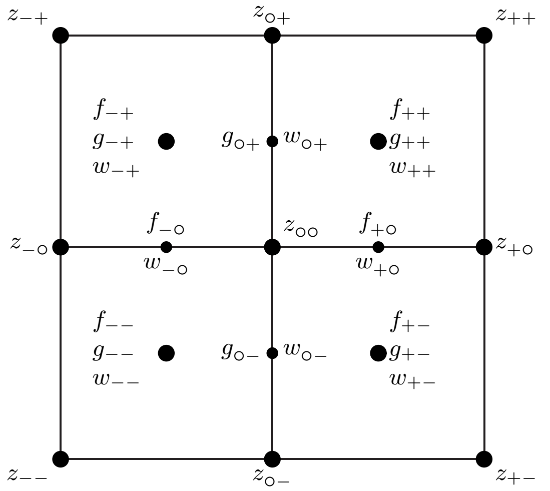

Fig. 5. Notation for the interpolated Z, gradient and weight values around the point $q[u,v]$.

In the unweighted form (that is, all the weights are 1) , we can use the discrete operators $\tilde{\mathcal{L}}(z)$ and $\tilde{\mathcal{H}}(f,g)$ to estimate $\mathbf{L}(Z)$ and $\mathbf{H}(F,G)$, where

$$\begin{aligned}\tilde{\mathcal{L}}(z)[u,v] = &+(z_{\circ+}-z_{\circ\circ})-(z_{\circ\circ}-z_{\circ-})\\ &+(z_{+\circ}-z_{\circ\circ})-(z_{\circ\circ}-z_{-\circ})\end{aligned} \tag{16}$$

$$\tilde{\mathcal{H}}(f,g)[u,v] = f_{+\circ}-f_{-\circ}+g_{\circ+}-g_{\circ-} \tag{17}$$

The discrete version of the equation (14) is then the set of linear equations

$$\tilde{\mathcal{L}}(z)[u,v] = \tilde{\mathcal{H}}(f,g)[u,v] \tag{18}$$

for each $[u,v]$ in the domain z.

Note that the first term $(z_{\circ+}-z_{\circ\circ})$ in the formula (16) is another estimate for the derivative $\partial Z/\partial x$ in the average point $(u+\frac{1}{2},v)$ of the horizontal edge in the grid between $q[u,v]$ e $q[u+1,v]$, computed from the map z. This can be compared with the first term $f_{+\circ}$ from the equation (17) computed by the interpolation of the given map f. Similarly, the other terms in the formula (16) are the estimates of the numerical derivatives of Z, which can be compared with the terms $f_{-\circ}$, $g_{\circ-}$, e $g_{\circ+}$ of the formula (17). Inspired in this observation, we divided each equation (18) in the four separated equations:

$$\begin{array}{ll} z_{+\circ}-z_{\circ\circ}=+f_{+\circ} & z_{\circ+}-z_{\circ\circ}=+g_{\circ+}\\ z_{-\circ}-z_{\circ\circ}=-f_{-\circ} & z_{\circ-}-z_{\circ\circ}=-g_{\circ-}\end{array} \tag{19}$$

In general, the system (19) obtained in this way is overdetermined, since it has approximately $2N$ equations (after eliminate repeated equations) with approximately N unknowns. We seek the approximate solution of this system by the least squares criteria; that is, a map z which minimizes the weighted sum of the square of differences between both sides of the equation. In this sum, we use for each equation the weight ($w_{-\circ}$, $w_{+\circ}$, $w_{\circ-}$) or $w_{\circ+}$ of the interpolated edge data. In this way, the least squares solution of the equations (19) are reduced to the solution of another system of N linear equations with N unknowns. Each equation of this system assert that each value $z[u,v]$ is equal to the weighted height of its four neighbours, increased by the interpolated derivatives in the middle of its corresponding edges:

$$\begin{aligned} z_{\circ\circ} = &+\frac{w_{-\circ}}{W_{\circ\circ}}(z_{-\circ}+f_{-\circ})+\frac{w_{+\circ}}{W_{\circ\circ}}(z_{+\circ}-f_{+\circ})\\ &+\frac{w_{\circ-}}{W_{\circ\circ}}(z_{\circ-}+g_{\circ-})+\frac{w_{\circ+}}{W_{\circ\circ}}(z_{\circ+}-g_{\circ+})\end{aligned} \tag{20}$$

In this formula, $W_{\circ\circ}$ is the *total vertex weight*,

$$W_{\circ\circ}=W[u,v]=w_{\circ-}+w_{\circ+}+w_{-\circ}+w_{+\circ} \tag{21}$$

Note that only the relative values of the weights are meaningful. Rearranging the equation (20) to separate the unknown and known terms we obtain

$$-\mathcal{L}(z)[u,v] = +z_{\circ\circ}-\frac{w_{\circ-}}{W_{\circ\circ}}z_{\circ-}-\frac{w_{\circ+}}{W_{\circ\circ}}z_{\circ+}-\frac{w_{-\circ}}{W_{\circ\circ}}z_{-\circ}-\frac{w_{+\circ}}{W_{\circ\circ}}z_{+\circ} \tag{22}$$

and

$$-\mathcal{H}(f,g)[u,v] = -\frac{w_{\circ-}}{W_{\circ\circ}}g_{\circ-}+\frac{W_{\circ+}}{w_{\circ\circ}}g_{\circ+}-\frac{w_{-\circ}}{W_{\circ\circ}}f_{-\circ}+\frac{W_{+\circ}}{w_{\circ\circ}}f_{+\circ} \tag{23}$$

To compute the right side of the equation (23) we need estimates of the derivatives for the mid point of each edge of the grid. For example, we need the estimate $g_{\circ+}$ for the derivative $\partial Z/\partial y$ in the mid point of the edge $r=(u,v+\frac{1}{2})$ between $q[u,v]$ e $q[u,v+1]$. In order to obtain such estimate we use the four values $g_{\mathrm{a}}=g[u-2,v]$, $g_{\mathrm{b}}=g[u-1,v]$, $g_{\mathrm{c}}=g[u,v]$ and $g_{\mathrm{d}}=g[u+1,v]$, which are the derivatives sampled 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 distances (signed) of those points to r are $-\frac{3}{2}$, $-\frac{1}{2}$, $+\frac{1}{2}$ and $+\frac{3}{2}$ respectively. We will denote by w_{a}, w_{b}, w_{c} and w_{d} the reliability weights of said values. See figure 6.

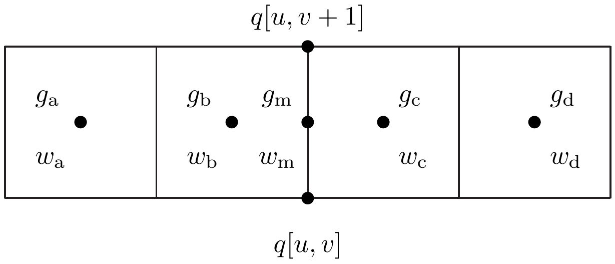

Fig. 6. Interpolating the derivative $\partial Z/\partial y$ in the point $(u,v+\frac{1}{2})$.

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

$$\begin{aligned} g_{-} &= (3g_{\mathrm{b}}-g_{\mathrm{a}})/2\\ g_{\circ} &= (g_{\mathrm{b}}+g_{\mathrm{c}})/2\\ g_{+} &= (3g_{\mathrm{c}}-g_{\mathrm{d}})/2\end{aligned} \tag{24}$$

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

$$\begin{aligned} w_- &= 4/(9/w_\mathrm{b} + 1/w_\mathrm{a}) \\ w_\circ &= 4/(1/w_\mathrm{b} + 1/w_\mathrm{c}) \\ w_+ &= 4/(9/w_\mathrm{c} + 1/w_\mathrm{d}) \end{aligned} \tag{25}$$

We then take the weighted average g_m of those three estimates and assign the appropriated weight w_m:

$$\begin{aligned} g_\mathrm{m} &= \frac{w_- g_- + w_\circ g_\circ + w_+ g_+}{w_- + w_\circ + w_+} \\ w_\mathrm{m} &= w_- + w_\circ + w_+ \end{aligned} \tag{26}$$

As said previously, all the samples which are situated outside the domain g receive zero weight, such that their values are ignored in the computation. Note that any estimate g_-, $g_\circ$ or g_+ which depends on a sample with zero weight will have zero weight itself, thus it wont contribute to the final result. In particular, if one or more of the weights are zero, formulas 24 and 27 get simplified as follows:

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

Case	Weighted average g_m	Weight w_m
$w_\mathrm{a} = 0$	$\dfrac{w_\circ g_\circ + w_+ g_+}{w_\circ + w_+}$	$w_\circ + w_+$
$w_\mathrm{b} = 0$	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 symmetrical to the above. It would not be safe expand the list (24) with interpolations of non-consecutive values such as $(g_\mathrm{a} + 3g_\mathrm{c})/4$. If the intermediary samples (g_b in this case) have low or zero weight, it can be a region of discontinuity and the computed value will be invalid. If all the weights w_-, $w_\circ$ and w_+ are zero, the final weight w_m will be zero by the formula (26). In this case, we define arbitrarily $g_\mathrm{m} = g_\circ$. We will denote the functions (24–26) by

$$(g_\mathrm{m}, w_\mathrm{m}) \leftarrow \textsc{Interpolate}(g_\mathrm{a}, w_\mathrm{a}, g_\mathrm{b}, w_\mathrm{b}, g_\mathrm{c}, w_\mathrm{c}, g_\mathrm{d}, w_\mathrm{d}) \tag{27}$$

In order to obtain $f_{+\circ}$, the estimated value of $\partial Z/\partial x$ in the mid point $s = (u + \frac{1}{2}, v)$ of an horizontal edge, we use this same function INTERPOLATE in the samples vertically adjacent to the edge, 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. This same function is used to estimate $g_{\circ -}$ and $f_{-\circ}$.

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 adding 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 then is solved recursively for the mesh G', generating an estimate z' for its vertices. By interpolation of such heights we obtain a initial guess z for the original mesh G. The heights z are then adjusted by the iterative Gauss-Seidel method. The recursion is interrupted when the graph z is reduced to a single vertex v to which we can assign a zero value z.

In other words, we build a 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.

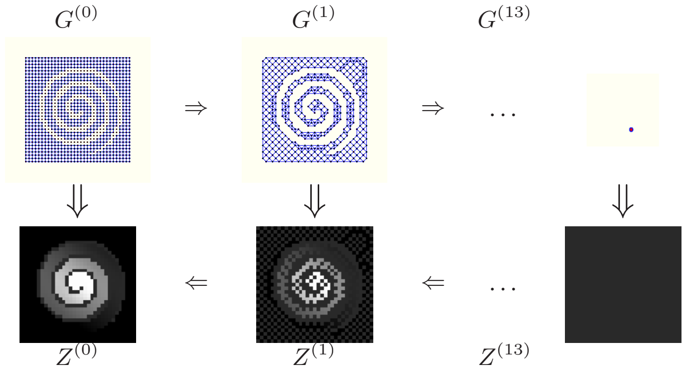

Fig. 7. Multi-scale mesh integration.

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

```
Procedure MSMeshIntegrate$(G, q, \varepsilon)$
  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$(z, G, q, \varepsilon)$;
  9. Return $z$.
```

Fig. 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' also simple, planar and connected. Since the procedure DECIMATE always returns a connected mesh, the connectivity and planarity of the original mesh is also preserved at all levels of the multi-scale pyramid. The algorithm MSMESHINTEGRATE is therefore immune to problems caused by premature loss of connectivity and works even if the original mesh has a region connected to the rest only by a single path.

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 son-adjacent) vertices with maximum degree 6. To achieve this goal, the procedure use an attribute *mark* for each vertice, which can have 3 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 neighbours 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 vertice u is removed, the edges connected to u are also removed. If u is of degree 1, no aditional action is needed. If u has degree greater or equal than 2, new edges are added to G', connecting the neighbours of u. Note that all those vertices are in K and therefore they will be always vertices of G'. The extremities, weights and differences of the new edges are chosen such that the solution $z'[v]$ for the mesh G' is close to the solution $z[v]$ for each vertice $v \in K$.

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 counterclockwise ordering, and let $v_0, v_1, \ldots, v_{k-1}$ the corresponding destination vertices. Let w_i the weight of the edge e_i and d_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 added an edge $e'_{i,j}$ from v_i to v_j

$$d'_{ij} = d_j - d_i \qquad w'_{ij} = \frac{w_i w_j}{w_t} \tag{28}$$

where w_t is the sum of weights $w_0, w_1, \ldots, w_{k-1}$. This operation — the removal of u, every edge e_i connected to u and the creation of the edges e'_{ij} between the neighbours of u — we denote *star-clique swap* . In particular, if the vertex u has degree $k = 2$, the swap will only add a new pair edges e'_{01} e e'_{10}. If the degree k is 3, only the edges e_{01}, e'_{12}, e'_{02} and their reverses will be added. In both cases the planarity of the edge is preserved.

However, if k is greater or equal than 4, the addition of every $k(k-1)$ edge $e'_{i,j}$ would make G' non-planar and therefore it would interfere severely with the algorithm efficiency. Therefore when $k \geq 4$, we add only the edges $e'_{i,i+1}$ which connects the successive vertices v_i e v_{i+1} for $i \in \{0, 1, \ldots, k-1\}$ in a cycle, as well its reverse edges (where the indices are computed by module k). We will denote this operation as *star-cycle swap*. The differences $d'_{i,i+1}$ between such edges are given by the formula (28), that is:

$$d'_{i,i+1} = d_{i+1} - d_i \tag{29}$$

In other hand, the weights $w'_{i,i+1}$ are given by distinct formulas, according with the degree k, given in the table 2.

TABLE 2
Formulas for the weight w'_{01} if 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 each other edge $e'_{i,i+1}$, except by the indices which are increased by i module k.

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

In contrast with the star-clique swap, the star-cycle swap do not ensure that the values determined by the mesh G' are exactly the same as the determined by the mesh G. In fact, it is not possible to retrieve an exact equivalence adding only a subset of edges e'_{ij} of the clique, if the weights and differences are computed local formulas (that is, dependent only on the edges e_i). It does not seem possible to compute the proper values of d_i and w_i without analyzing the entire G mesh and thus solving the integration problem.

However, our experiments showed that 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 errors between z' and the derived solution z are highly localized, and can be removed with a few Gauss-Seidel iterations.

The star-cycle swap can create parallel edges, which can be either new edges added by star-cycle swaps on distinct R vertices or edges from 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 replaces every group of 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 substituted by a single edge e with the following attributes

$$w[e] = w[e'] + w[e''] \qquad d[e] = \frac{w[e']d[e'] + w[e']d[e']}{w[e'] + w[e'']} \tag{30}$$

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

5.2 Interpolation

Once the solution z' is obtained for the reduced mesh G' (step 6), it is converted into a 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 neighbour $v \in G[u]$ belongs to K, and therefore its height $z[u]$ is already determined.

5.3 Iterative adjustment

The initial estimate z computed by INTERPOLATE is used as initial guess for the procedureSOLVESYSTEM (step 8) used by the Gauss-Seidel algorithm. 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 neighbours. The procedure ends after a specific number of iterations κ, or when the variation between a value $z[u]$ from a iteration to another is smaller than a given tolerance ε, for each vertice u, whichever happens first.

Note that for 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 Algorithm Analysis

Correctness The star-cycle swap and the collapse of parallel edges preserves the planarity and connectivity properties of the graph, such that in all recursive calls of MSMESHINTEGRATE the mesh G satisfies such conditions. Additionally, the vertex equations (7) are dominated by the diagonal, thus the application of the algorithm at the scale 0 with proper values of κ and ε converges 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 the number of Gauss-Seidel iterations needed in each level. Let $N = \#\,\mathcal{V}\,G, N_k = \#\,\mathcal{V}\,G^{(k)}$, $M = \#\,\mathcal{E}\,G, M_k = \#\,\mathcal{E}\,G^{(k)}$. Let $\hat{\beta}$ the largest value of β observed in all mesh reduction steps. If $\hat{\beta} < 1$ then the maximum scale m will be $\log_{1/\hat{\beta}} N = O(\log N)$ and the total N_{tot} of vertices of all meshes will be at maximum $N/(1-\hat{\beta}) = O(N)$.

The planarity of the mesh G ensures that the upper limit for $\hat{\beta}$ is smaller than 1. Since G is a simple planar graph, it is known that $M \leq 6N$ and G has at maximum $N/7$ vertices with degree smaller or equal than 6. The same conclusions are valid to $G^{(k)}$, $N^{(k)}$ and $M^{(k)}$. From those facts, it is possible conclude that the theoretical limit is $\hat{\beta} \leq 41/42 \approx 0.976$ [27]. Therefore M is at maximum $20 \log_2 N$ and N_{tot} is at maximum $42N$. In practice, however, the reduction factor β is approximately 0.6 in most cases, 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 which represents each mesh $G^{(k)}$. A simple representation which suffices for our purposes consists in a *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 one *index table* with N_k entries, which stores for each vertex v, the index, the first edge in the edge table with source v. The total storage space is therefore at maximum $N_k + 2 \times 3M_k \leq 19N_k$ words per mesh $G^{(k)}$, and at maximum $19N_{\text{tot}} \approx 47.5N$ words for every mesh in the pyramid.

The planarity condition also ensures that the decimation algorithm finishes in time $O(N_k + M_k) = O(N_k)$ for each level k. Therefore all the pyramids 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.* [9], 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 suppresed, while the low frequency components have their wavelength halved. Thus, the solution $z^{(k+1)}$ recursively computed, after expanded to a previous scale $z^{(k)}$ will be correct mostly in the low frequency components with small details missing. Those missing components 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" in terms of graph structure, which is not covered by the scope of this work. However, the experimental tests demonstrate that convergence is achieved after few iterations, even with instances where other multi-scale methods fail.

6 TESTS

In this section we compare the cost and precision of our topological multi-grid integrator (MG) with other published methods. We consider only methods which are able to deal with discontinuities and missing data, such as the Poisson-based integrators presented by Agrawal *et al.* [19] such as the M-Estimators(AM) and Diffusive Affine Transform(AT), the multi-scale grid integrator of Saracchini *et al.* [9] and the iterative methods presented by Queáu *et al.* [23], namely the Isotropic Total Variation (DT) and L^1 Functional (DL). We opted to not test the integrator proposed by Breuß *et al.* [26] since its source code was not available in the moment of writing this paper, as well the fact that it is part of an initial step for a Poisson or Krylov-based integrator, which could be incorporated as part of the other iterative solvers.

The tested integrators are summarized in table 3.

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 grid [9]
MG	Weighted multi-scale on mesh

The algorithms provided by Agrawal and Durou were implemented by the authors in Matlab. We modified Agrawal's algorithms to use the given weights, instead of trying to estimate them from the f and g matrices. Our algorithms were implemented in C, compiled with the Gnu C compiler. Our test platform had a Intel I5-3470 at 3.2 Ghz with 16GB of RAM memory. The Matlab codes were run in Matlab in Windows 7 whether C programs were run in Linux Debian 7.0. In order to avoid inconsistent results and avoid the in-built paralellization of Matlab when computing the running times, we set the processor affinity of each process to be bound to only one core of the CPU.

6.1 Test datasets

We used 6 test datasets. Four of them (`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 modelling [29]. The `dtbust` dataset was obtained from a live subject by the 3DMT scanner [30]. 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. Particularly, `dtbust` has weakly connected regions due occlusions and small sections with missing data. For all models, the weight mask was created by hand with an image editor, as a grayscale image where the cliffs and undefined regions were marked in black over a white background. See figure 9 and 10.

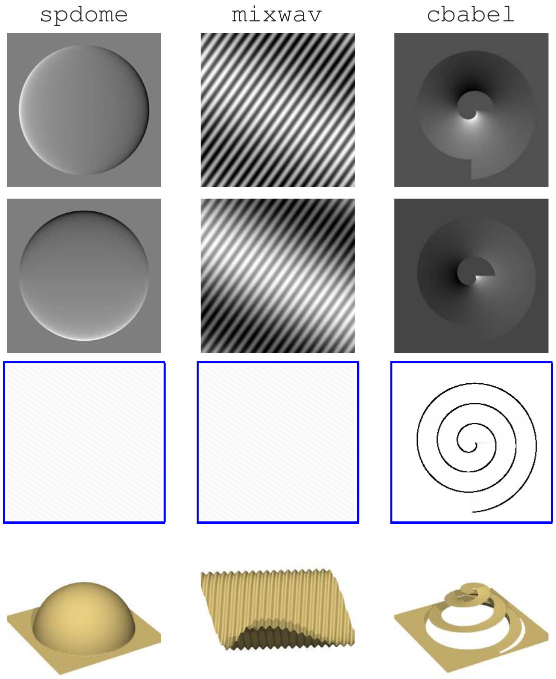

Fig. 9. The test datasets `spdome`, `mixwav` and `cbabel`, showing the gradient maps f (top row) and g (second row), the weight maps w (third row), and the correct z height map (bottom row).

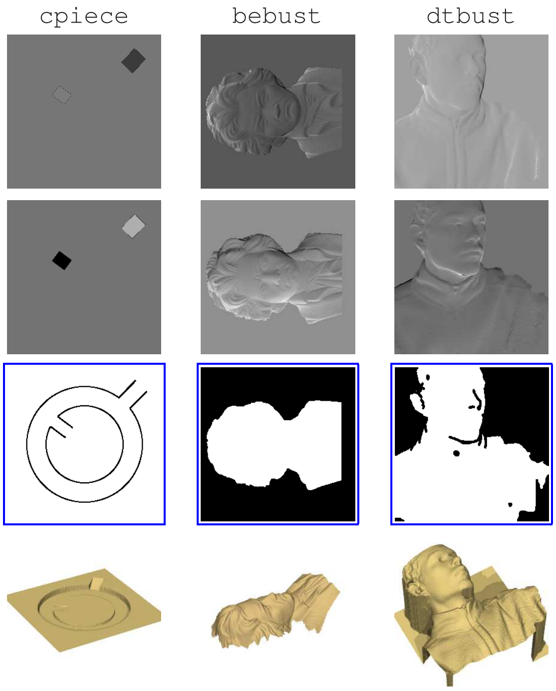

Fig. 10. Test datasets `cpiece`, `bebust` and `dtbust`.

We processed each dataset as given, and also after perturbing the 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 comparing them with the ground-truth height map. The maximum number of iterations κ was set respectively at 50 and 20 for scale 0 for the `MS` and `MG` integrator whether the `DT` and `DL` were set with their default number of iterations and additional parameters (respectively 50 and 500 iterations). The absolute accuracy of each computed map was quantified as the weighted standard deviation of the difference e between it and the correct height map. The relative accuracy was quantified as e/R, where R is the weighted standard deviation of the correct map. These values are summarized in tables 4–5.

TABLE 4
Absolute and relative root-mean-square errors of each method for the test datasets, without noise.

	`spdome`		`mixwav`		`cbabel`	
Method	e	e/R	e	e/R	e	e/R
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 of each method for the test datasets, with 30% of Gaussian noise added.

	`spdome`		`mixwav`		`cbabel`	
Method	e	e/R	e	e/R	e	e/R
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 graph-based multiscale method described in this article (`MG`) is more accurate than the other five methods tested. Note also that our previous uniform-grid multiscale method (`MS`) fails on the datasets `cbabel`, `cpiece`, and `dtbust` because of loss of connectivity at the smallest levels of the pyramid. The iterative methods `DT` and `DL` had comparable accuracy with the direct-solving methods for most cases, although they had issues with the weakly connected datasets `cbabel` and `cpiece` even after reaching convergence. All the integrators showed to be rather resilient to the presence of noise, and the main factor affecting the results was the connectivity of the height map. The results of the most significant failure cases are shown in figure 11–13.

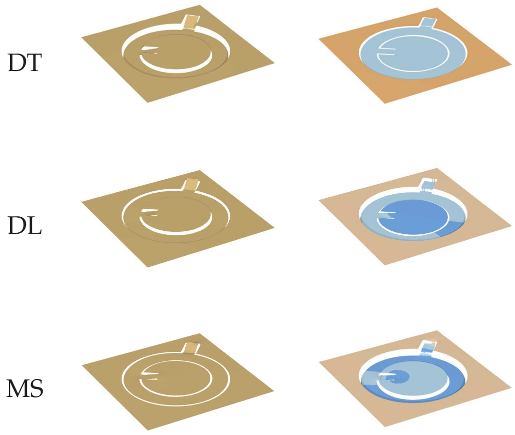

Fig. 11. Failure case example: the height maps computed by algorithms `DL`, `DT`, and `MS` on the dataset `cpiece` without noise. Blue and orange hues on the error maps show that the computed height was below or above the correct height, respectively.

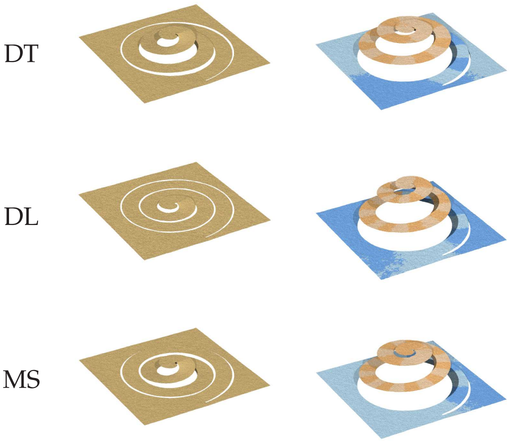

Fig. 12. Failure cases of the dataset `cbabel` with 30% of Gaussian nose.

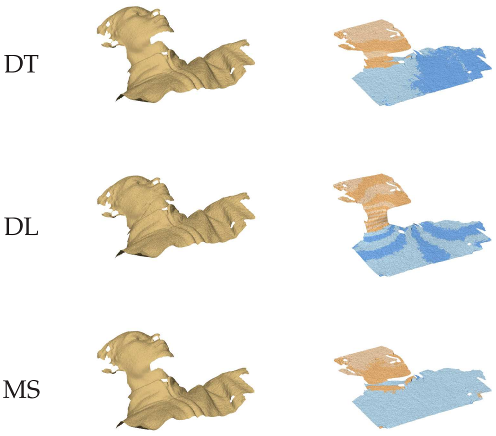

Fig. 13. Failure cases of the dataset `dtbust` with 30% of Gaussian nose.

6.3 Computational costs

In order to evaluate the efficiency and scalability of our algorithm, we measured the running time and the amount of memory used in the integration of two datasets (`spdome` and `dtbust`). The gradient data was sampled on uniform grids of various sizes, from 64×64 to 2048×2048. For the single-scale methods of Agrawal and Durou (`AT`, `AM`, `DT`, and `DL`) we considered only the the time needed to solve the linear system, since most of pre-processing stages have linear computational cost which may be higher than solve the system itself. Due the nature of iterative algorithms, we set a fixed number of iterations: two iterations for `DT` and `DL` and 50 and 20 iterations for `MS` and `MG`. See figures 14 and table 6.

For the multi-scale methods, we added the computational time for each set of iterations performed in all scales. The absolute costs cannot be compared due substantial differences of programming language and libraries. Therefore, our focus is how the computing costs scale with the size of the problem. The results show that the running time grows like $O(N)$ for both multi-scale methods (`MS` and `MG`) and for `DL`. All other methods had their computational costs growing as $O(N^{1.5})$, mostly due their reliance in a Cholesky sparse solver in later stages.

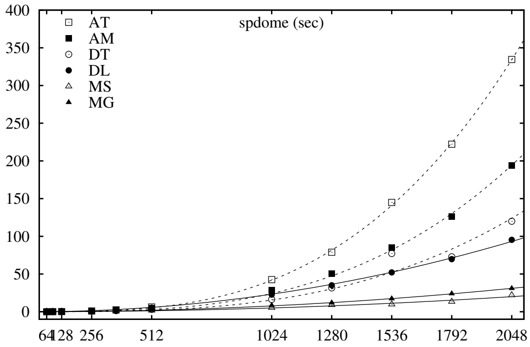

Fig. 14. Plots of the system solving time for the four single-scale methods (`AT`, `AM`, `DT`, and `DL`), the uniform grid multi-scale method (`MS`) and the graph multiscale method of this article (`MG`), for the `spdome` dataset. Note that the plots are in function of $n = N^2$ in order to aid the visualization. The fitted curves are $\alpha n^2 = \alpha N$ (full) and $\alpha n^3 = \alpha N^{1.5}$ (dotted) with the best-fit coefficient α.

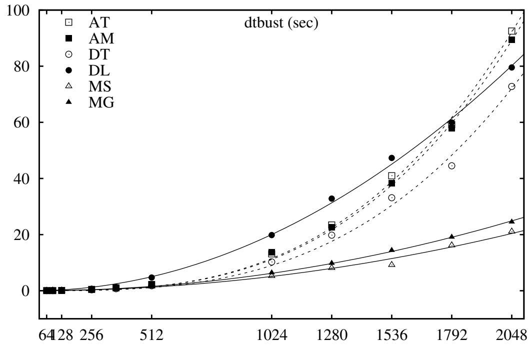

Fig. 15. Plots of the system solving time for the `dtbust` dataset.

TABLE 6
Time costs for the six methods on the `spdome` and `dtbust` datasets.

`spdome`						
	`time`					
N	AT	AM	DT	DL	MS	MG
64×64	0.04	0.04	0.05	0.07	0.02	0.02
90×90	0.07	0.09	0.11	0.11	0.04	0.04
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
360×360	2.50	1.77	1.44	1.66	0.82	0.99
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
1280×1280	78.89	50.82	31.63	35.08	8.96	11.86
1536×1536	144.97	85.11	77.19	52.18	9.74	17.30
1792×1792	222.17	126.49	72.89	69.73	13.25	23.90
2048×2048	351.04	194.06	119.93	95.32	21.92	30.88

`dtbust`						
	`time`					
N	AT	AM	DT	DL	MS	MG
64×64	0.04	0.06	0.04	0.07	0.02	0.01
90×90	0.05	0.07	0.06	0.09	0.04	0.03
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
360×360	1.03	1.01	0.74	1.33	0.79	0.80
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
1280×1280	23.41	22.59	19.80	32.82	8.12	9.80
1536×1536	40.98	38.34	33.12	47.31	9.19	14.38
1792×1792	59.17	57.98	44.50	59.93	16.18	19.12
2048×2048	92.51	89.46	72.78	79.54	21.10	25.03

7 CONCLUSIONS

The graph-based multiscale method `MG` described in this article is asymptotically faster than state-of-the-art single-scale methods, and extremely as accurate. Although slower than an a uniform-grid multiscale method `MS`, it can cope with poorly connected datasets that cause `MS` to fail. The method has linear growth either in memory as running times and can be easily parallelised to SIMD processing platforms (such as GPUs and FPUs) due the simplicity of its data representation. The algorithm currently was tested only with the the Gauss-Seidel iterative solver, which has inherent slow convergence. Although it needs very few (in order of 10 or 20) iterations in the lower level of the multi-scale pyramid to obtain acceptable results, it needs a significant number of iterations to obtain marginal improvements regarding errors in a given scale. In a nearby future we expect to make use of approaches to speed-up the computation of initial guess, still keeping the linear memory and time asymptotic costs, reducing even further the number of needed iterations. One approach which may have such potential is the Fast-Marching approach proposed by Breuß *et al*. [26].

ACKNOWLEDGMENTS

We thank Jean-Denis Durou, Yvain Quéau and Amit Agrawal, who kindly provided the code of their integrators, test datasets for our tests as well their inspiring work in the field. We also thank also the support given from CnPQ, CAPES, FAPESP, EPSRC funding agencies and the Technological Institute of Castilla y León during the development of this algorithm and writing of its manuscripts.

REFERENCES

[1] B. K. P. Horn and M. J. Brooks, *Shape from Shading*. Cambridge, Mass.: MIT Press, 1989.
[2] B. K. P. Horn, “Height and gradient from shading,” *Intl. Journal of Computer Vision*, vol. 5, no. 1, pp. 37–75, 1990.
[3] B. K. P. Horn, R. J. Woodham, and W. M. Silver, “Determining shape and reflectance using multiple images,” MIT Artificial Inteligence Laboratory, Tech. Rep. AI Memo 490, 1978.
[4] R. J. Woodham, “Photometric method for determining surface orientation from multiple images,” *Optical Engineering*, vol. 19, no. 1, pp. 139–144, 1980.
[5] M. L. Smith and L. N. Smith, *Polished Stone Surface Inspection using Machine Vision*. OSNET, 2004, p. 33.
[6] M. Kampel and R. Sablatnig, “3D puzzling of archeological fragments,” in *Proc. of 9th Computer Vision Winter Workshop*, D. Skocaj, Ed., 2004, pp. 31–40.
[7] J. Sun, M. L. Smith, A. R. Farooq, and L. N. Smith, “Concealed object perception and recognition using a photometric stereo strategy,” in *Proc. 11th Intl. Conf. on Advanced Concepts for Intelligent Vision Systems (ACIVS)*, vol. 5807, 2009, pp. 445–455.
[8] M. F. Hansen, G. A. Atkinson, L. N. Smith, and M. L. Smith, “3d face reconstructions from photometric stereo using near infrared and visible light,” *Computer Vision and Image Understanding*, vol. In Press, 2010.
[9] R. F. Saracchini, J. Stolfi, H. C. Leitão, G. A. Atkinson, and M. L. Smith, “A Robust Multi-Scale Integration Method to Obtain the Depth from Gradient Maps,” *Computer Vision and Image Understanding*, vol. 116, no. 8, p. 882–895, Aug. 2012.
[10] J.-D. Durou, Y. Quéau, and J.-F. Aujol, “Normal integration–part i: A survey,” 2016.
[11] Y. Quéau, J.-D. Durou, and J.-F. Aujol, “Normal integration–part ii: New insights,” 2016.
[12] Z. Wu and L. Li, “A line-integration based method for depth recovery from surface normals,” *Computer Vision, Graphics and Image Processing*, vol. 43, no. 1, pp. 53–66, 1988.

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

[14] R. Fraile and 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, and 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 and R. Chellappa, “A method for enforcing integrability in shape from shading algorithms,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 10, no. 4, pp. 439–451, 1988.

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

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

[19] A. Agrawal, R. Raskar, and 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, and 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, and 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*, vol. 23, pp. 643–660, 2001.

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

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

[24] D. Terzopoulos, “Image analysis using multigrid relaxation methods,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 2, pp. 129–139, Mar. 1986.

[25] ——, “The computation of visible-surface representations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 4, pp. 417–438, 1988.

[26] M. Breuß, Y. Quéau, M. Bähr, and 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*, vol. 12, pp. 28–35, 1983.

[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, 2006, see [19].

[29] Université de Tolouse, “Codes for photometric stereo,” Dataset at http://ubee.enseeiht.fr/photometricstereo/ , accessed on 2016-06-22, 2010.

[30] “3dmdface system,” Company page at http://www.3dmd.com/3dmdface.html, accessed on 2010-04-22, 2010.