STAG: Smart Tools and Applications in Graphics (2018)
M. Livesu, G. Pintore and A. Signoroni (Editors)

Gradient Field Estimation on Triangle Meshes

Claudio Mancinelli[1] Marco Livesu[2] Enrico Puppo[1]

[1]University of Genoa - DIBRIS, Genoa, Italy
[2]CNR IMATI, Genoa, Italy

Abstract

The estimation of the differential properties of a function sampled at the vertices of a discrete domain is at the basis of many applied sciences. In this paper, we focus on the computation of function gradients on triangle meshes. We study one face-based method (the standard the facto), plus three vertex based methods. Comparisons regard accuracy, ability to perform on different domain discretizations, and efficiency. We performed extensive tests and provide an in-depth analysis of our results. Besides some behaviour that is common to all methods, in our study we found that, considering both accuracy and efficiency, some methods are preferable to others. This directly translates to useful suggestions for the implementation of gradient estimators in research and industrial code.

CCS Concepts

•Mathematics of computing → ***Numerical differentiation; •Computing methodologies*** → ***Mesh models; •Human-centered computing*** → *Scientific visualization;*

1. Introduction

Geometric meshes play a central role in a number of domains, including computer graphics and scientific visualization, mechanical, structural and electrical engineering, and computational methods in physics, chemistry and geology. Quite often, a scalar field is sampled at the vertices of a geometric mesh, which discretizes a given domain. Computational analysis of such a field may require evaluating its gradient and, possibly, tracing its integral lines.

In the geometry processing and FEM literature, a scalar field is often extended from vertices to the interior of higher dimensional cells by linear interpolation. Under this approach, a constant gradient is associated to each cell with a straightforward computation, thus providing the simplest form of evaluation of the gradient field. Although sufficient for many applications, the piece-wise constant gradient has several limitations and drawbacks: being not continuous, the gradient has divergent covariant derivative at the interface between cells; singularities are forced to lie just at vertices of the mesh; and integral lines are discretized into polylines that travel parallel to each other inside each element, so that their distribution does not depend just on the field, but also on the orientation of edges and vertex valences of the underlying mesh (see Figure 1).

As discussed in [DGDT16], discrete vector fields can be given per face, per edge, or per vertex. While per-edge and per-face vector fields are usually not continuous, per-vertex vector fields can be extended to the whole mesh by linear interpolation, and they can be traced exactly inside each element [KRG03, NJ99]. However, there exist surprisingly few works dealing with the estimation and assessment of per-vertex gradient fields.

In this work, we review common methods for estimating a per-vertex gradient field, and we compare their accuracy with respect to

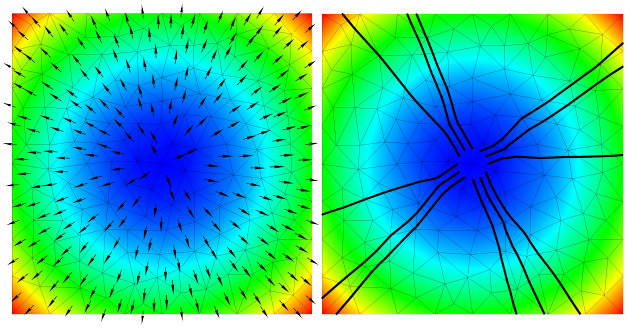

Figure 1: *Per-face gradient estimation on a paraboloid with minimum at the center of the domain (left). Although the estimate is very accurate, integral lines traced from a neighborhood of the minimum do not diverge uniformly, but rather travel in four bundles of nearly parallel lines (right). This behaviour is induced by the discrete piece-wise constant nature of the gradient field. Starting the tracing from the umbrella of a vertex with low valence exacerbates the problem.*

the standard method to compute a per-face gradient field. Since our interest is to study the performances of the piece-wise linear methods in every point of the domain, such comparisons will be made considering not only the vertices of the mesh, as shown in Sec. 4.3. This choice is further motivated by the expectation of applying these results to the tracing of integral curves. We limit our study to the planar case, and plan to extend it to manifold surfaces and volumes in future works. It is worth observing that, in order to apply piece-wise linear methods on a generic triangular mesh immersed in a 3D space, a way to *parallel transport* vectors from a tangent space to another will be needed (see Sec 6.2). Performances of the various methods are measured according to a family of parametric functions sampled at mesh vertices. The influence of the underlying tessellation is also studied, considering different domain discretizations strategies.

The analysis of our results shows that – not surprisingly – piecewise constant gradients perform worse than gradients linearly interpolated within each triangle in almost all our experiments. The only exception being performances on boundaries, where computing the gradient separately for each triangle offers an advantage to methods that use a bigger stencil to evaluate the function. Restricting to piece-wise linear approaches, we found that in terms of accuracy all methods performed similarly in many experiments, with some exception regarding resiliency to anisotropy and signal noise. We broke ties combining accuracy with a deep analysis of the scalability and computational cost associated to each technique, being able to provide useful suggestions for the implementation of per-vertex gradients in research and industrial code.

2. Related Works

Sozer and colleagues [SBK14] surveyed different approaches to gradient estimation for a variety of meshes used in Computational Fluid Dynamics (CFD), ranging from tetrahedral and hexahedral, to general polyhedral meshes. In all the considered examples, the function is encoded at the vertices of the mesh and the average per-cell gradient is computed according to different techniques, always leading to a piece-wise constant gradient field. The recent course on field processing [DGDT16] discusses gradient fields as a special case of the more general vector fields; the course focuses on triangular meshes embedded in $\mathbb{R}^3$, and does not extend to other spaces or discretizations. In [HS97] Hyman and Shashkov introduced a framework for discrete calculus of fields defined on the vertices of a 2D domain meshed with quads, formulating a discrete counterpart of continuous operators such as gradient, divergence and curl. Neumann and colleagues [NCKG00] propose an efficient method to estimate surface normals of a density field encoded in a voxel grid, and use this information for shading. The method is based on a gradient estimation obtained with 4D linear regression. Jirka and Skala [JS02] studied gradient estimation based on fitting quadrics on mesh vertices; the regression strategy we study is based on the method they describe. Correa et al. [CHM09] studied the accuracy of gradient estimation in the context of scientific visualization. Their study is similar in spirit to ours (similar gradient estimation techniques are considered), but is focused on volumetric meshes only. For the 2D case, Cerbato et al. [CHdSM14] tested the performances of the Green Gauss gradient estimation on a variety of different polygonal meshes (simplicial meshes were not considered). To the best of our knowledge no comprehensive study of gradient field estimation on 2D simplicial meshes is present in literature.

3. Methods for gradient field estimation

We introduce here the four methods we consider in our study, also fixing our notation. Let $\Omega \subset \mathbb{R}^d$ be a compact domain. While in the experimental part we concentrate on the case $d = 2$, we present theory in a more general way, referring to d dimensional simplicial meshes and sometimes providing explicit formulas also for the case $d = 3$, which corresponds to the case of tetrahedral meshes and is ubiquitous in applied sciences. Nevertheless, concepts extend to any dimension.

Let Σ be a simplicial mesh having Ω as carrier. Mesh Σ can be described by the collection V of its vertices, and the collection T of its maximal cells – which are either triangles or tetrahedra, for $d = 2$ and $d = 3$, respectively. Cells of intermediate dimensions, such as edges and faces, can be derived uniquely by subsets of vertices belonging to the same maximal cell. We assume the standard topological relations among adjacent and incident cells. In this context, we recall that the *star* (or *1-ring*) of a vertex $v \in V$ is defined as the collection of all cells (edges, triangles) of Σ that are incident at v. By abuse of notation we will refer to vertices in the star of v by meaning the vertices sharing an edge with v. The star provides a discrete version of a mathematical *neighborhood*, and will be generically denoted by $\mathcal{N}(v)$. The *k-ring*, for $k > 1$, can be defined inductively as the union of the stars of all vertices in the $(k-1)$-ring.

Let $f : \Omega \longrightarrow \mathbb{R}$ be a smooth scalar function. We assume to know the value of f only at the vertices of Σ. The discrete version of f is therefore a collection $F = \{f_1, \ldots, f_n\}$, where n is the number of vertices in V and each f_i corresponds to the function value sampled at vertex v_i, for all $i = 1, \ldots, n$.

The problem addressed in the following is that of estimating the gradient ∇f on Ω, on the basis of the discretizations F and Σ. The gradient will be estimated at cells of Σ and extended to the whole domain via en either piece-wise constant (Section 3.1), or piecewise linear (Sections 3.2.1, 3.2.2 and 3.2.3) approaches.

3.1. Per-Cell linear Estimation (PCE)

Our bottom line is a method that estimates a constant gradient at each maximal cell. We start by extending the values in F to the cells of Σ by linear interpolation. This is well defined on cells of all orders because Σ is a simplicial mesh, hence there exists a unique linear function that interpolates the values in F at all vertices of any cell $\sigma \in \Sigma$, namely

$$\tilde{f}_\sigma(p) = \sum_{v_i \in \sigma} \lambda_i f_i,$$

where p is a generic point of σ, the v_i's are the vertices of cell σ and the λ_i's are the barycentric coordinates of p with respect to the v_i's. In this model, function $\tilde{f}$ estimates f as a piecewise-linear function, which is continuous over Ω and differentiable only in the interior of its maximal cells. The gradient of $\tilde{f}$ is thus constant inside every cell σ and it is associated either to the whole σ, or conventionally to

its centroid c_σ, depending on the applications. For a triangle t with vertices v_i, v_j, v_k it is easy to show that we have

$$\nabla f_t = (f_j - f_i)\frac{(v_i - v_k)^\perp}{2A_t} + (f_k - f_i)\frac{(v_j - v_i)^\perp}{2A_t}. \quad (1)$$

Analogously, for a tetrahedron τ with vertices v_i, v_j, v_k, v_h we have

$$\begin{aligned}\nabla f_\tau = (f_j - f_i)&\frac{(v_i - v_k)\times(v_h - v_k)}{2V_\tau} \\ + (f_k - f_i)&\frac{(v_i - v_h)\times(v_j - v_h)}{2V_\tau} \\ + (f_h - f_i)&\frac{(v_k - v_i)\times(v_j - v_i)}{2V_\tau},\end{aligned}$$

where $e^\perp$ denotes edge e rotated by 90°, and A_t, V_τ are the area of t and the volume of τ, respectively.

3.2. Per-vertex gradient estimation

As already observed, in the piece-wise linear model of $\tilde{f}$ the gradient is not defined at vertices of Σ. The methods we review in the following assume that f is a higher order function, smooth at edges and vertices of Σ. The different methods exploit different facts that hold in the continuous case, and try to bring them to the discrete setting. All such methods work either by averaging (integration) or by approximation (fitting), because no exact model can be assumed for f in the generic case.

Note that, once the gradient has been estimated at all vertices, the gradient field can be extended by linear interpolation inside cells of any order. It is therefore continuous in Ω and overall more accurate than the piece-wise constant field reviewed in the previous section, as we will see in our experiments.

3.2.1. Average Gradient on Star (AGS)

A common procedure in discrete differential geometry consists of estimating a differential property at a point p as the average value of the same property in a neighborhood of p [MDSB03]. More formally, in our case, we can write

$$\nabla f(p) \simeq \frac{1}{V_{B(p)}}\int_{B(p)} \nabla f dV,$$

where $B(p)$ is a neighborhood of p and $V_{B(p)}$ is its volume/area.

Now, given a vertex v of Σ, we can use the method in the previous section to estimate (an average value of) ∇f in the maximal cells of the star of v, and compute the integral as a sum of constant terms, obtaining

$$\nabla f_v \simeq \frac{1}{\sum_{\sigma\in\mathcal{N}(v)} V_\sigma}\sum_{\sigma\in\mathcal{N}(v)} V_\sigma \nabla f_\sigma, \quad (2)$$

where ∇f_σ is the value computed with Equation 1. Note that, since the gradient is constant inside each maximal cell, we obtain the same result if we consider any area of integration that gives equivalent weights to the cells in the star. For instance, Equation 2 holds unchanged if we consider the centroidal integration area as defined in [MDSB03], which partitions the mesh into disjoint integration areas for the different vertices.

3.2.2. Least Squares fit of Directional Derivatives (LSDD)

This approach consists in estimating first a few directional derivatives of f at v_i, and imposing their relation with the gradient. Let be $\{v_0, ..., v_{k_i}\}$ the vertices belonging to the 1-ring of v_i. Taylor's expansion of f at the first order allows us to write:

$$f(v_j) - f(v_i) \approx \nabla f \cdot (v_i - v_j),$$

for every $j = 0, ..., k_i$. The idea is to build a linear system exploiting the above approximation, i.e writing

$$f(v_j) - f(v_i) = \nabla f \cdot (v_i - v_j), \qquad j = 0, ..., k_i. \quad (3)$$

Note that the second term of (3) is the directional derivative of f along vector $(v_i - v_j)$, hence the name. Since k_i usually greater than the dimension of the space, the linear system is usually overdetermined and it only admits a least squares solution. Let A_i be the $k_i \times d$ matrix obtained by collecting all the $(v_j - v_i)$, and let D_i be the column matrix consisting of all the $f(v_j) - f(v_i)$. Then, the system can be written in matrix form as $A_i \nabla f(v_i) = D_i$, and its least squares solution is obtained by resolving the $d \times d$ linear system

$$A_i^T A_i \nabla f(v_i) = A_i^T D_i. \quad (4)$$

Note that we have to assemble and solve such a system at every vertex, hence this method is usually more expensive than the previous ones.

3.2.3. Linear regression (LR)

The last approach we review consists of approximating function f in the neighborhood of v_i with a polynomial π_i of given degree, by setting a system of linear equations that asks π_i to assume the given values of F at all vertices of a given k-ring of v_i. After the fitting polynomial has been obtained, the gradient of f at v_i is estimated analytically as the gradient of π_i.

In our experiments we consider quadratic polynomials and 1-rings, which are extended to 2-rings only if the number of neighbors of v_i is insufficient to fix all degrees of freedom. In 2D we have

$$\pi_i(x, y) = a_i x^2 + b_i y^2 + c_i xy + d_i x + e_i y + f_i,$$

where coefficients $P_i^T = [a_i, b_i, c_i, d_i, e_i, f_i]$ are unknown. For each vertex v_j in the neighborhood of v_i (including v_i itself), we impose $\pi_i(v_j) = f_j$, thus obtaining a linear system with as many equations as the vertices in the neighborhood of v_i. The coefficients of the best fitting polynomial can therefore be found by solving the system in the least squares sense, accoring to the normal equation

$$A_i^T A_i P_i = A_i^T F_i, \quad (5)$$

where:

- A_i is a $k_i \times 6$ matrix containing one row per vertex in the neighborhood of v_i (including v_i itself); the row corresponding to vertex $v_j = (x_j, y_j)$ contains values $(x_j^2, y_j^2, x_j y_j, x_j, y_j, 1)$;
- F_i is a column vector containing the values f_j corresponding to the vertices v_j in the neighborhood of v_i.

Once the coefficients of π_i are known, the gradient at $v_i = (x_i, y_i)$ is given trivially by

$$\nabla f(v_i) = (2a_i x_i + c_i y_i + d_i, 2b_i y_i + c_i x_i + e_i).$$

For $d = 3$ the solution is analogous: the polynomial has 10 unknown coefficients and the method needs solving a 10×10 linear system. Note that we have to assemble and solve such a system at every vertex, hence this method is the most expensive in the set we review.

4. Experimental setup

We evaluate the four techniques presented in Section 3 on analytic functions, comparing numerical estimates with the ground truth. In order to analyze different situations, we use a parametric family of non-polynomial periodic functions, and meshes with different characteristics.

4.1. Test functions

We consider the domain $\Omega = [0,1] \times [0,1]$ and the following parametric family of functions:

$$f_{a,b}(x,y) = a \sin(bx) \cos(by).$$

Parameters a and b control the amplitude and the frequency of the function, respectively. Figure 2 shows four plots of $f_{a,b}$ for different values of a and b.

4.2. Meshes

We aim to test the performances of the various gradients on different discrete settings, where the domain is tessellated according to different strategies. To this end, we select three meshes, which we believe are representative of ubiquitous scenarios in applied sciences. Specifically, we consider:

- Σ_S: a *structured* mesh made of equilateral triangles;
- Σ_U: an *unstructured* mesh obtained computing a Constrained Delaunay Tessellation of a random sampling of the domain Ω;
- Σ_A: an *anisotropic* mesh obtained by computing a CDT of a sampling of the $[0,K] \times [0,1]$ domain, and squeezing it to fit the unit square. Value K is called the coefficient of anisotropy.

For Σ_U and Σ_A we used Triangle [She96] for mesh generation. Each mesh contains approximately 1K triangles. Closeups of our meshes are shown in Figure 3. In our experiments, we do not investigate progressively finer meshes; we rather decided to keep the tessellation fixed and to act on the frequency of the function (i.e., parameter b). Note that the two approaches are equivalent.

4.3. Error metrics

We evaluate the Mean Squared Error (MSE) of the estimated gradient by sampling data on a regular grid of 100×100 points. The error of a gradient field has two components: angular and magnitude. In our experiments when possible we merge them, considering a composite metric defined as the norm of the difference vector between the analytic gradient of the continuous function ∇f, and its numerical estimation $\tilde{\nabla} F$ (computed on the discrete sampling F of f at the vertices of the mesh)

$$MSE_{tot} = \frac{1}{N} \sum_{i=0}^{N} \left\| \nabla f(v_i) - \tilde{\nabla} F(v_i) \right\|_2^2,$$

In some applications, only the directional or magnitude components of the gradient are relevant. Therefore, sometimes we plot only the angle error, defined as

$$MSE_{ang} = \frac{1}{N} \sum_{i=0}^{N} \measuredangle(\nabla f(v_i), \tilde{\nabla} F(v_i))^2,$$

or only the magnitude error, defined as

$$MSE_{mag} = \frac{1}{N} \sum_{i=0}^{N} (\|\nabla f(v_i)\| - \|\tilde{\nabla} F(v_i)\|)^2,$$

In general, we plot the latter two errors only in cases where they show significantly different behaviours (e.g., Figure 6). In all such cases, we explicitly talk about angle or magnitude error. If nothing is said, we always assume the total error is considered.

5. Evaluation

We report here our experiments and analysis. We consider the four different gradient estimators described before:

- **PCE:** Per-Cell Estimator (Section 3.1);
- **AGS:** Average Gradient on Star (Section 3.2.1);
- **LSDD:** Least Squares Directional Derivatives (Section 3.2.2);
- **LR:** Linear Regression (Section 3.2.3).

Piece-wise linear gradients (i.e., AGS, LSDD and LR) are linearly interpolated inside each cell using barycentric coordinates. We test their performances according to various instances of our parametric test function (Section 5.2), also considering different domain discretizations (Section 5.3). Since in real applications data coming from sensors are often affected by noise, we also check performances of gradient estimators under perturbed functions (Section 5.4).

Experiments were run on a MacBook Pro equipped with an Intel i5 with 2.7 GHz and 8 GB of RAM. We wrote a C++ single threaded application, implementing the four methods described in Section 3. We used CinoLib [Liv17] for geometry processing and Eigen [GJ*10] to solve linear systems.

5.1. Boundaries

In our experiments we observed that all vertex-based methods exhibit the same pathological behavior at the boundaries of the domain, where approximation error is averagely higher than in the interior (Figure 5). To understand this increased error we must remember that all such methods use a stencil (i.e., the 1-*ring*) to sample the function in a local neighborhood and estimate the gradient. For boundary vertices, the stencil is partially outside of the domain, resulting in an *unbalanced* sampling that leads to wrong estimates (Figure 4). Note that the performances decay as the stencil becomes bigger, and estimation errors propagate towards the interior. For example, using the 2-*ring*, all the boundary vertices and the vertices adjacent to them will have inaccurate gradients. This is empirically confirmed by the performance of LR, which is the worst among all the methods we tested. In fact, most of the times LR uses a bigger stencil (the 2-*ring*) at boundary vertices, because such vertices do not have enough neighbors in their one-ring, to fix all degrees of freedom in Equation 5.

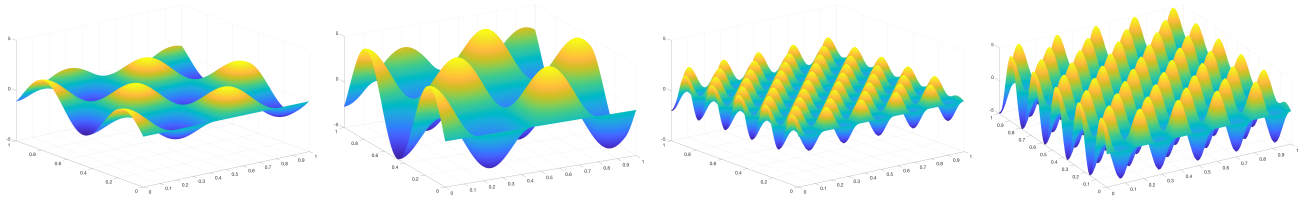

Figure 2: *Four examples of our test function $f_{a,b}$. Parameter a controls the amplitude, whereas parameter b controls the frequency. From left to right: $f_{2,10}$, $f_{5,10}$, $f_{2,30}$, $f_{5,30}$.*

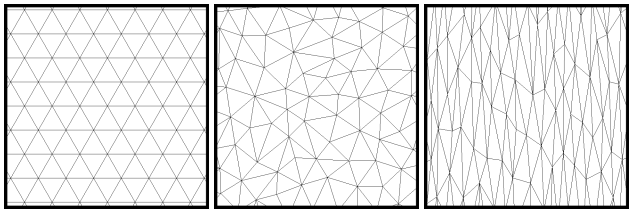

Figure 3: *Close up of our test meshes. From left to right: structured, unstructured and anisotropic ($K = 5$).*

Piece-wise constant gradient (PCE) is insensitive to boundaries, because gradients are computed on each triangle separately, by using only data at the three triangle vertices. This is confirmed by the graphs in (Figure 4).

To avoid the bias introduced by boundaries, in the experiments we present in the remainder of the paper we ignore boundaries. In a sense, we act as if around the domain we had a padding layer with size compatible with the local stencil used by numerical algorithms to estimate gradients.

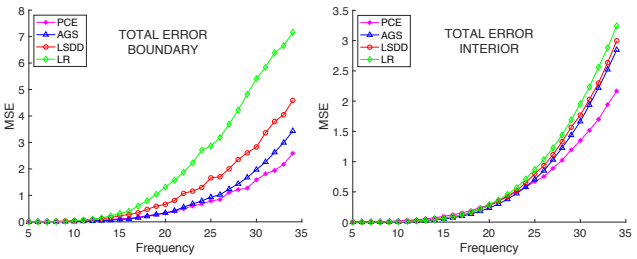

Figure 4: *Error plots obtained computing the gradient of our test function $f_{a,b}$ with fixed amplitude and growing frequency. We show separate plots for boundary (left) and interior (right). Note that for vertex-based methods the approximation error is averagely higher at the boundaries.*

5.2. Function parameters

Here we report performances of the four gradient estimators when tested on different instances of our test function $f_{a,b}$. As stated in Section 4.1 parameters a and b set the amplitude and frequency of the function, respectively.

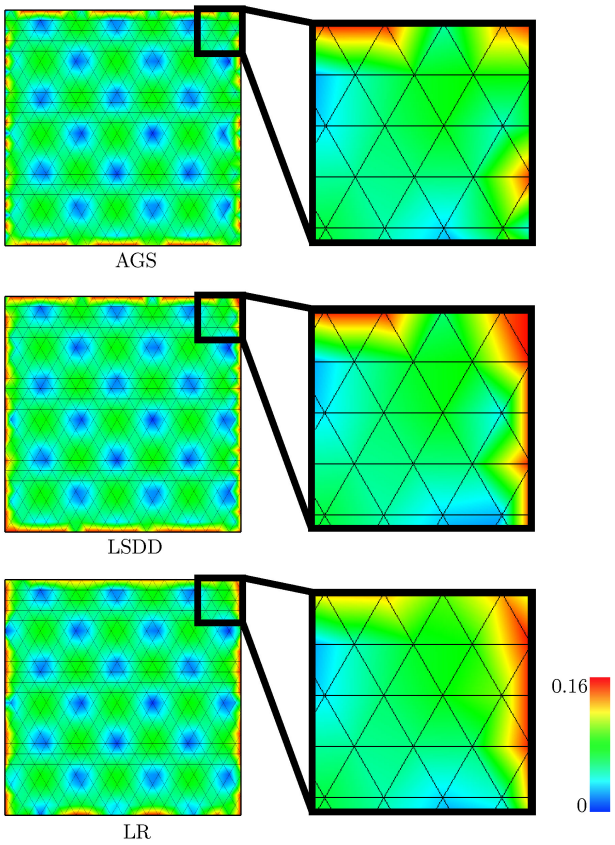

Figure 5: *Color coded errors for the three vertex-based gradient estimators. All of them exhibit the same pathological behavior at the boundary of the domain, where the error increases (see close-ups).*

Amplitude. We tested different values of parameter a, ranging from 0.1 to 2. As it can be noticed from Figure 2, changes in amplitude affect the magnitude of the gradient (the bells become steeper), leaving unchanged its direction (maxima and minima do not change). This is confirmed by our experiments, where we see that angle error is constant for varying a (Figure 6, left), whereas magnitude error grows linearly for increasing frequencies (Figure 6, right).

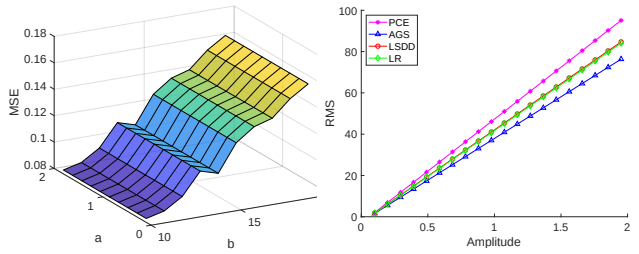

Figure 6: *Left: plot of angle error, obtained varying both the function amplitude (a) and frequency (b). Right: plot of the RMSE (Root Mean Squared Error) of the magnitude for fixed frequency and varying amplitude. When varying amplitude, the gradient keeps its orientation and changes magnitude. Therefore, for growing values of a, angle error remains constant and magnitude error grows linearly.*

Frequency. We investigate the response of the considered gradient estimators for different frequencies by varying the parameter b in the range $[5, 35]$. Differently from the amplitude, when varying frequency both the angular and magnitude components of the gradient change. For this experiment we considered structured and unstructured meshes. Overall, we noticed a coherent asymptotic behavior for all methods on both meshes, with higher error values for the unstructured case.

Considering angle error (Figure 7, top line), we observe that PCE is the least accurate. This is not surprising, as PCE does not catch the non-linear variation of the function within each face, due to its piece-wise constant nature (Figure 8). Among piece-wise linear approaches, the Linear Regression (LR) performs better at low frequencies, but above a certain frequency its error grows faster and it starts to perform worse than the others. As it can be noticed from the error plots, the value above which performance starts to decay at faster rate is approximately the same for both structured and unstructured meshes (see the dashed vertical lines). We believe this relates to the ratio between edge lengths and the period of the function. Basically, at a certain point the mesh density starts to be too coarse for the period of the function, and fitting a quadric in the one ring becomes counter-productive as the function oscillates too much within it. This is an inherent limitation of all fitting methods, which depends on the degrees of both the approximated and fitting functions. Overall, LR could provide the best estimate when the mesh is quite refined with respect to signal frequency, while AGS exhibits the best performance in terms of resilience to increase in signal frequency. LSDD exhibits a behavior similar to AGS, but with a consistently higher error.

Magnitude error offers an opposite landscape, where piece-wise linear methods perform worse than PCE for growing frequency values of $f_{a,b}$ (Figure 7, bottom line). Looking at data, we noticed that vertex-based methods tend to smoothly distribute the error in the whole domain, without positive or negative peaks. Conversely, the magnitude error of PCE tends to be overall lower, with spikes in some isolated spots. We do not have a clear explanation for this. Our current guess is that this is another effect of the mismatch between mesh density and function period. Essentially all the piecewise linear methods evaluate the gradient at a vertex by considering its whole one ring, whereas PCE is more local, as it evaluates the gradient for each triangle separately. When the mesh is too coarse with respect to the function, the higher locality of PCE allows to offer a better estimate of the extent to which the function locally varies.

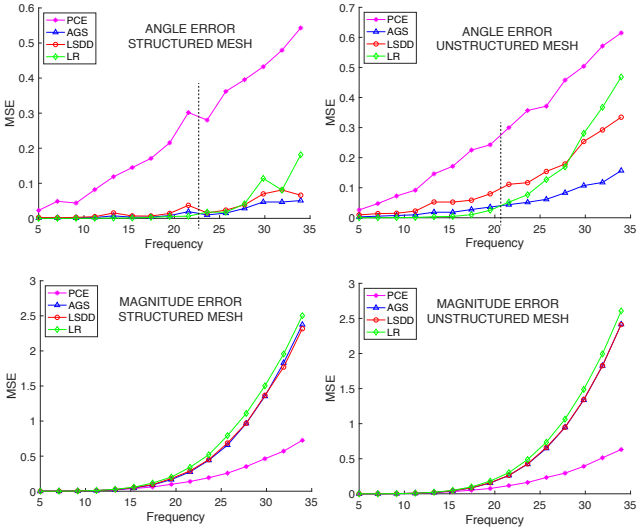

Figure 7: *Plot of angle error (top line) and magnitude error (bottom line), obtained varying the function frequency on the structured (left column) and unstructured mesh (right column). Angle-wise there are two interesting things to notice: (i) PCE performs worse than others due to its piece-wise constant nature, which does not catch field drift within each element; (ii) LR performs best for frequencies below the vertical dashed line, and starts to diverge for higher frequencies because the mesh becomes too coarse to catch function fluctuations. Magnitude-wise, PCE performs consistently better than piece-wise linear methods. We believe this is because it is more local, and therefore less sensitive to increases in the function period.*

5.3. Anisotropy

We test here the performances of the four gradient estimators with respect to meshes exposing increasing levels of anisotropy (Figure 3, right). We report both angle and magnitude errors in Figure 9. As for the frequency tests, there is a neat separation between piecewise constant and piece-wise linear methods, which are superior at any level of anisotropicity of the mesh. Among piecewise-linear methods, linear regression (LR) is remarkably better than the others (Figure 10). Directional derivatives (LSDD) seem to suffer the poor edge directions spanned by the anisotropic neighborhood and perform worse than the other two. AGS stays in between.

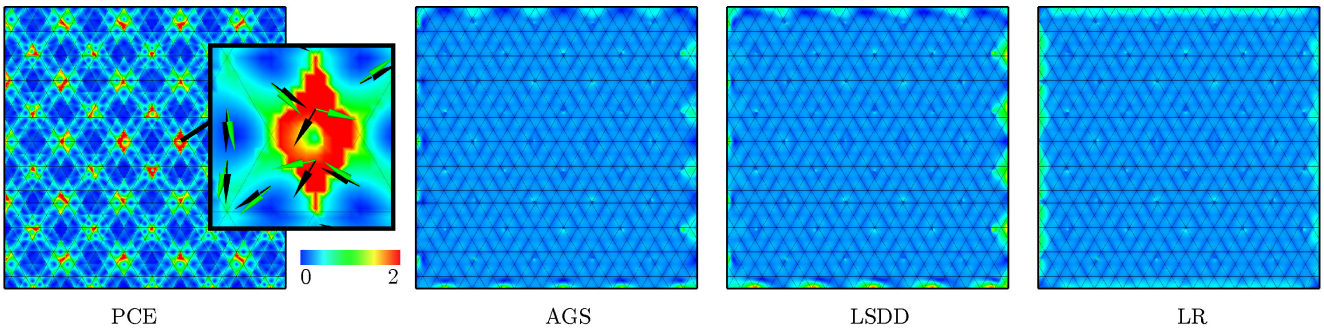

Figure 8: *Plot of total error of function $f_{0.1,15}$, measured on a structured mesh with PCE, AGS, LSDD and LR. Notice that piece-wise linear methods (AGS, LSDD, LR) outperform PCE. In the closeup, visual comparison between the ground truth gradient (green arrows) and the gradient numerically estimated with PCE (black arrows).*

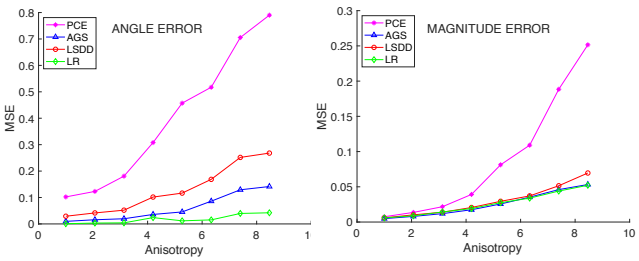

Figure 9: *Plot of angle error (left) and magnitude error (right) of the four methods computed on unstructured meshes with increasing levels of anisotropy.*

5.4. Noise

In many experiments involving real data (e.g. acquired with sensors), the signal measured at vertices is affected by noise. We test here the resiliency of all methods with respect to perturbations of the test function at the mesh vertices. In Figure 11, we report both angle and magnitude errors for growing values of noise (we plot the M value on the x axis). Again, we observe a clear separation between PCE and the piece-wise linear methods, with PCE exhibiting a super-linear increase the the magnitude error. The piece-wise linear methods exhibit similar behavior, resulting more sensitive to noise in terms of angle error and more resilient in terms of magnitude error.

5.5. Computational cost

Here we discuss the four tested methods with respect to their computational cost. To this end, there is a major difference between PCE, AGS and LSDD, LR. Given a mesh with $|T|$ triangles and $|V|$ vertices, PCE can be efficiently packed into a $d|T|\times|V|$ matrix G. Multiplying G for a column vector containing the function values at each vertex in the mesh, a $d|T|$ long column vector containing the serialized per triangle gradient can be efficiently computed by matrix vector multiplication. The same goes for AGS, where the matrix G will have different size ($d|V|\times|V|$), and multiplying it with a scalar field in vector form one will obtain a $d|V|$ vector containing the serialized per vertex gradient. This results in an extremely fast computation, where the matrix can be constructed once and used to evaluate the gradient of several fields (e.g. when values vary over time). Moreover, having the gradient in matrix form is useful to define a discrete divergence operator, which is nothing but the matrix G^T, which transforms gradients into per vertex divergence values by means of matrix vector multiplication [Liv18].

Differently from PCE and AGS, LSDD and LR cannot be pre-computed and stored in a matrix. They need to be computed from scratch each time, without saving data from previous computations. In particular, LSDD requires solving a $d \times d$ linear system at each vertex. Note that while solving a 2×2 system for each vertex of a 2D mesh is affordable, the method intrinsically suffers from the curse of dimensionality, and does not scale well on high dimensional data (e.g. for data mining or machine learning applications).

Similarly to LSDD, also LR requires solving a linear system at each vertex and does not allow for pre-computation. However, in this case the system is even bigger. As explained in Section 3.2.3, fitting a quadric in 2D requires solving a 6×6 linear system of unknowns (the quadric coefficients). In 3D the number of coefficient increases, leading to a 10×10 linear solve per vertex. Note that, regardless the degree of the polynomial being fit, also this method suffers from the curse of dimensionality, as the number of unknown coefficients grows combinatorially with the size of the space. The good news are that for low size manifolds sitting on a higher dimensional space, the size of the problem grows with the intrinsic size of the data, and not with the extrinsic size of the embedding.

In Figure 12 we plot computation times of the four methods we tested for meshes with growing size. Notice that methods that can be expressed in matrix form (PCE and AGS) are less expensive than LSDD and LR, which require solving a linear system at each vertex. The amortized versions of PCE and AGS do not account for the time necessary to build the matrix. AGS seems more expensive because in our current implementation averaging around each vertex is not plugged inside the matrix but rather computed afterwards navigating the mesh topology. Encoding the whole computation within the matrix should produce results similar to amortized

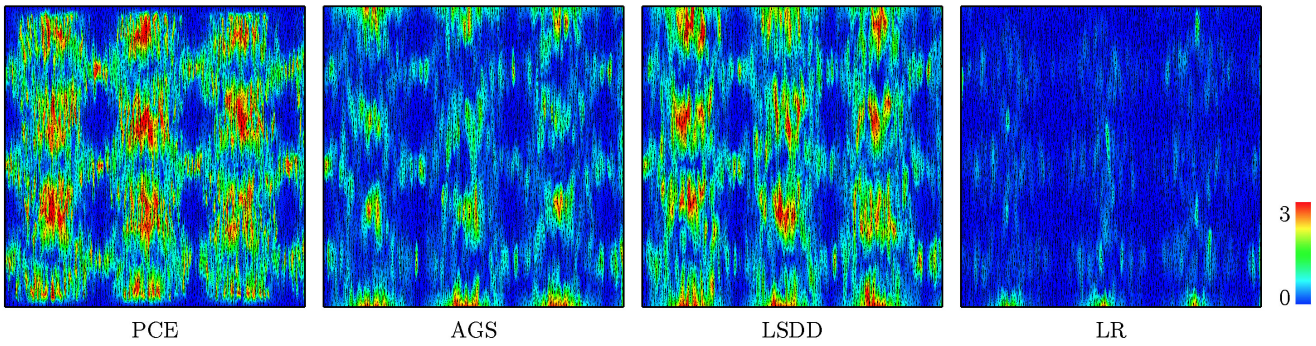

Figure 10: *Plot of total error of the four methods, computed on an unstructured mesh with anisotropy factor $K = 9$.*

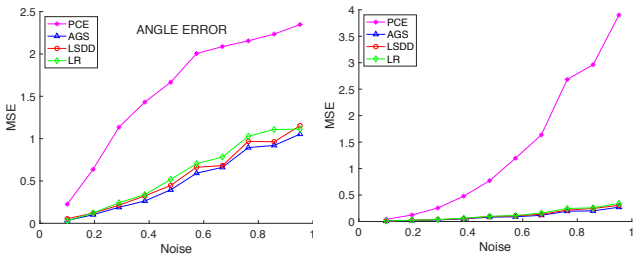

Figure 11: *Plot of angle error (left) and magnitude error (right) of the four methods computed on a structured mesh, where the function sampled at its vertices was affected by increasingly high displacement to simulate noisy data. There are no significant differences between the piece-wise linear methods. The piece-wise constant method performs worse than all.*

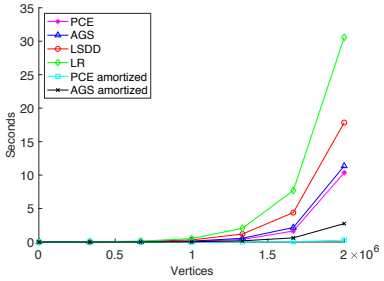

Figure 12: *Computational cost associated to each gradient estimator. The amortized curves refer to the cost of matrix vector multiplication only (the matrix is assumed to be pre-computed). Our current implementation of AGS is not optimal, as gradients are evaluated per triangle using the matrix, and then averaged at vertices navigating the mesh topology. Encoding the whole computation in matrix form the performances of AGS amortized should match the ones on PCE amortized.*

PCE (the matrix is a bit bigger and denser, but we expect a negligible increase in time for this).

6. Extensions

The scope of our analysis is limited to the evaluation of a gradient field on simplicial meshes covering a Euclidean domain. There are several important extensions of this problem that we briefly review in the following.

6.1. Polygonal/polyhedral meshes

Linear interpolation of function f cannot be used on non-simplicial cells. Per-cell gradient estimation can be obtained by applying the Green-Gauss formula to each cell σ

$$\int_{\sigma} \nabla f dV = \int_{\partial\sigma} f \mathbf{n} dA$$

where $\mathbf{n}$ is the normal direction to the boundary of σ. In [SBK14] this approach is investigated in detail and some computational alternatives are proposed. Note that this formula gives the same result of per-cell linear estimation in case σ is a simplicial cell.

The extension of per-vertex methods is straightforward. Once per-cell constant gradient has been obtained, the method described in Section 3.2.1 can be applied with no change, by averaging gradients in the incident cells of each vertex. Also the methods described in Sections 3.2.2 and 3.2.3 can be applied directly, since they only require retrieving vertices in the star (or k-ring) of a given vertex. In the latter case, since the direct neighbors of each vertex are usually fewer in a polyhedral mesh, all vertices of cells incident at a given vertex might be considered as neighbors.

6.2. Manifold domains

When domain Ω is a manifold, such as a surface embedded in 3D space, gradient fields are defined in tangent space. Per-face constant gradient estimation can be applied unchanged, because the tangent plane at a face of a surface mesh corresponds with the plane of the face itself. Conversely, the estimation of gradients at vertices, as well as their extension to the whole domain, need an affine connection that can relate tangent planes at different cells of Σ. The problem of obtaining such an affine connection is treated in detail in [LTGD16], and this allows evaluating several differential properties of vector fields on surfaces. There exists a vast literature on

the design and analysis of vector fields on surfaces. However, to the best of our knowledge, the explicit problem of estimating gradient fields per-vertex on a surface has not been investigated in the literature. In a recent survey [DGDT16], several techniques for vector field processing are reviewed, yet gradient estimation is reported only under the per-face approach.

6.3. Field tracing

Once a gradient has been estimated on a mesh Σ, it is possible to trace its integral lines. Tracing a piecewise-constant gradient is straightfoward and gives piecewise-linear integral lines, but it often produces artifacts.

A piecewise-linear gradient model is much more reliable, but it is also much more difficult to trace. As shown in [NJ99], the integral lines of a linear vector field on a simplex may be expressed in analytic form with a rather complex parametrization, which cannot be easily intersected with the boundary of the cell. Tracing such exact lines has been attempted in [KRG03], but it may lead to numerical issues. One straightforward alternative consists of computing a piecewise-linear approximation of an integral line: starting at a given point, the line is traced by considering a polyline that follows the gradient for a given step at each node; with this approach, the intersection with the boundary of cells is easy, but the tracing procedure is prone to a potentially large drift which may propagate at each step. Although this solution converges to the exact one, as the length of the step tends to zero, it is hard to set a step that guarantees a bound to the drift. One dangerous consequence of accumulated numerical errors and/or drift is that integral lines that should proceed parallel may meet and intersect, thus corrupting the topology of the field.

Some recent approaches try to address the topological correctness of a piecewise-linear vector field on a surface, by detecting coherent bundles of integral lines that travel parallel inside a cell [BJB*12, RS14, MPZ14]. Under these approaches, big drifts are avoided and the overall topological structure of the field is maintained, at the cost of a rougher approximation of lines inside a given cell.

7. Concluding remarks

We proposed an experimental evaluation in 2D of four different gradient estimators for simplicial meshes. We started from the consideration that the ubiquitous piece-wise constant gradients, obtained computing the partial derivatives of the function sampled at the mesh vertices and linearly extended within each cell, suffer from a number of limitations (Section 1).

In our study, we have considered a family of periodic functions, and tested each method on a variety of discrete meshes which are common in applied sciences, also testing resiliency to noise and computational performances.

Not surprisingly, experiments confirm that overall – due to its piece-wise constant nature – PCE performs worse than any piecewise linear gradient estimator that we tested. The worst performances come for anisotropic meshes or signals affected by noise, where PCE has angle error which grows as the same rate observed for piece-wise linear gradients, and magnitude error which grows at a much higher rate (Figures 9 and 11). This is probably due to its higher locality, which makes it extremely sensitive to perturbations in the signal or sampling, although it is not yet clear why this effect manifests itself more on the magnitude component than on angles. On the positive sides, being computed separately on each triangle, PCE is basically insensitive to boundaries, where it outperforms all its piece-wise linear counterparts (Figure 4). This makes it the best possible choice for domains with boundaries and high ratio between perimeter and area, where the error introduced by piece-wise linear methods can take over the whole gradient field.

Restricting to piece-wise linear methods, linear regression (LR) seems to do slightly better than AGS and LSDD when the mesh is quite refined with respect to signal frequency (Figure 7), and has a superior behavior on anisotropic meshes (Figure 9). However, LR suffers the most on boundaries (Figure 4), and is the most expensive one, as it requires solving a linear system for each vertex, making it hardly scalable on big meshes (Figure 12). On the other hand, AGS and LSDD often show the same asymptotic behaviour (but AGS has lower error). Considering computational cost, LSDD is similar to LR, as it requires solving a linear system (though smaller) per vertex, and it does not allow to pre-factor data to compute multiple gradients for the same mesh and different fields (e.g. for time evolving measures). To this end, PCE and AGS offer the best performances, as they are both encoded in a sparse matrix that can be pre-computed and used with any field defined on the mesh. Excluding the cost of computing the matrix, gradient computation is as expensive as performing a matrix-vector multiplication, making PCE and AGS extremely scalable on big meshes.

All in all, we conclude that the AGS method seems to be the best implementative choice for industrial and research code in which gradient estimation is relevant, as it combines the superior accuracy of piece-wise linear gradient fields with the efficiency and scalability of its computation via matrix vector product.

For future works, we aim at extending this study in a number of ways. First and foremost, we aim to analyze the case $d = 3$, for which we already did some preliminary tests that seem to agree with results in 2D. Then, we are interested in considering estimation on manifold domains and integral curve tracing problems. The latter seem to be particularly interesting (and challenging) for the piece-wise linear methods, were singularities may arise at any point inside elements domain and not only at the mesh vertices.

Acknowledgements

This work is supported by the Italian Ministry of Education, University and Research under Program PRIN 2015, Project DSurf, Grant N. 2015B8TRFM_002, and by the EU ERC Advanced Grant CHANGE, grant agreement No 694515.

References

[BJB*12] BHATIA H., JADHAV S., BREMER P., CHEN G., LEVINE J. A., NONATO L. G., PASCUCCI V.: Flow Visualization with Quantified Spatial and Temporal Errors Using Edge Maps. *Visualization and Computer Graphics, IEEE Transactions on 18*, 9 (2012), 1383–1396. 9

[CHdSM14] CERBATO G., HURTADO F. S., DA SILVA A. F. C., MALISKA C. R.: Analysis of gradient reconstruction methods on polygonal grids applied to petroleum reservoir simulation. In *15th Brazilian Congress of Thermal Science and Engineering, Belém. ENCIT Proceedings* (2014). 2

[CHM09] CORREA C. D., HERO R., MA K.-L.: A comparison of gradient estimation methods for volume rendering on unstructured meshes. *IEEE Transactions on Visualization & Computer Graphics*, 3 (2009), 305–319. 2

[DGDT16] DE GOES F., DESBRUN M., TONG Y.: Vector field processing on triangle meshes. In *ACM SIGGRAPH 2016 Courses, SIGGRAPH 2016* (New York, New York, USA, July 2016), Pixar Animation Studios, United States, ACM Press, pp. 1–49. 1, 2, 9

[GJ*10] GUENNEBAUD G., JACOB B., ET AL.: Eigen v3. http://eigen.tuxfamily.org, 2010. 4

[HS97] HYMAN J. M., SHASHKOV M.: Natural discretizations for the divergence, gradient, and curl on logically rectangular grids. *Computers and Mathematics with Applications 33*, 4 (Jan. 1997), 81–104. 2

[JS02] JIRKA T., SKALA V.: Gradient Vector Estimation Using Quadratic Regression Function. In *Int. Conf. on Computer Vision and Graphics* (Zakopane, Poland, 2002). 2

[KRG03] KIPFER P., RECK F., GREINER G.: Local Exact Particle Tracing on Unstructured Grids. *Computer Graphics Forum 22*, 2 (June 2003), 133–142. 1, 9

[Liv17] LIVESU M.: cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes, 2017. https://github.com/mlivesu/cinolib/. 4

[Liv18] LIVESU M.: A heat flow relaxation scheme for n dimensional discrete hyper surfaces. *Computers & Graphics 71* (2018), 124 – 131. doi:10.1016/j.cag.2018.01.004. 7

[LTGD16] LIU B., TONG Y., GOES F. D., DESBRUN M.: Discrete connection and covariant derivative for vector field analysis and design. *ACM Trans. Graph. 35*, 3 (Mar. 2016), 23:1–23:17. URL: http://doi.acm.org/10.1145/2870629, doi:10.1145/2870629. 9

[MDSB03] MEYER M., DESBRUN M., SCHRÖDER P., BARR A. H.: Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. In *Visualization and Mathematics III*. Springer, Berlin, Heidelberg, Berlin, Heidelberg, 2003, pp. 35–57. 3

[MPZ14] MYLES A., PIETRONI N., ZORIN D.: Robust field-aligned global parametrization. *ACM Transactions on Graphics 33*, 4 (July 2014), 1–14. 9

[NCKG00] NEUMANN L., CSEBFALVI B., KONIG A., GROLLER E.: Gradient estimation in volume data using 4D linear regression. *Computer Graphics Forum 19*, 3 (2000), C351–C357. 2

[NJ99] NIELSON G. M., JUNG I.-H.: Tools for computing tangent curves for linearly varying vector fields over tetrahedral domains. *IEEE Transactions on Visualization and Computer Graphics* (1999). 1, 9

[RS14] RAY N., SOKOLOV D.: Robust polylines tracing for n-symmetry direction field on triangulated surfaces. *ACM Transactions on Graphics (TOG) 33*, 3 (2014), 30. 9

[SBK14] SOZER E., BREHM C., KIRIS C. C.: Gradient calculation methods on arbitrary polyhedral unstructured meshes for cell-centered CFD solvers. In *52nd AIAA Aerospace Sciences Meeting - AIAA Science and Technology Forum and Exposition, SciTech 2014* (Reston, Virginia, Jan. 2014), Science and Technology Corporation, Hampton, Hampton, United States, American Institute of Aeronautics and Astronautics. 2, 8

[She96] SHEWCHUK J. R.: Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied computational geometry towards geometric engineering*. Springer, 1996, pp. 203–222. 4