Jet fitting 3: A Generic C++ Package for Estimating the Differential Properties on Sampled Surfaces via Polynomial Fitting

Frédéric Cazals, Marc Pouget

`Jet_fitting_3`: A Generic C++ Package for Estimating the Differential Properties on Sampled Surfaces via Polynomial Fitting

FRÉDÉRIC CAZALS
INRIA Sophia-Antipolis, France.
and
MARC POUGET
INRIA Nancy Gand Est - LORIA, France.

Surfaces of $\mathbb{R}^3$ are ubiquitous in science and engineering, and estimating the local differential properties of a surface discretized as a point cloud or a triangle mesh is a central building block in Computer Graphics, Computer Aided Design, Computational Geometry, Computer Vision. One strategy to perform such an estimation consists of resorting to polynomial fitting, either interpolation or approximation, but this route is difficult for several reasons: choice of the coordinate system, numerical handling of the fitting problem, extraction of the differential properties.

This paper presents a generic C++ software package solving these problems. On the theoretical side and as established in a companion paper, the interpolation and approximation methods provided achieve the best asymptotic error bounds known to date. On the implementation side and following state-of-the-art coding rules in Computational Geometry, genericity of the package is achieved thanks to four template classes accounting for (a) the type of the input points (b) the internal geometric computations (c) a convertion mechanism between these two geometries and (d) the linear algebra operations. An instantiation within the Computational Geometry Algorithms Library (CGAL, version 3.3) and using LAPACK is also provided.

Categories and Subject Descriptors: G.4 [**Mathematics of Computing**]: MATHEMATICAL SOFTWARE—*Algorithm design and analysis*; I.3.5 [**Computing Methodologies**]: COMPUTER GRAPHICS—*Computational Geometry and Object Modeling*

General Terms: Algorithms, Design, Documentation

Additional Key Words and Phrases: Approximation, Computational Geometry, C++ design, Differential Geometry, Interpolation, Numerical Linear Algebra, Sampled Surfaces

Author's emails: Frederic.Cazals@sophia.inria.fr and Marc.Pouget@loria.fr.
This work is partially supported by the IST Programme of the 6th Framework Programme of the EU as a STREP (FET Open Scheme) Project under Contract No IST-006413 - ACS (Algorithms for Complex Shapes with certified topology and numerics) - `http://acs.cs.rug.nl/`

1. INTRODUCTION

1.1 Estimating Differential Quantities

Surface segmentation in Computer Aided Geometric Design and reverse engineering, surface smoothing / denoising in Computer Graphics, or surface reconstruction in Computational Geometry are example applications where one faces the problem of estimating the differential properties of a smooth surface known through a discretization —collection of sample points or triangular mesh. Depending on the particular constraints, first order (tangent space), second order (principal curvatures and directions), or third order informations (derivatives of the principal curvatures along the curvature lines) are required. Due to its importance, this problem triggered an important activity in several communities, and the reader is referred to [Petitjean 2002] for a report up to 2001. Given a point at which the differential quantities are sought, all methods developed require gathering points or triangles in a neighborhood of that point. Methods may be classified into two families. On one hand, strategies relying on *discrete differential geometry* only use the information provided by the mesh [Pinkall and Polthier 1993; Meyer et al. 2003; Borrelli et al. 2003; Cohen-Steiner and Morvan 2003]. On the other hand, fitting methods rely on smooth differential geometry and approximation, and hence directly accommodate point clouds.

Each category has a champion in terms theoretical guarantees provided by the estimation procedure, when the sampling becomes infinitely dense. In the former category, the normal cycle theory used in [Cohen-Steiner and Morvan 2003] provides a linear error bound on the estimate of the integral of the Weingarten map of surface discretized by a mesh. In the later, the polynomial fitting strategy developed by the authors in [Cazals and Pouget 2005a] allows a precise specification of the coefficients of the local representation of the surface as a height function —see Thm 2.1 below. The polynomial fitting method has three advantages : first, differential properties of any order can be retrieved; second, the coefficients estimated feature the best error bounds known so far; third, it accommodates point clouds —a mesh is not mandatory.

1.2 Contributions and Paper Overview

This paper describes the C++ software package `Jet_fitting_3` which implements the polynomial fitting method [Cazals and Pouget 2005a], that is (i) the mathematics and numerical issues involved (ii) the C++ implementation. The paper is organized as follows. Section 2 specifies the objects manipulated and recalls the principles of the fitting algorithm, while algorithmic and mathematical details are developed in section 3. Software design issues are discussed in section 4, while example code and illustrations are provided in section 5.

2. ESTIMATING DIFFERENTIAL PROPERTIES BY POLYNOMIAL FITTING

2.1 Jets, Monge Form and Polynomial Fitting

2.1.1 *Smooth Surfaces, d-jets and the Monge Form.* To present the method, we shall need the following notions. Consider a smooth surface. About one of its points, consider a coordinate system whose z-axis is not contained in the tangent space of the surface at that point. In such a frame, the surface can locally be

written as the graph of a bivariate function. Letting *h.o.t* stand for *higher order terms*, one has :

$$z(x,y) = J_{B,d}(x,y) + h.o.t \quad \text{with } J_{B,d}(x,y) = \sum_{k=0}^{k=d}(\sum_{i=0}^{i=k} \frac{B_{k-i,i}x^{k-i}y^i}{i!(k-i)!}). \quad (1)$$

The degree d polynomial $J_{B,d}$ is the Taylor expansion of the function z, we call it its d-jet. Notice that a d-jet contains $N_d = (d+1)(d+2)/2$ coefficients.

While the first order properties of the surface are related to its tangent space, its second order properties are encoded by the principal curvatures and directions [de Carmo 1976]. When the principal curvatures are distinct —the point is not a so-called umbilic, the principal directions d_1, d_2 are well defined, and these (non oriented) directions together with the normal vector n define two direct orthonormal frames. If v_1 is a unit vector of direction d_1, there exists a unique unit vector v_2 so that (v_1, v_2, n) is direct; and the other possible frame is $(-v_1, -v_2, n)$. These coordinate systems are called the *Monge* frames, and in any of them, the Taylor expansion of the surface has the following canonical expression, called the *Monge form*:

$$z(x,y) = \frac{1}{2}(k_1x^2 + k_2y^2) + \frac{1}{6}(b_0x^3 + 3b_1x^2y + 3b_2xy^2 + b_3y^3) \quad (2)$$

$$+ \frac{1}{24}(c_0x^4 + 4c_1x^3y + 6c_2x^2y^2 + 4c_3xy^3 + c_4y^4) + h.o.t \quad (3)$$

Recall that coefficients k_1, k_2 are the principal curvatures, b_0, b_3 are the directional derivatives of k_1, k_2 along their respective curvature lines, while b_1, b_2 are the directional derivatives of k_1, k_2 along the other curvature lines.

The Monge coordinate system can be computed from any d-jet with $d \geq 2$, and so are the Monge coefficients. The Monge representation characterizes the surface in a canonical way, and is the output of our algorithm.

2.1.2 *Interpolating or Approximating the d-jet.* Consider a sampled smooth surface, and assume we are given a collection of points P about a given sample p. We aim at estimating the order-d differential properties of the surface at point p from the point set $P^+ = P \cup \{p\}$ –we note $N = \mid P^+ \mid$. The estimation algorithms developed in [Cazals and Pouget 2005a] consist of fitting the d-jet, using bivariate polynomial interpolation or approximation on the point set P^+. More precisely, in a well chosen coordinate system, the fitting consists of finding the coefficients $A_{i,j}$ of the degree d polynomial $J_{A,d} = \sum_{k=0}^{k=d}(\sum_{i=0}^{i=k} \frac{A_{k-i,i}x^{k-i}y^i}{i!(k-i)!})$.

Denote $p_i = (x_i, y_i, z_i),\ i = 1, \ldots, N$, the coordinates of the sample points of P^+. For interpolation the linear equations to solve are $J_{A,d}(x_i, y_i) = z_i,\ i = 1, \ldots, N$, and for approximation one has to minimize $\sum_{i=1}^{N}(J_{A,d}(x_i, y_i) - z_i)^2$. To provide the linear algebra formulation of the problem, define:

$$\begin{aligned} A =& (A_{0,0}, A_{1,0}, A_{0,1}, \ldots, A_{0,d})^T, \\ Z =& (z_1, z_2, \ldots, z_N)^T, \\ M =& (1, x_i,\ y_i,\ \frac{x_i^2}{2}, \ldots,\ \frac{x_i y_i^{d-1}}{(d-1)!},\ \frac{y_i^d}{d!})_{i=1,\ldots,N}. \end{aligned}$$

Using these notations, solving the fitting problem by interpolation is tantamount to solving $MA = Z$, while the approximation strategy requires solving $\min ||MA - Z||_2$.

The following theorem precisely states the order of convergence of the polynomial fitting method. Given a parameter h measuring the *sampling density*, the following theorem, providing the best asymptotic estimates known to date, is proved in [Cazals and Pouget 2005a] :

THEOREM 2.1. *A polynomial fitting of degree d estimates any k^{th}-order differential quantity to accuracy $O(h^{d-k+1})$:*

$$A_{i,k-i} = B_{i,k-i} + O(h^{d-k+1}). \tag{4}$$

In particular:

—*the coefficients of the unit normal vector are estimated with accuracy $O(h^d)$.*

—*the coefficients of the second fundamental form and the shape operator are approximated with accuracy $O(h^{d-1})$, and so are the principal curvatures and directions (as long as they are well defined, i.e. away from umbilics).*

2.1.3 *Algorithm.* Based on the above concepts, the algorithm consists of four steps.

(1) We perform a principal component analysis (PCA) on P^+. This analysis outputs three orthonormal eigenvectors and the associated eigenvalues. If the surface is well sampled, we expect the PCA to provide one small and two large eigenvalues, the eigenvector associated to the small one approximating the normal vector.
(2) We perform a change of coordinates to move the samples into the coordinate system defined by the PCA eigenvectors. We then resort to polynomial fitting, so as to either interpolate or approximate the d-jet $J_{B,d}$ of the surface. This fitting reduces to linear algebra operations.
(3) From the d-jet $J_{A,d}$, we compute the Monge basis (d_1, d_2, n).
(4) Finally, we compute the Monge coefficients.

For the fitting, we do not assume the z-axis of the fitting to be the normal of the surface. Hence we keep the first order coefficients of the polynomial $J_{A,d}$. It is proved that such a choice improves the estimations [Cazals and Pouget 2005a].

Note that we do not aim at identifying exactly special points such as umbilics where both principal curvatures are equal. This would require the original surface and the fitted surface to coincide. This implying in turn that the original surface is a graph of a bivariate polynomial of a lower degree than the fitted polynomial, and that the coordinate system used for the fitting has the same height direction.

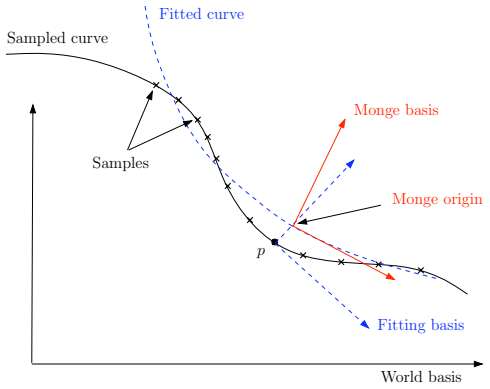

Fig. 1. The three orthonormal basis involved in the estimation.

2.2 Degenerate Cases

For an interpolation of degree d, there are special configuration of the sample points for which the interpolation problem is not well defined —if the point lie on an algebraic curve of degree at most d. In such a case, the condition number of the system to solve is infinite. In practice, this condition number is passed to the user for an assessment of the accuracy of the result.

3. MATHEMATICAL AND ALGORITHMIC DETAILS

In this section, we detail the mathematics involved, in order to justify the C++ design choices.

As illustrated on Fig. 3, we deal with three relevant direct orthonormal basis: the world-basis W, fitting-basis F, and the Monge-basis $M = (d_1, d_2, n)$. If X, Y are two such basis, the matrix used to transport a vector from X to Y is denoted $P_{X\to Y}$. If V_X (resp. V_Y) are the coordinates of a vector V with respect to the basis X (resp. Y), then : $V_Y = P_{X\to Y}V_X$. Note that as all considered basis are orthonormal, the matrix to change from one to another is orthonormal, that is its inverse is its transpose : $P_{Y\to X} = P_{X\to Y}^{-1} = P_{X\to Y}^{T}$. For computations, note that the columns of $P_{X\to Y}$ are the coordinates of the vectors of X in the basis Y.

3.1 Computing a Basis for the Fitting

Input : sample points
Output : the fitting-basis

We perform a Principal Component Analysis of the sample points, which requires a linear algebra method to perform an eigen analysis of a 3×3 symmetric matrix. This analysis gives an orthonormal basis whose z-axis is chosen as the eigenvector associated to the smallest eigenvalue [1]. Notice that one may have to swap the orientation of a vector to get a direct basis. We call fitting-basis and denote F this basis. Notice also that the quality of the sampling is encoded in the PCA

[1] Another possibility is to choose as z-axis the axis of the world-basis with the least angle with the axis determined with the PCA. Then the change of basis reduces to a permutation of axis.

eigenvalues, as a good sampling is characterized by a small eigenvalue (indicating the normal vector) and two similar bigger ones (indicating the tangent space).

3.2 Solving the Interpolation / Approximation Problem

Input : sample points, fitting-basis
Output : coefficients $A_{i,j}$ of the fitted polynomial (in the fitting-basis)

The fitting problem is solved in the fitting basis, whose origin is the point p. Therefore, sample points are expressed in this frame thanks to a translation $(-p)$ and a multiplication by $P_{W \to F}$.

We solve the system $MA = Z$ —in the least square sense for approximation. There is a preconditioning of the matrix M so as to improve the condition number. Assuming the $\{x_i\}$, $\{y_i\}$ are of order h, the preconditioning consists of performing a column scaling by dividing each monomial $x_i^k y_i^l$ by h^{k+l}. The parameter h is chosen as the mean value of the $\{x_i\}$ and $\{y_i\}$. In other words, the new system is $M'Y = (MD^{-1})(DA) = Z$ with D the diagonal matrix $D = (1, h, h, h^2, \ldots, h^d, h^d)$, so that the solution A of the original system is $A = D^{-1}Y$.

There is always a single solution to the system $M'Y = Z$ since for under constrained systems we also minimize $||Y||_2$. The method uses a singular value decomposition of the $N \times N_d$ matrix $M' = USV^T$, where U is a $N \times N$ orthogonal matrix, V is a $N_d \times N_d$ orthogonal matrix and S is a $N \times N_d$ matrix with the singular values on its diagonal. Denoting r the rank of M', we can decompose $S = \begin{pmatrix} D_r & 0_{r,\ N_d-r} \\ 0_{N-r,\ r} & 0_{N-r,\ N_d-r} \end{pmatrix}$. The number r, which is the number of non zero singular values, is strictly lower than N_d if the system is under constrained. In any case, the unique solution which minimize $||Y||_2$ is given by :

$$Y = V \begin{pmatrix} D_r^{-1} & 0_{N_d-r,\ r} \\ 0_{r,\ N-r} & 0_{N_d-r,\ N-r} \end{pmatrix} U^T Z. \tag{5}$$

The condition number of the matrix M' is the ratio of the maximal and the minimal singular values. It is infinite if the system is under constrained, that is the smallest singular value is zero.

Remark. A system $MA = Z$ can be solved using a variety of methods. A QR decomposition can be substituted to the SVD. One can also use the normal equation $M^T MA = MTZ$ and apply methods for square systems such as LU, QR or Cholesky since $M^T M$ is symmetric definite positive when M has full rank. The advantages of the SVD is that it works directly on the rectangular system and gives the condition number of the system. For more on these alternatives, see [Golub and van Loan 1983].

The method to solve the system is provided by the template parameter $SvdTraits$, so that it is up to the user to choose amongst these techniques. See section 4.

3.3 Computing the Principal Curvatures and Directions

Input : coefficients $A_{i,j}$ of the fitted polynomial
Output : Monge-basis wrt fitting-basis and world-basis

The surface on which local differential quantities are computed is the graph of $J_{A,d}$, and the point on this surface where the estimation is sought has coordinates

$(0, 0, J_{A,d}(0, 0)$. Computations are done in the fitting-basis. The partial derivatives, evaluated at $(x, y) = (0, 0)$, of the fitted polynomial $J_{A,d}(x, y)$ are $A_{i,j} = \frac{\partial^{i+j} J_{A,d}}{\partial^i x \partial^j y}$, hence

$$J_{A,d}(x, y) = A_{0,0} + A_{1,0}x + A_{0,1}y + \frac{1}{2}(A_{2,0}x^2 + 2A_{1,1}xy + A_{0,2}y^2) \quad (6)$$

$$+ \frac{1}{6}(A_{3,0}x^3 + 3A_{2,1}x^2y + \ldots) + \ldots \quad (7)$$

Equipped with these coefficients, differential quantities up to order two are retrieved as follows:

—The origin, that is the point of the fitted surface where the estimation is performed, is $(0, 0, A_{0,0})$.

—The normal is $n = (-A_{1,0}, -A_{0,1}, 1)/\sqrt{1 + {A_{1,0}}^2 + {A_{0,1}}^2}$.

—Curvature related properties are retrieved resorting to standard differential calculus [de Carmo 1976, Chap. 3]. More precisely, the Weingarten operator $W = -I^{-1}II$ is first computed in the basis of the tangent plane $\{(1, 0, A_{1,0}), (0, 1, A_{0,1})\}$, where I, II are the first and second fundamental forms :

$$I = \begin{pmatrix} 1 + A_{1,0}^2 & A_{1,0}A_{0,1} \\ A_{1,0}A_{0,1} & 1 + A_{0,1}^2 \end{pmatrix} \quad II = \begin{pmatrix} \frac{A_{2,0}}{\sqrt{1+{A_{1,0}}^2+{A_{0,1}}^2}} & \frac{A_{1,1}}{\sqrt{1+{A_{1,0}}^2+{A_{0,1}}^2}} \\ \frac{A_{1,1}}{\sqrt{1+{A_{1,0}}^2+{A_{0,1}}^2}} & \frac{A_{0,2}}{\sqrt{1+{A_{1,0}}^2+{A_{0,1}}^2}} \end{pmatrix} \quad (8)$$

We compute an orthonormal basis of the tangent plane using the Gram-Schmidt algorithm. Denoting P the matrix of the orthonormal vectors in the fitting basis, the matrix of the Weingarten map in this orthonormal basis of the tangent space is given by $W' = P^{-1}WP$. In this orthonormal basis, the matrix of the Weingarten map is symmetric and we diagonalize it. One finally gets the principal curvatures which are the eigenvalues of W, and the associated principal directions d_1, d_2. This gives an orthonormal direct basis $(d_1, d_2, n = d_1 \wedge d_2)$ called the Monge-basis those vectors are given by their coordinates in the fitting-basis.

The coordinates of d_1, d_2 and n expressed in the world-basis are obtained by multiplying their coordinates in the fitting-basis by $P_{F \to W}$. The same holds for the origin point which has in addition to be translated by p, i.e. the coordinates of the origin point are $P_{F \to W}(0, 0, A_{0,0}) + p$.

3.4 Computing Higher Order Monge Coefficients

Input : coefficients $A_{i,j}$ of the fitted polynomial, Monge-basis wrt fitting-basis ($P_{F \to M}$)
Output : third and fourth order coefficients of the Monge form

Having established the equation of the fitted polynomial in the fitting basis, we derive an implicit equivalent, say $f(x, y, z) = 0$, in the Monge basis. Then, from the implicit function theorem [de Carmo 1976], we know that the surface $f(x, y, z) = 0$ can locally be written as the graph of a height function $z = g(x, y)$. The Taylor expansion of g at $(0, 0)$ are the Monge coefficients sought. More formally:

THEOREM 3.1. *Third and fourth order coefficients of the Monge form with respect to derivatives of the implicit equation $f(x, y, z) = 0$ of the fitted polynomial in*

the Monge basis are given by:

$$b_0 = g_{3,0} = -\frac{f_{3,0,0}f_{0,0,1}-3\,f_{1,0,1}f_{2,0,0}}{f_{0,0,1}^2}$$

$$b_1 = g_{2,1} = -\frac{-f_{0,1,1}f_{2,0,0}+f_{2,1,0}f_{0,0,1}}{f_{0,0,1}^2}$$

$$b_2 = g_{1,2} = -\frac{f_{1,2,0}f_{0,0,1}-f_{1,0,1}f_{0,2,0}}{f_{0,0,1}^2}$$

$$b_3 = g_{0,3} = -\frac{f_{0,3,0}f_{0,0,1}-3\,f_{0,1,1}f_{0,2,0}}{f_{0,0,1}^2}$$

$$c_0 = g_{4,0} = \frac{-3\,f_{0,0,2}f_{2,0,0}^2-f_{4,0,0}f_{0,0,1}^2+6\,f_{2,0,1}f_{0,0,1}f_{2,0,0}+4\,f_{1,0,1}f_{0,0,1}f_{3,0,0}-12\,f_{1,0,1}^2f_{2,0,0}}{f_{0,0,1}^3}$$

$$c_1 = g_{3,1} = \frac{-6\,f_{1,0,1}f_{0,1,1}f_{2,0,0}+3\,f_{0,0,1}f_{1,1,1}f_{2,0,0}+3\,f_{1,0,1}f_{0,0,1}f_{2,1,0}+f_{0,1,1}f_{0,0,1}f_{3,0,0}-f_{3,1,0}f_{0,0,1}^2}{f_{0,0,1}^3}$$

$$\begin{aligned} c_2 = g_{2,2} = -(&f_{2,2,0}f_{0,0,1}^2 - f_{2,0,1}f_{0,0,1}f_{0,2,0} - 2\,f_{1,0,1}f_{0,0,1}f_{1,2,0} + 2\,f_{1,0,1}^2f_{0,2,0}\\ &-f_{0,2,1}f_{0,0,1}f_{2,0,0} + 2\,f_{0,1,1}^2f_{2,0,0} - 2\,f_{0,1,1}f_{0,0,1}f_{2,1,0} + f_{0,0,2}f_{0,2,0}f_{2,0,0})/f_{0,0,1}^3 \end{aligned}$$

$$c_3 = g_{1,3} = -\frac{f_{1,3,0}f_{0,0,1}^2+6\,f_{1,0,1}f_{0,1,1}f_{0,2,0}-3\,f_{0,1,1}f_{0,0,1}f_{1,2,0}-3\,f_{1,1,1}f_{0,0,1}f_{0,2,0}-f_{1,0,1}f_{0,0,1}f_{0,3,0}}{f_{0,0,1}^3}$$

$$c_4 = g_{0,4} = -\frac{f_{0,4,0}f_{0,0,1}^2+3\,f_{0,0,2}f_{0,2,0}^2-6\,f_{0,2,1}f_{0,0,1}f_{0,2,0}-4\,f_{0,1,1}f_{0,0,1}f_{0,3,0}+12\,f_{0,1,1}^2f_{0,2,0}}{f_{0,0,1}^3}$$

PROOF. The implicit equation of the fitted polynomial surface in the fitting-basis with origin the point $(0, 0, A_{0,0})$ is $Q = 0$ with

$$Q = -w - A_{0,0} + \sum_{i,j} \frac{A_{i,j}u^iv^j}{i!j!}. \tag{9}$$

Notice the term $-A_{0,0}$, which stems from the fact that the surface is translated so as to go through the intersection with the z-axis.

The equation in the Monge-basis is obtained by substituting (u, v, w) by $P_{M\to F}(x, y, z)$, let us denote $f(x, y, z)$ this implicit equation. By the implicit function Thm and the definition of the Monge-basis, we locally have at $(0, 0, 0)$

$$f(x, y, z) = 0 \Leftrightarrow z = g(x, y), \tag{10}$$

and the Taylor expansion of g at $(0, 0)$ are the Monge coefficients sought. Let us denote the partial derivatives evaluated at the origin of f and g by $f_{i,j,k} = \frac{\partial^{i+j+k}f}{\partial^i x\partial^j y\partial^k z}$ and $g_{i,j} = \frac{\partial^{i+j}g}{\partial^i x\partial^j y}$. By the definition of the Monge basis, one has $g_{0,0} = g_{1,0} = g_{0,1} = g_{1,1} = 0$ and $g_{2,0} = k_1$, $g_{0,2} = k_2$. The partial derivative of order n of f depends on the matrix $P_{M\to F}$ and the partial derivatives of order at most n of $J_{A,d}$ (that is coefficients $A_{i,j}$ for $i + j \leq n$). The third and fourth order Monge coefficients of are computed applying the chain rule while differentiating the equation $f(x, y, g(x, y)) = 0$. For instance, differentiating once, twice and thrice with respect to x gives

$$f_{1,0,0} + f_{0,0,1}g_{1,0} = 0 \tag{11}$$

$$f_{2,0,0} + 2f_{1,0,1}g_{1,0} + f_{0,0,2}g_{1,0}^2 + f_{0,0,1}g_{2,0} = 0 \tag{12}$$

$$f_{3,0,0} + 3f_{2,0,1}g_{1,0}^2 + 3f_{1,0,1}g_{2,0} + 3f_{1,0,2}g_{1,0}^2 + f_{0,0,3}g_{1,0}^3 \tag{13}$$

$$+ 3f_{0,0,2}g_{1,0}g_{2,0} + f_{0,0,1}g_{3,0} = 0 \tag{14}$$

In addition, at $(x, y) = (0, 0)$, equation (12) gives $g_{2,0} = -f_{2,0,0}/f_{0,0,1}$ and equation

(13) leads to $f_{0,0,1}g_{3,0} = -(f_{3,0,0} + 2f_{1,0,1}g_{2,0})$, we then obtain the equation for b_0. Other equations are obtained by similar computations. □

4. CODE DESIGN

In this section, we outline the C++ design of the `Jet_fitting_3` package. For readers not familiar with generic geometric code, we begin with some generalities. In particular, we recall the principles underlying the development of CGAL, which is the standard library [2] for robust and efficient geometric software —see `www.cgal.org`. From these principles, we discuss the CGAL and stand-alone implementations of the `Jet_fitting_3` package.

4.1 CGAL: Templated Geometric Code

CGAL consists of generic classes implementing a number of geometric algorithms. Each implementation distinguishes the combinatorial from the numerics. For example, a triangulated surface consists of a collection of triangles with the proper incidences between them (the combinatorial part describing the surface topology), and of a collection of points associated to the vertices of the triangulation (the coordinates representing the numerical part). As an algorithm operating on a surface should accommodate several types of points, a high level of genericity is achieved thanks to function and class templates. Quoting CGAL's manual:

≪ The algorithms in CGAL's basic library are implemented as function templates or class templates, usually having a template parameter whose name contains the word Traits. This template parameter represents a concept and so has a corresponding set of requirements that define the interface between the algorithm and the geometric (or numeric) primitives it uses. Any concrete class that serves as a model for this concept is a traits class for the given algorithm or data structure. ≫

4.2 Library and stand-alone versions of the code

4.2.1 *Template parameters.* For our algorithm, one has to distinguish the types of the input data from the types used for internal computations. From the previous discussion, these types correspond to template parameters. The input points come from some geometric problem for which the user has already chosen a representation. The type of the coordinates may be of limited precision, or a type supporting exact calculations –this is often requested for robustness issues in geometric computations. On the other hand, as our algorithm provides estimates, we only need a limited numeric precision for internal computations. This calls for two template parameters corresponding to the input data and the internal computations. A conversion mechanism between these representations must also be available. Finally, our algorithm needs a least square solver. As there is no standard solver –a number of linear algebra libraries provide alternatives, the required solver is another template of the algorithm.

4.2.2 *Library version of the code.* The CGAL library naturally provides classes which can be used to represent the input and the objects involved in internal compu-

[2] A package is accepted for integration after a review process similar to that of a journal paper, and upon acceptation by the Editorial Board.

tations. In fact, in the CGAL library, basic geometric objects like points or vectors and some operations on them are represented by the Kernel concept. This concept is parametrized by a type corresponding to the coordinates of the objects. Hence the representation of the objects in our algorithm can be done resorting to models of the kernel concept: one for the input data, one for internal computations. For consistency, the estimated quantities are also given by objects whose types are those of the input kernel. Summarizing, the CGAL implementation of the `Jet_fitting_3` package consists of one main class, `Monge_via_jet_fitting`, which is actually templated by three parameters, that is `Monge_via_jet_fitting<DataKernel, LocalKernel, SvdTraits>`. Default values for the first two parameters are provided as CGAL kernels, which makes the use of the code especially handy. The package has been released with version 3.3 of the library.

4.2.3 *Stand-alone version of the code.* For a collection of algorithms such as CALGO, the dependency upon a library should be avoided. In removing the dependence of `Jet_fitting_3` package upon CGAL, two modifications were performed: first, an additional template parameter `KernelConverters` was added to perform conversion between objects from the `DataKernel` and the `LocalKernel`; second, the default values of the template parameters were removed. The class declaration is thus `Monge_via_jet_fitting<DataKernel, LocalKernel, KernelConverters, SvdTraits>`.

Practically, this has two incidences. On one hand, the `Monge_via_jet_fitting` class is versatile since both the geometric representations of the objects and the linear algebra tools used are totally free. On the other hand, no default values are provided. The CALGO package comes, though, with an example instantiation of the class using CGAL and LAPACK.

4.3 Template Parameters of the Stand-alone Package

4.3.1 *Template Parameter* `DataKernel`. This concept provides the types for the input sample points, together with $3d$ vectors and a number type to encode their coordinates. For consistency, the estimated quantities are also given by objects of this class. One can use any `CGAL::Cartesian<FieldNumberType>` class where `FieldNumberType` is an exact or approximate type for the coordinates.

4.3.2 *Template Parameter* `LocalKernel`. This concept defines the vector and number types used for local computations and to store the PCA basis —fitting basis. One can use `CGAL::Cartesian<FieldNumberType>` where `FieldNumberType` is an approximate type for the coordinates. The default is `CGAL::Cartesian<double>`.

4.3.3 *Template Parameter* `KernelConverters`. This concept enables conversions forth and back between number types, points and vectors of the DataKernel and LocalKernel. That is, the two conversion methods are `D2L_converter` and `L2D_converter`.

4.3.4 *Template Parameter* `SvdTraits`. This concept provides the number, vector and matrix types for algebra operations required by the fitting method. The only method `solve` is a linear solver. A CGAL model is provided by the class `Lapack_svd` using the singular value decomposition of the LAPACK library. But

the user may pass another method of his choice by providing another traits class following the requirements of the concept.

4.3.5 *Compatibility Requirements.* To solve the fitting problem, the sample points are first converted from the `DataKernel` to the `LocalKernel`. (This is respectively done using the `CGAL::Cartesian_converter` and the `KernelConverters` class in the library and stand-alone versions.) Then, changes of coordinate systems and linear algebra operations are performed within this `LocalKernel`. Eventually, the Monge basis and coefficients, computed with the `LocalKernel`, are converted back to the `DataKernel`. (As previously, this is respectively done resorting to `CGAL::Cartesian_converter` and `CGAL::NT_converter`, and to the `KernelConverters` class in the library and stand-alone versions.) This conversion mechanism is transparent for the user. The only requirement comes from the use of the linear solver of the `SvdTraits` for internal computations on objects from the `LocalKernel`: the number types `LocalKernel::FT` and `SvdTraits::FT` must be identical.

4.4 Application Programming Interface

Having discussed the template parameters, we now discuss the API.

The code consists of two main classes: `Monge_form` (storing the final result), and `Monge_via_jet_fitting` (storing the parameters used and the condition numbers). The rationale for using two classes is that while any user resorting to our code is interested in the Monge form of the surface, some users may not care for the condition numbers and more generally for pieces of information stemming from intermediate calculations.

From the class `Monge_via_jet_fitting`, the jet fitting calculation is accessible through *operator()* functions. Several alternative would have been possible. In particular, an option would have been to provide a class performing the calculation within the constructor, the result being accessed through a member function. Using an operator has a major advantage over this solution. To see which, consider a generic geometric algorithm that needs to compute Monge forms. In order to experiment several alternatives to perform this calculation –jet fitting being one of them, the standard way to code such an algorithm is to parametrize it with a template class which is a function object of functor [3]. Thus, providing the calculation through *operator()* is more versatile as the class `Monge_via_jet_fitting` can be used as a functor.

5. USING JET_FITTING_3

5.1 User Interface Specifications

In a nutshell, the `Jet_fitting_3` package has one class, `Monge_via_jet_fitting`, to perform the computation via the method `operator()`. The return value of this method is the nested class `Monge_form` which stores the Monge basis and coefficients.

[3]A functor is a class that can be invoked as if it were an ordinary function. Such a class must redefine *operator()*. An elementary example is that of comparators used by sorting algorithms in the Standard Template Library.

5.1.1 *Input Parameters.* The fitting strategy performed by the class `Monge_via_jet_fitting` requires the following parameters:

—the degree d of the fitted polynomial ($d \geq 1$),

—the degree d' of the Monge coefficients sought, with $1 \leq d' \leq \min(d, 4)$,

—a range of N input points on the surface, with the precondition that $N \geq N_d = (d+1)(d+2)/2$. Note that if $N = N_d$, interpolation is performed; and if $N > N_d$, approximation is used. This range is given by two iterators with value-type `Data_kernel::Point_3`.

—as an optional parameter, the user may specify the coordinate system in which the fitting is performed and thus skip the pca analysis of the input points. This parameter is a orthonormal basis given by three vectors of type `Data_kernel::Vector_3`.

5.1.2 *Output.* As explained in Section 1, the output consists of a coordinate system, the Monge basis, together with the Monge coefficients which are stored in the `Monge_form` class. In addition, more information on the computational issues are stored in the `Monge_via_jet_fitting` class. The `Monge_form` class provides the following information.

—Origin. This is the point on the fitted polynomial surface where the differential quantities have been computed. In the approximation case, it differs from the first input point p : it is the projection of p onto the fitted surface following the z-direction of the fitting basis.

—Monge Basis. The Monge basis (d_1, d_2, n) is orthonormal direct, and the maximal, minimal curvatures are defined wrt this basis. If the user has a predefined normal n_0 (e.g. the sample points come from an oriented mesh) then if $n_0.n > 0$ then max-min is correct; if not, i.e. $n_0.n < 0$, the user should switch to the orthonormal direct basis $(d'_1, d'_2, n') = (d_2, d_1, -n)$ with the maximal curvature $k'_1 = -k_2$ and the minimal curvature $k'_2 = -k_1$ (the method `comply_wrt_given_normal()` does these changes). If $n_0.n = 0$ or is small, the orientation of the surface is clearly ill-defined, and the user may proof-check the samples used to comply with its predefined normal.

—Monge Coefficients. The coefficient of the Monge form are $(k_1, k_2 (\leq k_1), b_0, b_1, b_2, b_3, c_0, c_1, c_2, c_3, c_4)$ for $d' = 4$, if $d' < 4$ then this list is truncated to the requested degree.

In addition, for the user interested in numerical accuracy of the fitting, the class `Monge_via_jet_fitting` stores

—the condition number of the fitting system,

—the coordinate system of the PCA (in which the fitting is performed) and the associated eigenvalues.

5.1.3 *Choosing the Sample Points.* One should note that our algorithm assumes a set of points is given, that is, we do not provide any function selecting the neighbors of a given point in a point cloud or tessellated surface. This selection may indeed be governed by additional knowledge on the data, such as the presence of sharp features on the surface (in which case the selection may not be isotropic), the

presence of noise, etc. As such informations are irrelevant for the fitting problem, we dissociate both steps.

5.2 Example Code

The following piece of code illustrates the easiest use of the package. A `std::vector` of points is constructed from an input file containing their coordinates, then the fitting is performed on this set of points. Note that since the class `CGAL::Exact_predicates_inexact_constructions_kernel` is used for the parameter `DataKernel`, the number type for the coordinates is a multiprecision one enabling the evaluation of exact predicates [4]. On the other hand, the `LocalKernel` parameter is modeled from the class `CGAL::Cartesian<double>`. The third parameter also uses CGAL; these two later parameters are defined in the file `Cgal_local_kernel.h` in the same directory as the example code. Finally, the last parameter comes from the CGAL interface to LAPACK.

```
// For Data_Kernel, Local_Kernel and Kernel_converters templates;
// here from the CGAL library
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include "Cgal_local_kernel.h"
//for the SvdTraits template; here also from CGAL
#include <CGAL/Lapack/Linear_algebra_lapack.h>
//Main code
#include "../include/Monge_via_jet_fitting.h"

typedef Cgal_local_kernel<double> Local_Kernel;
typedef CGAL::Exact_predicates_exact_constructions_kernel
        Data_Kernel;
typedef Data_Kernel::Point_3 DPoint;
typedef CGAL::Monge_via_jet_fitting<Data_Kernel, Local_Kernel,
 Cgal_kernel_converters<Data_Kernel, Local_Kernel>, Lapack_svd>
 My_Monge_via_jet_fitting;
typedef My_Monge_via_jet_fitting::Monge_form My_Monge_form;

int main()
{
  //open the input file
  std::ifstream inFile( "data/in_points.txt", std::ios::in);

  //initalize the in_points container
  double x, y, z;
  std::vector<DPoint> in_points;
  while (inFile >> x) {
    inFile >> y >> z;
    DPoint p(x,y,z);
```

[4] In computational geometry, one distinguishes predicates from constructions. While the a predicate consists of evaluating the sign of an expression involving the data, a construction requires constructing a new geometric objects from the data.

```
    in_points.push_back(p);
  }
  inFile.close();

  // Perform the fitting
  int d_fitting = 4;
  int d_monge = 4;
  My_Monge_form monge_form;
  My_Monge_via_jet_fitting monge_fit;
  monge_form = monge_fit(in_points.begin(), in_points.end(),
                         d_fitting, d_monge);

  //OUTPUT on std::cout
  std::cout << "n :  " << monge_form.normal_direction ()
            << "d1 : " << monge_form.maximal_principal_direction()
            << "d2 : " << monge_form.minimal_principal_direction()
            << "k1 : " << monge_form.principal_curvatures(0)
            << "k2 : " << monge_form.principal_curvatures(1)
            << "b0 : " << monge_form.third_order_coefficients(0)
            << "c0 : " << monge_form.fourth_order_coefficients(0);

  std::cout  << "condition_number : " << monge_fit.condition_number()
             << "pca_eigen_vals and associated pca_eigen_vecs :";
  for (int i=0; i<3; i++)
    std::cout << monge_fit.pca_basis(i).first
              << monge_fit.pca_basis(i).second;
}
```

5.3 Illustrations

5.3.1 *Datasets Used.* We illustrate our algorithm on triangulated surfaces. For a given point, its neighbors are easily retrieved from the connectivity of the mesh. Concerning the accuracy of the method for samplings of surfaces with known differential quantities, the reader is referred to [Cazals and Pouget 2005a]. The complexity of the computation is linear with respect to the number of vertices. On an Intel 2Ghz duo core processor, an average of 5000 point estimations per second are performed for differential quantities up to the second order, and half less for differential quantities up to the fourth order. Our test models are a version of Michelangelo's David (96 Kpoints, see `graphics.stanford.edu/projects/mich`), and a Bézier surface (66 Kpoints) defined as the graph of the function :

$$\begin{aligned} h(u,v) =& 116u^4v^4 - 200u^4v^3 + 108u^4v^2 - 24u^4v - 312u^3v^4 + 592u^3v^3 - 360u^3v^2 \\ &+ 80u^3v + 252u^2v^4 - 504u^2v^3 + 324u^2v^2 - 72u^2v - 56uv^4 + 112uv^3 \\ &- 72uv^2 + 16uv. \end{aligned}$$

5.3.2 *Computing Principal Curvatures and Directions.* We first illustrate the calculation of the principal directions of curvature, which requires estimating differential quantities up to order two at each vertex of the mesh. The corresponding

code can be found in the C++ archive. On the figures 2 and 3, a red segment in the direction of the maximal principal curvature is drawn at each vertex.

Fig. 2. Principal directions associated with k_{max} scaled by k_{min}.

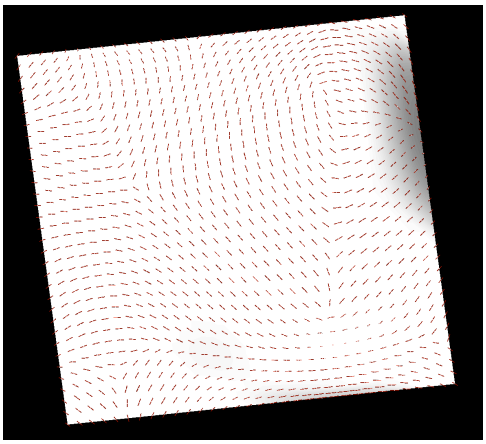

Fig. 3. Principal directions associated with k_{max} on the Bézier surface.

5.3.3 *Computing Ridges.* As a second illustration, we present global features on smooth surfaces: ridges. More precisely, a ridge is a curve along which one of the principal curvatures has an extremum along its curvature line. As ridges features extrema of a principal curvatures, their extraction requires estimating differential quantities up to the third order —and actually up to the fourth order to decide whether the extremum is a maximum or a minimum. These local informations are

further exploited at the triangle level, so as to define ridge segments yielding the ridge lines. See [Cazals and Pouget 2005b] for the algorithm.

As curves of extremal curvature, ridges encode important informations used in segmentation, registration, matching and surface analysis. The lines of minimum/maximum of the minimal/maximal principal curvatures are drawn on the Bézier surface, figure 5. Only part of the ridges, the most visually salient ones, called crests, are shown on the David model, see Fig. 4.

The corresponding C++ code can be found in the CGAL package `Ridges_3`.

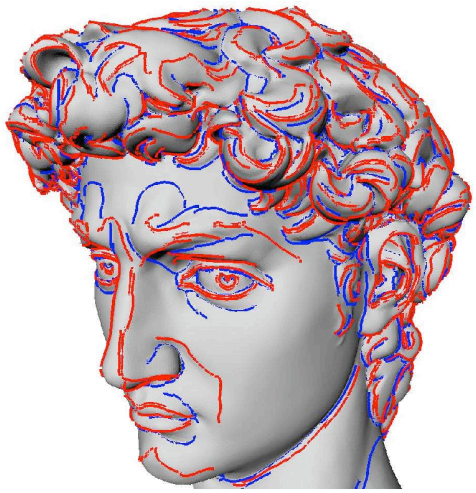

Fig. 4. Crest ridges on the David. Red and blue lines distinguish crest associated to the maximal and minimal principal curvatures.

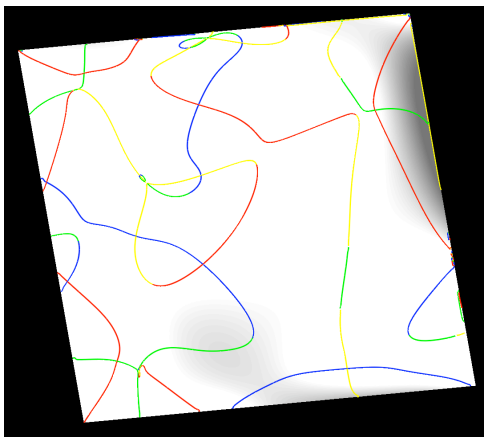

Fig. 5. Ridges on the Bézier surface. Blue (green) lines are ridges associated to the maximum (resp. minimum) of the maximal principal curvature, red (yellow) lines are ridges associated to the minimum (resp. maximum) of the minimal principal curvature.

6. CONCLUSION

The knowledge of local differential quantities of surfaces is ubiquitous in geometry processing. This paper presents an algorithm and its implementation to estimate such quantities from sample points via polynomial fitting, the asymptotic estimates achieved being the best known so far as reported in the companion paper [Cazals and Pouget 2005a]. The algorithm is implemented as a package of the Computational Geometry Algorithms Library `www.cgal.org`, namely `Jet_fitting_3`. A stand-alone version of the code is also provided for the CALGO library. The package uses the generic C++ template design inherent to CGAL, in order to allow the user to parametrize the algorithm to its convenience. Example applications of the package to compute principal directions of curvature and lines of extremal curvatures on meshes are provided.

REFERENCES

Borrelli, V., Cazals, F., and Morvan, J.-M. 2003. On the angular defect of triangulations and the pointwise approximation of curvatures. *CAGD 20,* 6, 319–341.

Cazals, F. and Pouget, M. 2005a. Estimating differential quantities using polynomial fitting of osculating jets. *Comput. Aided Geom. Des. 22,* 2, 121–146. Conference version: Symp. on Geometry Processing 2003.

Cazals, F. and Pouget, M. 2005b. Topology driven algorithms for ridge extraction on meshes. Rapport de recherche 5526, INRIA.

Cohen-Steiner, D. and Morvan, J.-M. 2003. Restricted delaunay triangulations and normal cycle. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry.* ACM, New York, NY, USA, 312–321.

de Carmo, M. 1976. *Differential Geometry of Curves and Surfaces.* Prentice Hall, Englewood Cliffs, NJ.

Golub, G. and van Loan, C. 1983. *Matrix Computations.* Johns Hopkins Univ. Press, Baltimore, MA.

Meyer, M., Desbrun, M., Schröder, P., and Barr, A. H. 2003. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, H.-C. Hege and K. Polthier, Eds. Springer-Verlag, Heidelberg, 35–57.

Petitjean, S. 2002. A survey of methods for recovering quadrics in triangle meshes. *ACM Computing Surveys 34,* 2, 1–61.

Pinkall, U. and Polthier, K. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics 2,* 1, 15–36.

...