Journal of Computational and Applied Mathematics 485 (2026) 117569

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

Introducing the Taubin-Weingarten algorithm to compute second-order geometric descriptors in 3D point clouds

Alberto M. Esmorís [a,b,*], Xabier García-Martínez [a], Manuel Ladra [a], José Carlos Cabaleiro [c,d], Francisco Fernández Rivera [c,d]

[a] Departamento de Matemáticas & CITMAga, Universidade de Santiago de Compostela, Rúa Lope Gómez de Marzoa, Santiago de Compostela, A Coruña, 15782, Spain
[b] 3DGeo Research Group, Institute of Geography, Heidelberg University, Im Neuenheimer Feld 368, Heidelberg, Baden-Württemberg, 69120, Germany
[c] Centro Singular de Investigación en Tecnoloxías Intelixentes, Universidade de Santiago de Compostela, Rúa de Jenaro de la Fuente Domínguez, Santiago de Compostela, A Coruña, 15782, Spain
[d] Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Rúa Lope Gómez de Marzoa, Santiago de Compostela, A Coruña, 15782, Spain

ARTICLE INFO

Keywords:
Algorithms
Geometric descriptors
3D point clouds
Semantic segmentation
Machine learning

ABSTRACT

We propose the Taubin-Weingarten algorithm to compute second-order geometric features in 3D point clouds. This method is well-suited for working with large-scale 3D datasets due to its embarrassingly parallel nature. The speedup of our open-source C++ implementation is systematically above 30 for a high-performance CPU with 32 cores. It provides reliable estimates for standard quantifications in differential geometry, such as the Gaussian and mean curvatures of a surface, when compared to other approaches. Additionally, it achieves improvements of between 1.6% and 10% in F1-score compared to the standard first-order approach in point-wise classification tasks, such as object part segmentation and large-scale semantic scene segmentation. The results demonstrate that the Taubin-Weingarten algorithm is both efficient and robust, enabling consistent improvements in machine learning performance across various tasks.

1. Introduction and context

Three-dimensional point clouds are used in a wide variety of applications in science, engineering, and humanities [1]. The different point cloud processing tasks that arise in these contexts are often governed by the finding of spatial neighborhoods and the computation of geometric features, with the latter being the object of this work.

1.1. Applications

For example, airborne laser scanning (ALS) point clouds are used for above-ground biomass (AGB) estimation in forests [2]. They are also used for automatic forest inventory through semantic and instance segmentation followed by characterization of individual trees in terms of height, diameter and volume of the crown, diameter at breast height (DBH), and georeferenced position [3]. When ALS point clouds are extended with terrestrial laser scanning (TLS) data, they can be used to achieve high-quality estimations of

* Corresponding author.
E-mail address: alberto.esmoris.pena@usc.es (A.M. Esmorís).

https://doi.org/10.1016/j.cam.2026.117569
Received 29 September 2025; Received in revised form 22 February 2026
Available online 17 March 2026

tree canopy volume [4]. In precision agriculture, 3D point clouds enable the automatic spraying of orchards with robots [5]. For social sciences and humanities, some example applications are the generation of Building Information Modeling (BIM), including structural details and characterization of the materials for cultural heritage conservation [6], and semiautomatic archaeological prospection guided by expert knowledge to discover new archaeological sites even in heterogeneous geographical contexts with complex vegetation distributions [7].

In urban environments, applications of 3D point clouds range from automatic urban tree species classification with unmanned laser scanning (ULS) [8] to accurate characterization of crossing zones with mobile laser scanning (MLS) [9]. They also include damage volumetric assessment of walls and columns in 3D digital twins [10] and damage assessment of critical infrastructures, such as sewer pipes [11]. Applications in industrial contexts include impact deformation analysis of thin-walled tubes [12] and accurate estimation of the load-carrying capacity of beams from 3D digital twins [13]. Furthermore, when noise and outlier objects are properly handled, 3D point clouds can be used to derive uniform meshes yielding a spatially accurate representation of mining excavations [14] or to assist the generation of high-precision digital twins of shield tunnels [15].

1.2. Definition and geometric features for machine learning

While applications of point clouds span diverse domains, any 3D point cloud can be formally defined as a finite set of points, i.e., $\mathcal{P} = \{\boldsymbol{x}_{1*}, \dots, \boldsymbol{x}_{m*}\} \subset \mathbb{R}^n$. Besides, it can be expressed as a matrix $\boldsymbol{P} \in \mathbb{R}^{m \times n}$ representing $m \in \mathbb{Z}_{>0}$ points. This matrix notation is compatible with the set definition as long as only row permutation invariant operations are considered, i.e., the order of the rows/points does not change the output. For a 3D point cloud, this matrix can be seen as a block matrix $\boldsymbol{P} = [\boldsymbol{X} \mid \boldsymbol{F}]$ where $\boldsymbol{X} \in \mathbb{R}^{m \times 3}$, $\boldsymbol{F} \in \mathbb{R}^{m \times n_f}$, and $n = 3 + n_f$. We say it is a 3D point cloud because its matrix of coordinates $\boldsymbol{X}$ consists of 3D points represented as rows of three coordinates, and we call $\boldsymbol{F}$ the feature space of the point cloud with an arbitrary dimensionality (including zero), i.e., $n_f \in \mathbb{Z}_{\geq 0}$. Examples of typical features are the intensity (i.e., the peak value derived from the digital waveform analysis) or the number of returns per laser pulse [16].

Point-wise classification tasks, including object part and semantic scene segmentation, are required for most applications as the first step to derive useful information from the data. The current approach for general-purpose geometric descriptors in classical machine learning contexts starts by computing the neighborhood of each point and representing them as a matrix of point-wise coordinates. Let us denote the coordinates matrix of the ith neighborhood as $\boldsymbol{X}_i \in \mathbb{R}^{m_i \times 3}$ where typically $i = 1, \dots, m$ for a point cloud of m points. Each neighborhood is represented with $m_i \in \mathbb{Z}_{>0}$ points that belong to the neighborhood of the ith point $\mathcal{N}(\boldsymbol{x}_{i*}) \subset \mathcal{P}$. Then the centroid $\boldsymbol{\mu}_i = m_i^{-1} \sum_{\boldsymbol{x}_{j*} \in \mathcal{N}(\boldsymbol{x}_{i*})} \boldsymbol{x}_{j*}$ is determined to compute the centered version of the neighborhood's coordinates $\widetilde{\boldsymbol{X}}_i$ where each jth row can be expressed as $\widetilde{\boldsymbol{x}}_{j*} = \boldsymbol{x}_{j*} - \boldsymbol{\mu}_i$ for the corresponding $\boldsymbol{x}_{j*} \in \mathcal{N}(\boldsymbol{x}_{i*})$. Finally, the first-order geometric descriptors for the ith point can be derived from the eigenvalues of the covariance matrix $(m_i - 1)^{-1} \widetilde{\boldsymbol{X}}_i^{\intercal} \widetilde{\boldsymbol{X}}_i \in \mathbb{R}^{3 \times 3}$, which is also known as structure tensor analysis. These descriptors were initially proposed by Weinmann et al. [17] and Hackel et al. [18], and they were later computed in multiscale spherical neighborhoods to improve the performance of machine learning models by Thomas H. et al. [19].

First-order geometric descriptors are used in the current state-of-the-art for machine learning-based semantic segmentation of 3D point clouds, and it coexists with deep learning approaches whenever these are not necessary to achieve good performance or there is not enough data to train a neural network for a particular task. On top of that, deep learning models can also be extended with input geometric features to improve their performance. First-order geometric descriptors are still used in many works like the assessment of building damage due to earthquakes from multi-temporal time series of 3D point clouds [20], the domain adaptation of machine learning models trained with simulated point clouds to generalize to real point clouds in natural and urban contexts [21], or the analysis of wind-induced movements for leaf-wood segmentation and tree characterization [22].

1.3. Contribution

In this work, we focus on an algorithm to compute agnostic geometric features, i.e., those that can be computed considering only the (x, y, z) coordinates of each point with no prior constraints or further information. More formally, **we propose the Taubin-Weingarten (TW) algorithm as a geometric feature extraction procedure** that transforms an input matrix of coordinates $\boldsymbol{X}$ to an output point cloud $\widehat{\boldsymbol{P}} = [\boldsymbol{X} \mid \widehat{\boldsymbol{F}}] \in \mathbb{R}^{m \times (3 + \widehat{n}_f)}$ with $\widehat{n}_f \in \mathbb{Z}_{>0}$ features. The main novelty of our approach lies in its ability to work with quadratic equations, enabling the computation of second partial derivatives. This property contrasts with first-order geometric features, which rely on a linear model without second partial derivatives, and allows us to encode curvature information in second-order geometric features. The name of our proposal comes from the two fundamental methods at the core of our algorithm, the Taubin method for quadric fitting [23] and the Weingarten map or shape operator for the computation of principal curvatures in implicit equations [24]. Our algorithm constitutes a novel and significant contribution because

1. It provides reliable **estimations of Gaussian and mean curvature of surfaces** in 3D point clouds.
2. It is an **embarrassingly parallel** algorithm whose performance scales with the number of cores.
3. It improves the results of machine learning models for **3D object part segmentation** compared to typical first-order geometric descriptors.
4. It improves the results of machine learning models for **large-scale 3D semantic segmentation** compared to typical first-order geometric descriptors.

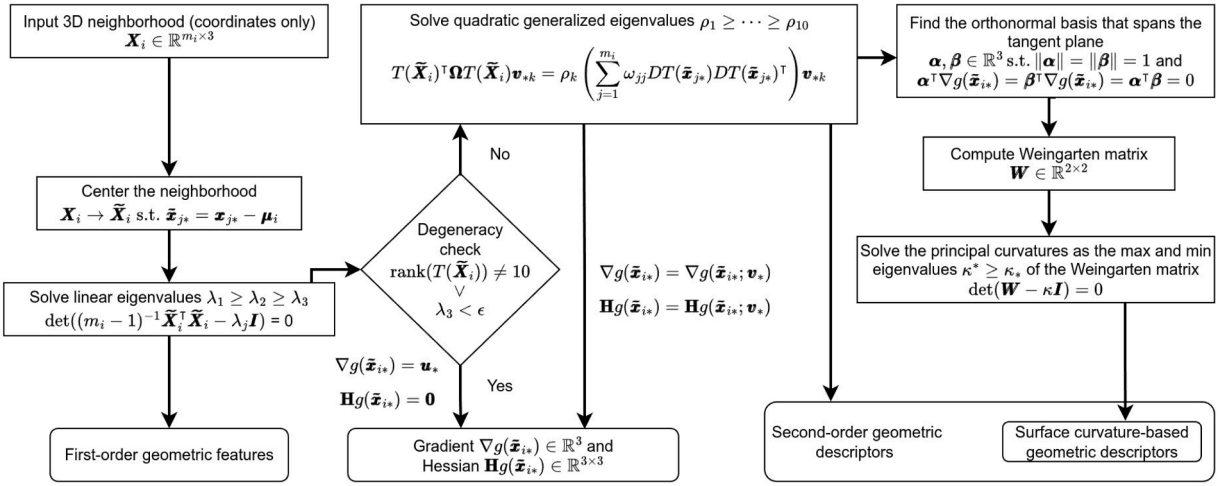

Fig. 1. Flow chart summarizing the Taubin-Weingarten algorithm for a neighborhood of m_i points in a 3D point cloud. Note that $\boldsymbol{\mu}_i \in \mathbb{R}^3$ is the centroid of the neighborhood, $\boldsymbol{u}_* \in \mathbb{R}^3$ is the eigenvector associated with the minimum linear eigenvalue, and $\boldsymbol{v}_* \in \mathbb{R}^{10}$ is the generalized eigenvector associated with the minimum quadratic generalized eigenvalue. The Jacobian of the transformation from coordinates to monomials is denoted by $DT(\tilde{\boldsymbol{x}}_{j*}) \in \mathbb{R}^{10\times 3}$.

Table 1
Summary of the main notation used to describe the Taubin-Weingarten algorithm.

Notation	Description
$\widetilde{\boldsymbol{X}}_i \in \mathbb{R}^{m\times 3}$	Centered 3D structure space matrix representing the geometry of the ith neighborhood
$\lambda_1 \geq \lambda_2 \geq \lambda_3$	Linear eigenvalues, i.e., those of the covariance matrix
$\boldsymbol{u}_{*1}, \boldsymbol{u}_{*2}, \boldsymbol{u}_{*3} \in \mathbb{R}^3$	Linear eigenvectors, i.e., those associated to the linear eigenvalues
$g(\tilde{\boldsymbol{x}}; \boldsymbol{w}) : \mathbb{R}^3 \to \mathbb{R}$	Quadratic model with coefficients $\boldsymbol{w} \in \mathbb{R}^{10}$ (see Eq. (1))
$T(\widetilde{\boldsymbol{X}}_i) : \mathbb{R}^{m\times 3} \to \mathbb{R}^{m\times 10}$	Monomial transform of a centered neighborhood
$\rho_1 \geq \cdots \geq \rho_{10}$	Quadratic eigenvalues, i.e., the generalized eigenvalues after the monomial transform
$\boldsymbol{v}_{1*}, \ldots, \boldsymbol{v}_{10*} \in \mathbb{R}^{10}$	Quadratic eigenvectors, i.e., the eigenvectors associated to the quadratic eigenvalues
$\boldsymbol{W}(\boldsymbol{x}_{i*}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \in \mathbb{R}^{2\times 2}$	Weingarten matrix on an orthonormal tangent basis spanned by $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ at the ith point
$\kappa_*, \kappa^* \in \mathbb{R}$	Minimum and maximum principal curvatures, respectively
$\epsilon \in \mathbb{R}_{>0}$	Eigenthreshold parameter governing the degeneracy check
$\tau \in \mathbb{R}_{>0}$	Tikhonov regularization parameter to lift the generalized eigenvalues

2. Method

In this section, we present a detailed description of the TW algorithm. The flow chart in Fig. 1 summarizes the main steps of the method. Its many components are explained in the following subsections. Note that from now on the point $\boldsymbol{x} \in \mathbb{R}^3$ refers to the point expressed in the original coordinate reference system of the point cloud while the point $\tilde{\boldsymbol{x}} \in \mathbb{R}^3$ refers to the point translated by subtracting the centroid of the neighborhood that is being analyzed (as described in Section 1.2). Table 1 summarizes the main notational elements used to describe the TW algorithm.

2.1. Linear fit and degeneracy check

The first step before the quadric fit is to check whether the input neighborhood is degenerate (i.e., it corresponds almost exactly with a linear variety, typically a plane). These cases can be checked with the standard approach for first-order geometric features. Assuming the eigenvalues are given in descending order (i.e., $\lambda_1 \geq \lambda_2 \geq \lambda_3$), we need to check whether $\lambda_3 < \epsilon$ for a given small $\epsilon \in \mathbb{R}_{>0}$. Note that the minimum eigenvalue may be interpreted as a measure of fitness quality, thus when $\lambda_3 \to 0$ the neighborhood does not represent a geometry with curvature and we shall not proceed to the quadric fit. On top of that, the degeneracy check also includes those cases where the monomials of the quadratic model on the input neighborhood $T(\widetilde{\boldsymbol{X}}_i)$ do not have at least one linearly independent equation for each coefficient. More formally, neighborhoods are degenerate if they have less than ten linearly independent equations on the nine monomials (plus one for the bias term) of the general implicit quadratic equation, i.e., $\operatorname{rank}\left(T(\widetilde{\boldsymbol{X}}_i)\right) < 10$. The monomial transformation is explained in detail in the subsequent Section 2.2.

In degenerate cases we are fitting a plane model $g(\tilde{\boldsymbol{x}}) = \boldsymbol{u}_{*3}^\intercal \tilde{\boldsymbol{x}}$ where $\boldsymbol{u}_{*3} \in \mathbb{R}^3$ is the column eigenvector associated with the minimum eigenvalue. The gradient of the model will be given by the components of the eigenvector, i.e., $\nabla g(\tilde{\boldsymbol{x}}) = \boldsymbol{u}_{*3}$. In this scenario, the gradient will always have unitary norm because the eigenvectors computed through Principal Component Analysis (PCA) yield an orthonormal basis, i.e., $\|\nabla g(\tilde{\boldsymbol{x}})\| = \|\boldsymbol{u}_{*3}\| = 1$ [25, see pp. 461–463]. As the TW algorithm implies solving the standard linear eigenvalue

problem, it can also provide first-order geometric descriptors. This property is an advantage of the algorithm because it prevents the computation of two different methods when both first-order and second-order geometric descriptors must be calculated.

2.2. Quadric fit

If the degeneracy check passed, then a quadratic model must be computed. The rationale behind this is that a linear model $g \in \mathbb{R}[x, y, z]$ satisfies $\deg(g) = 1$, which implies that first-order derivatives will be constant and second-order derivatives will be null. Consequently, it cannot handle curvature information in the data. When the geometric distribution of the input neighborhood cannot be almost perfectly explained with a linear model, we assume it must have some curvature information, and thus we want a non-linear algebraic model satisfying $\deg(g) \geq 2$. In this work, we use a general implicit quadratic equation as defined in Eq. (1). This model has $\deg(g) = 2$ and thus will lead to linear first derivatives and constant second derivatives for a given point. The coefficients of the polynomial are represented with w_k for $k = 1, \dots, 10$. The gradient vector $\nabla g(\widetilde{\boldsymbol{x}}) \in \mathbb{R}^3$ and Hessian matrix $\mathbf{H}g(\widetilde{\boldsymbol{x}}) \in \mathbb{R}^{3\times3}$ will be given by the expressions in Eq. (2). It is worth mentioning that the quadratic model will no longer necessarily satisfy $\|\nabla g(\widetilde{\boldsymbol{x}})\| = 1$, and thus the gradients might have different norms (which already makes the gradient norm a geometric feature characterizing the neighborhood).

$$g(\widetilde{\boldsymbol{x}}; \boldsymbol{w}) = w_1\widetilde{x} + w_2\widetilde{y} + w_3\widetilde{z} + w_4\widetilde{x}\widetilde{y} + w_5\widetilde{x}\widetilde{z} + w_6\widetilde{y}\widetilde{z} + w_7\widetilde{x}^2 + w_8\widetilde{y}^2 + w_9\widetilde{z}^2 + w_{10} \tag{1}$$

$$\nabla g(\widetilde{\boldsymbol{x}}) = \begin{bmatrix} 2w_7\widetilde{x} + w_4\widetilde{y} + w_5\widetilde{z} + w_1 \\ w_4\widetilde{x} + 2w_8\widetilde{y} + w_6\widetilde{z} + w_2 \\ w_5\widetilde{x} + w_6\widetilde{y} + 2w_9\widetilde{z} + w_3 \end{bmatrix}, \quad \mathbf{H}g(\widetilde{\boldsymbol{x}}) = \begin{bmatrix} 2w_7 & w_4 & w_5 \\ w_4 & 2w_8 & w_6 \\ w_5 & w_6 & 2w_9 \end{bmatrix} \tag{2}$$

Now, for the sake of convenience, let us define $T(\widetilde{\boldsymbol{x}}) : \mathbb{R}^3 \to \mathbb{R}^{10}$ as the transformation from a 3D point $\widetilde{\boldsymbol{x}} = (x, y, z)$ to its monomial representation $T(\widetilde{\boldsymbol{x}}) = (x, y, z, xy, xz, yz, x^2, y^2, z^2, 1)$. Analogously, $T(\widetilde{\boldsymbol{X}}) : \mathbb{R}^{m\times3} \to \mathbb{R}^{m\times10}$ would transform each row $\widetilde{\boldsymbol{x}}_{j*}$ of $\widetilde{\boldsymbol{X}}$ to $T(\widetilde{\boldsymbol{x}}_{j*})$. This transformation leads to a straightforward generalization of the first-order method, fitting the quadratic model with PCA by finding the eigenvalues $\rho_1 \geq \cdots \geq \rho_{10} \in \mathbb{R}_{>0}$ and eigenvectors $\boldsymbol{v}_{*1}, \dots, \boldsymbol{v}_{*10} \in \mathbb{R}^{10}$ of $(m-1)^{-1}T(\widetilde{\boldsymbol{X}})^\intercal T(\widetilde{\boldsymbol{X}})$. Since the quadratic model is more complicated than the linear one, we could explore a numerically more robust alternative like Singular Value Decomposition (SVD) [26] [27, see pp. 624–638]. In that case, we could decompose the transformed matrix of coordinates such that $T(\widetilde{\boldsymbol{X}}) = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal$. Assuming the singular values are in descending order $\sigma_1 \geq \cdots \geq \sigma_{10}$, the coefficients of the quadratic model would be given by the non-trivial singular vector associated with the smallest singular value $\boldsymbol{v}_* \in \mathbb{R}^{10}$. The geometric features would be identical to those acquired by PCA if we take the squares of the singular values because $\rho_k = \sigma_k^2$.

Unfortunately, these straightforward approaches are problematic as we will show with the theoretical error quantifications in Section 4.1. The main problem is that surfaces with curvature fit through SVD or PCA are simply minimizing the sum of squared algebraic distances that do not generally correspond with the geometric distances. Note that if the model g is a plane with coefficients $\boldsymbol{w} = (w_1, w_2, w_3)$, then the geometric distance for any point $\widetilde{\boldsymbol{x}}$ is given by $\|\boldsymbol{w}\|^{-1}|\boldsymbol{w}^\intercal\widetilde{\boldsymbol{x}}|$. Hence, for the particular case of normalized planes with $\|\boldsymbol{w}\| = 1$, the geometric distance matches the absolute algebraic distance because it reduces to $|\boldsymbol{w}^\intercal\widetilde{\boldsymbol{x}}|$ (which happens for PCA and SVD when fitting a plane). This is no longer the case for a quadratic model. One alternative is to consider a non-linear optimization algorithm like gradient descent [28, see pp. 466–475] or Levenberg-Marquardt [29] [30, see pp. 801–806] and modify it to minimize the geometric distance or extend it to include regularization terms to reduce the unwanted bias of algebraic distance minimization. However, dense 3D point clouds in real applications often present millions of neighborhoods with hundreds, thousands, or even tens of thousands of points each. Therefore, computing an iterative non-linear optimization method is likely to lead to a prohibitive computational cost. Instead, we need an algebraic method that does not require many iterations to converge and yet allows us to reduce the bias of minimizing the algebraic distance by applying some computationally cheap constraint.

The work of Taubin proposed a method for fitting planar curves, polynomial surfaces, and algebraic space curves represented by implicit equations [23]. It consists of a smart initialization method based on solving a generalized eigenvalue problem to achieve a good initial fit that minimizes the mean square error, constrained by the mean of the outer products of the gradient evaluated at each point of the dataset. The main benefit of the constraint is reducing the bias of a straightforward algebraic fit that minimizes the squared algebraic distances. This initial fit is then refined with a few iterations of Levenberg-Marquardt. In our work, we consider only the initialization method as a good-enough approach and avoid the computational cost of the Levenberg-Marquardt algorithm or any other iterative process.

The constraint of the initial fit method in [23] (from now on called Taubin's method) can be expressed in our case as the mean of $\nabla g(\widetilde{\boldsymbol{x}}_{j*}; \boldsymbol{w})\nabla g(\widetilde{\boldsymbol{x}}_{j*}; \boldsymbol{w})^\intercal$ considering each point in the neighborhood $\widetilde{\boldsymbol{x}}_{j*} \in \mathcal{N}(\boldsymbol{x}_{i*})$. Although $\boldsymbol{w}$ is not known before the fit, the constraint can still be incorporated. Let us start by noting that $\nabla g(\widetilde{\boldsymbol{x}}_{j*}; \boldsymbol{w}) = \nabla(T(\widetilde{\boldsymbol{x}}_{j*})^\intercal \boldsymbol{w})$. Consequently, as differentiation is a linear operation, $\nabla g(\widetilde{\boldsymbol{x}}_{j*}; \boldsymbol{w}) = D(T(\widetilde{\boldsymbol{x}}_{j*}))^\intercal \boldsymbol{w}$. Note that $D(T(\widetilde{\boldsymbol{x}}_{j*})) \in \mathbb{R}^{10\times3}$ is the Jacobian evaluated on the jth point of the neighborhood and let $\boldsymbol{N}(\boldsymbol{x}_{i*}) \in \mathbb{R}^{10\times10}$ be the mean of the outer products of the Jacobians for each point in the neighborhood as expressed in Eq. (3).

$$D(T(\widetilde{\boldsymbol{x}}_{j*}))^\intercal = \begin{bmatrix} 1 & 0 & 0 & y & z & 0 & 2x & 0 & 0 & 0 \\ 0 & 1 & 0 & x & 0 & z & 0 & 2y & 0 & 0 \\ 0 & 0 & 1 & 0 & x & y & 0 & 0 & 2z & 0 \end{bmatrix}, \quad \boldsymbol{N}(\boldsymbol{x}_{i*}) = m_i^{-1} \sum_{j=1}^{m_i} D(T(\widetilde{\boldsymbol{x}}_{j*}))D(T(\widetilde{\boldsymbol{x}}_{j*}))^\intercal \tag{3}$$

For the sake of simplicity, let us also define $\boldsymbol{M}(\boldsymbol{x}_{i*}) = m_i^{-1}T(\widetilde{\boldsymbol{X}}_i)^\intercal T(\widetilde{\boldsymbol{X}}_i)$ as the second moment matrix of the neighborhood. The typical approach is to impose a constraint $\boldsymbol{w}^\intercal\boldsymbol{N}(\boldsymbol{x}_{i*})\boldsymbol{w} \neq 0$ to rule out the trivial solution induced by the bias term. This strategy leads to

the generalized eigenvalue problem $\boldsymbol{M}(\boldsymbol{x}_{i*})\boldsymbol{v} = \rho\boldsymbol{N}(\boldsymbol{x}_{i*})\boldsymbol{v}$, whose minimal eigenvector is often normalized to solve the optimization problem given in Eq. (4). With our approach, assuming that the generalized eigenvalues are in descending order ($\rho_1 \geq \cdots \geq \rho_{10}$), the components of the generalized eigenvector $\boldsymbol{v}_{\xi*}$ associated with the minimum generalized eigenvalue give the coefficients $w_1, \ldots, w_{10}$ of the quadratic polynomial. The index of the target eigenvalue is determined as the index of the minimum reliable generalized eigenvalue $\xi = \arg\min_{2\leq k\leq 10} \quad \{\rho_k : \rho_k > \tau\}$, where $\tau \in \mathbb{R}_{>0}$ corresponds with the Tikhonov regularization parameter [25, see pp. 321–328]. Note that ρ_1 is not considered because the quadratic model includes a bias term, which is expected to be associated with the trivial eigenvector. Hence, in our case, we can ignore the maximal eigenvalue without explicitly imposing any constraint on $\boldsymbol{w}^\intercal(\boldsymbol{N} + \tau\boldsymbol{I})\boldsymbol{w}$ to avoid trivial solutions.

$$\begin{aligned} \min_{\boldsymbol{w}\in\mathbb{R}^{10}} \quad & \boldsymbol{w}^\intercal\boldsymbol{M}(\boldsymbol{x}_{i*})\boldsymbol{w} \\ \text{s.t.} \quad & \boldsymbol{w}^\intercal\boldsymbol{N}(\boldsymbol{x}_{i*})\boldsymbol{w} = 1 \end{aligned} \tag{4}$$

To understand why we expect the maximal generalized eigenvalue ρ_1 to correspond with the trivial eigenvector, let us introduce the generalized Rayleigh quotient [25, see pp. 474–484] of our model for any non-null vector $\boldsymbol{w} \in \mathbb{R}^{10}$ in Eq. (5). Note that for any trivial solution vector $\boldsymbol{w}_* = (0, \ldots, 0, b)$ with $b \neq 0$ we have $\boldsymbol{w}_*^\intercal\boldsymbol{M}\boldsymbol{w}_* = b^2$ and $\boldsymbol{w}_*^\intercal(\boldsymbol{N} + \tau\boldsymbol{I})\boldsymbol{w}_* = \tau b^2$. Therefore, $R(\boldsymbol{w}_*) = \tau^{-1}$. Now consider any non-trivial vector $\boldsymbol{w}^* \neq \alpha\boldsymbol{w}_*$ with $\alpha \in \mathbb{R}$. Looking at the denominator of $R(\boldsymbol{w}^*)$, we can see that when $\tau \to 0$ the non-trivial generalized eigenvalues with Tikhonov regularization converge to the standard non-trivial generalized eigenvalues because $\tau(\boldsymbol{w}^*)^\intercal\boldsymbol{w}^* \to 0$. To the contrary, when $\tau \to 0$ we have that $R(\boldsymbol{w}_*) \to \infty$. From this situation, we can infer that, by choosing a small enough τ, the maximum generalized eigenvalue (which is also the maximum of the generalized Rayleigh quotient) will correspond to the eigenvector of the trivial solution. Numerically speaking, we recommend choosing τ close enough to the machine epsilon for the number of bits employed in the decimal representations with finite precision.

$$R(\boldsymbol{w}) = \frac{\boldsymbol{w}^\intercal\boldsymbol{M}\boldsymbol{w}}{\boldsymbol{w}^\intercal(\boldsymbol{N} + \tau\boldsymbol{I})\boldsymbol{w}} = \frac{\boldsymbol{w}^\intercal\boldsymbol{M}\boldsymbol{w}}{\boldsymbol{w}^\intercal\boldsymbol{N}\boldsymbol{w} + \tau\boldsymbol{w}^\intercal\boldsymbol{w}} \tag{5}$$

Algorithm 1 describes the implementation of Taubin's method in the TW algorithm. The matrix $\boldsymbol{\Omega} \in \mathbb{R}^{m_i\times m_i}$ is a diagonal matrix of point-wise weights that can be used to govern the contribution of each point to the quadric fit. Note that the $\boldsymbol{M}$ and $\boldsymbol{N}$ matrices are generalized to work with weights. More concretely, the matrix $\boldsymbol{N}$ generalizes Eq. (3) by including the point-wise weights in the diagonal of $\boldsymbol{\Omega}$ and the Tikhonov regularization with parameter $\tau \in \mathbb{R}_{>0}$ to avoid numerical issues where the aggregation of outer Jacobian products fails to be positive definite.

Algorithm 1 Taubin's method for the neighborhood of a 3D point cloud.

Require: $\text{rank}\left(T(\widetilde{\boldsymbol{X}}_i)\right) = 10$ ▷ Require 10 linearly independent equations on the monomials

1: **function** TAUBIN_FIT($\widetilde{\boldsymbol{X}}_i \in \mathbb{R}^{m_i\times 3}, \boldsymbol{\Omega} \in \mathbb{R}^{m_i\times m_i}, \tau \in \mathbb{R}_{>0}$)
2: $\boldsymbol{M} = T(\widetilde{\boldsymbol{X}}_i)^\intercal\boldsymbol{\Omega}T(\widetilde{\boldsymbol{X}}_i)$ ▷ Weighted second moment matrix
3: $\boldsymbol{N} = \tau\boldsymbol{I} + \sum_{j=1}^{m_i}\omega_{jj}DT(\widetilde{\boldsymbol{x}}_{j*})DT(\widetilde{\boldsymbol{x}}_{j*})^\intercal$ ▷ Tikhonov regularized weighted mean of outer Jacobian products
4: $\boldsymbol{L}\boldsymbol{L}^\intercal = \boldsymbol{N}$ ▷ Cholesky decomposition of $\boldsymbol{N}$
5: $\rho_1, \ldots, \rho_{10} = \text{solve}\left(\det\left(\boldsymbol{L}^{-1}\boldsymbol{M} - \rho\boldsymbol{I}\right) = 0\right)$ ▷ Find generalized eigenvalues
6: $\boldsymbol{V}' = \text{solve}\left(\left(\boldsymbol{L}^{-1}\boldsymbol{M} - \rho_k\boldsymbol{I}\right)\boldsymbol{v}'_{*k} = \boldsymbol{0}\right)$ ▷ Find eigenvectors as column vectors
7: $\boldsymbol{V} = (\boldsymbol{L}^\intercal)^{-1}\boldsymbol{V}'$ ▷ Change the basis of the eigenvectors to obtain the generalized eigenvectors
8: **return** $\rho_1, \ldots, \rho_{10}, \boldsymbol{V}$ ▷ Return generalized eigenvalues and eigenvectors
9: **end function**

We exploit the fact that the regularized $\boldsymbol{N}$ is symmetric $\boldsymbol{N} = \boldsymbol{N}^\intercal$ and positive definite $\forall\boldsymbol{v} \in \mathbb{R}^{10}$, $\boldsymbol{v}^\intercal\boldsymbol{N}\boldsymbol{v} > 0$ to solve the generalized eigenvalue problem through its Cholesky decomposition into lower $\boldsymbol{L} \in \mathbb{R}^{10\times 10}$ and upper $\boldsymbol{L}^\intercal \in \mathbb{R}^{10\times 10}$ triangular matrices. The Cholesky decomposition allows us to solve the problem simply by forward-backward substitution and the eigen-decomposition of a 10×10 matrix. This method is typically faster than other common approaches like computing the eigen-decomposition or SVD of $\boldsymbol{N}^{-1/2}\boldsymbol{M}\boldsymbol{N}^{-1/2}$, or even the generalized Schur Decomposition (QZ algorithm) of the matrix pencil $(\boldsymbol{M}, \boldsymbol{N})$ [31].

Concerning the matrix $\boldsymbol{\Omega}$, on the one hand, it can be set to $\omega_{jj} = m_i^{-1}$ to use the standard unweighted mean. On the other hand, it can be computed with methods like inverse distance weighting (IDW) or Gaussian radial basis functions (RBF) [32, see pp. 159–162] to increase the contribution of the points that are closer to the center of the neighborhood. Finally, recall that the coefficients of the quadratic polynomial will be given by the ξth column of $\boldsymbol{V}$, i.e., $\boldsymbol{v}_{*\xi}$.

2.3. Principal curvatures

Let $\kappa^*, \kappa_* \in \mathbb{R}$ be the maximum and minimum normal curvatures among the many curves that lie on a surface S, i.e., its principal curvatures [33, see pp. 188–198]. Let us also introduce the Gauss map between a surface and the unit sphere $N : S \to \mathbb{S}^2$ [33, see pp. 172–178]. For our quadratic polynomial model, the Gauss map is $N(\boldsymbol{x}_{i*}) = \|\nabla g(\widetilde{\boldsymbol{x}}_{i*})\|^{-1}\nabla g(\widetilde{\boldsymbol{x}}_{i*})$. Now we can introduce the Weingarten map (also known as shape operator) as the negative of the directional derivative along an arbitrary vector $\boldsymbol{u} \in \mathbb{R}^3$ of the Gauss map, i.e., $W(\boldsymbol{x}_{i*}, \boldsymbol{u}) = -D_{\boldsymbol{u}}N(\boldsymbol{x}_{i*})$ [24]. This map can be used to compute the Weingarten matrix $\boldsymbol{W}(\boldsymbol{x}_{i*}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \in \mathbb{R}^{2\times 2}$ which will allow us to find the principal curvatures of our implicit quadratic model.

To compute the Weingarten matrix we need an orthonormal basis of two vectors $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{R}^3$ that spans the tangent plane to the surface defined by g passing through the point $\widetilde{\boldsymbol{x}}_{i*}$. These vectors can be computed by Algorithm 2, provided we have a non-null gradient at $\widetilde{\boldsymbol{x}}_{i*}$. In doing so, selecting the component of the normalized gradient whose absolute value is the closest to one prevents divisions by tiny numbers when finding $\boldsymbol{\alpha}$, which helps with numerical stability.

Algorithm 2 Method to find an orthonormal basis spanning the tangent plane of g through $\widetilde{\boldsymbol{x}}_{i*}$.

Require: $\nabla g(\widetilde{\boldsymbol{x}}_{i*}) \neq \boldsymbol{0}$ ▷ Gradient must not be null
function TANGENT_PLANE_BASIS($\nabla g(\widetilde{\boldsymbol{x}}_{i*}) \in \mathbb{R}^3$)
  $\boldsymbol{u} = \|\nabla g(\widetilde{\boldsymbol{x}}_{i*})\|^{-1} \nabla g(\widetilde{\boldsymbol{x}}_{i*})$ ▷ Normalize gradient vector
  $\boldsymbol{\alpha} = (1, 1, 1)$ ▷ Initialize $\boldsymbol{\alpha}$ as a vector of ones
  $k = \arg\max_{1 \leq j \leq 3} \left\{ \left| u_j \right| \right\}$ ▷ Find index of absolute component of $\boldsymbol{u}$ closest to one
  $\alpha_k = -u_k^{-1} \sum_{j \neq k} u_j$ ▷ Compute α_k so $\boldsymbol{\alpha}^\intercal \nabla g(\widetilde{\boldsymbol{x}}_{i*})) = 0$
  $\boldsymbol{\alpha} = \|\boldsymbol{\alpha}\|^{-1} \boldsymbol{\alpha}$ ▷ Normalize $\boldsymbol{\alpha}$
  $\boldsymbol{\beta} = \boldsymbol{u} \times \boldsymbol{\alpha}$ ▷ Compute third orthonormal vector by cross product
  return $\boldsymbol{\alpha}, \boldsymbol{\beta}$ ▷ Return orthonormal vectors spanning the tangent plane
end function

Given the pair of orthonormal vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}$ spanning the tangent plane, we can use the Weingarten map from Eq. (6) to compute the Weingarten matrix in Eq. (7). Note that the simplification of the Weingarten matrix is justified because the vectors that span the tangent plane are orthogonal with respect to the gradient, i.e., $\boldsymbol{\alpha}^\intercal \nabla g(\widetilde{\boldsymbol{x}}_{i*}) = \boldsymbol{\beta}^\intercal \nabla g(\widetilde{\boldsymbol{x}}_{i*}) = 0$. Consequently, we only need to consider the $-\mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*})\boldsymbol{u}$ term of the parenthesis in the last line of Eq. (6). Once the Weingarten matrix is computed, the principal curvatures can be derived from its eigenvalues. Fortunately, the roots of the characteristic polynomial of the 2×2 Weingarten matrix reduce to a second-degree algebraic equation whose set of solutions $\mathcal{W}$ can be computed from the closed-form solution in Eq. (8), where w_{pq} refers to the element at the pth row and qth column of the Weingarten matrix. Finally, the maximum and minimum principal curvatures are given by $\kappa^* = \max\ \mathcal{W}$ and $\kappa_* = \min\ \mathcal{W}$, respectively.

$$\begin{aligned} W(\boldsymbol{x}_{i*}, \boldsymbol{u}) &= -\|\nabla g(\widetilde{\boldsymbol{x}}_{i*})\|^{-1} D_{\boldsymbol{u}}(\nabla g(\widetilde{\boldsymbol{x}}_{i*})) - D_{\boldsymbol{u}}\left(\|\nabla g(\widetilde{\boldsymbol{x}}_{i*})\|^{-1}\right) \nabla g(\widetilde{\boldsymbol{x}}_{i*}) \\ &= \|\nabla g(\widetilde{\boldsymbol{x}}_{i*})\|^{-1} \left(\frac{\nabla g(\widetilde{\boldsymbol{x}}_{i*})^\intercal \mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*})\boldsymbol{u}}{\|\nabla g(\widetilde{\boldsymbol{x}}_{i*})\|^2} \nabla g(\widetilde{\boldsymbol{x}}_{i*}) - \mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*})\boldsymbol{u} \right) \end{aligned} \tag{6}$$

$$\boldsymbol{W}(\boldsymbol{x}_{i*}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \begin{bmatrix} \boldsymbol{\alpha}^\intercal W(\boldsymbol{x}_{i*}, \boldsymbol{\alpha}) & \boldsymbol{\alpha}^\intercal W(\boldsymbol{x}_{i*}, \boldsymbol{\beta}) \\ \boldsymbol{\beta}^\intercal W(\boldsymbol{x}_{i*}, \boldsymbol{\alpha}) & \boldsymbol{\beta}^\intercal W(\boldsymbol{x}_{i*}, \boldsymbol{\beta}) \end{bmatrix} = -\|\nabla g(\widetilde{\boldsymbol{x}}_{i*})\|^{-1} \begin{bmatrix} \boldsymbol{\alpha}^\intercal \mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*})\boldsymbol{\alpha} & \boldsymbol{\alpha}^\intercal \mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*})\boldsymbol{\beta} \\ \boldsymbol{\beta}^\intercal \mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*})\boldsymbol{\alpha} & \boldsymbol{\beta}^\intercal \mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*})\boldsymbol{\beta} \end{bmatrix} \tag{7}$$

$$\mathcal{W} = \left\{ \frac{1}{2} \left(w_{11} + w_{22} \pm \sqrt{(w_{11} + w_{22})^2 - 4(w_{11}w_{22} - w_{12}w_{21})} \right) \right\} \tag{8}$$

2.4. Taubin-Weingarten algorithm

The Taubin-Weingarten algorithm for a neighborhood of a 3D point cloud is formally described in Algorithm 3. It can process any input neighborhood whose points lead to at least three linearly independent equations, i.e., there are at least three points for which there is not a single colinear pair. For the experiments in this work, we set the degeneracy threshold to $\epsilon = 10^{-5}$ and the Tikhonov regularization parameter to $\tau = 10^{-7}$ using floating point arithmetic with 32 bits.

The TW algorithm computes the linear eigenvalues $\lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}$ and column eigenvectors $\boldsymbol{U} \in \mathbb{R}^{3\times3}$, the quadratic generalized eigenvalues $\rho_1, \ldots, \rho_{10} \in \mathbb{R}$ and column eigenvectors $\boldsymbol{V} \in \mathbb{R}^{10\times10}$, the principal curvatures $\kappa^*, \kappa_* \in \mathbb{R}$, the gradient $\nabla g(\widetilde{\boldsymbol{x}}_{i*}) \in \mathbb{R}^3$, and the Hessian $\mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*}) \in \mathbb{R}^{3\times3}$. These values can be used to compute a set of geometric features as explained in Section 2.5. Note that if the degeneracy check is not passed, the quadratic generalized eigenvalues and eigenvectors, the principal curvatures, and the Hessian will be null, with the gradient matching the unitary eigenvector $\boldsymbol{u}_{*3}$ associated with the minimum eigenvalue of the linear model. Even if the quadratic model is fit successfully, it is possible (though not often the case) that the gradient is singular at the center point of the neighborhood $\widetilde{\boldsymbol{x}}_{i*}$. If this is the case, the principal curvatures are set to zero for convenience as they cannot be estimated at singular points.

One could question why the Tikhonov regularization and finding the index ξ are necessary if there is already a check on the rank of $T(\widetilde{\boldsymbol{X}})$. The reason is that we want our model to be robust enough to work on real datasets that often involve problematic scenarios. As an example, let us consider a geometry where the points are taken as samples from one-dimensional subspaces of $\mathbb{R}^{10}$. Let $L_k = \{\gamma \boldsymbol{d}_k : \gamma \in \mathbb{R}\}$ represent the kth subspace, where $\boldsymbol{d}_k \in \mathbb{R}^{10}$ is the direction vector. Now, let $\varphi(L_p, L_q) = \arccos(\|\boldsymbol{d}_p\|^{-1} \|\boldsymbol{d}_q\|^{-1} |\boldsymbol{d}_p^\intercal \boldsymbol{d}_q|) \in [0, \pi/2] \subset \mathbb{R}$ be a function that gives the angular difference between a pair of subspaces L_p and L_q. We might have 10 points from 10 different subspaces $L_1, \ldots, L_{10}$ with linearly independent direction vectors. However, for a given pair of subspaces satisfying $\varphi(L_p, L_q) \to 0$, Algorithm 1 may fail due to numerical limitations when dealing with nearly linearly dependent vectors. The Tikhonov regularization lifts the eigenvalues of the Gram matrix (including the zeroes), thereby making it numerically invertible. However, generalized eigenvalues below this threshold have been artificially lifted and should not be considered as a reliable answer. That is

Algorithm 3 Taubin-Weingarten algorithm for the neighborhood of a 3D point cloud.

Require: $\widetilde{\boldsymbol{X}}_i \in \mathbb{R}^{m_i \times 3}$ s.t rank $\left(\widetilde{\boldsymbol{X}}_i\right) = 3, \epsilon \in \mathbb{R}_{>0}, \tau \in \mathbb{R}_{>0}$

$\lambda_1, \lambda_2, \lambda_3 = \text{solve}\left(\det\left((m-1)^{-1}\widetilde{\boldsymbol{X}}_i^{\intercal}\widetilde{\boldsymbol{X}}_i\right) = 0\right)$ ▷ Eigenvalues of linear model

$\boldsymbol{U} = \text{solve}\left(\left((m-1)^{-1}\widetilde{\boldsymbol{X}}_i^{\intercal}\widetilde{\boldsymbol{X}}_i\right)\boldsymbol{u}_{*k} = \boldsymbol{0}\right)$ ▷ Column eigenvectors of linear model

if $\lambda_3 < \epsilon \vee \text{rank}\left(T(\widetilde{\boldsymbol{X}}_i)\right) < 10$ **then** ▷ Degeneracy check

  return $\lambda_1, \lambda_2, \lambda_3, \boldsymbol{U}, 0, \ldots, 0, \boldsymbol{0}, 0, 0, \boldsymbol{u}_{*3}, \boldsymbol{0}$

end if

$\boldsymbol{\Omega} = \text{POINT\_WISE\_WEIGHTS}(\widetilde{\boldsymbol{X}}_i)$ ▷ Weight the contribution of each point

$\boldsymbol{\mu} = \sum_{j=1}^{m_i} \omega_{jj}\widetilde{\boldsymbol{x}}_{j*}$ ▷ Update centroid to weighted centroid

$\forall 1 \leq j \leq m_i,\ \widetilde{\boldsymbol{x}}_{j*} = \widetilde{\boldsymbol{x}}_{j*} - \boldsymbol{\mu}$ ▷ Recenter neighborhood with weighted centroid

$\rho_1, \ldots, \rho_{10}, \boldsymbol{V} = \text{TAUBIN\_FIT}(\widetilde{\boldsymbol{X}}_i, \boldsymbol{\Omega}, \tau)$ ▷ Fit quadric (see Algorithm 1)

if $\nabla g(\widetilde{\boldsymbol{x}}_{i*}) = \boldsymbol{0}$ **then** ▷ Gradient singularity check

  return $\lambda_1, \lambda_2, \lambda_3, \boldsymbol{U}, \rho_1, \ldots, \rho_{10}, \boldsymbol{V}, 0, 0, \boldsymbol{0}, \mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*}; \boldsymbol{v}_{*\xi})$

end if

$\xi = \arg\min_{2 \leq k \leq 10} \{\rho_k : \rho_k > \tau\}$ ▷ Index of the generalized eigenvector with the polynomial's coefficients

$\boldsymbol{\alpha}, \boldsymbol{\beta} = \text{TANGENT\_PLANE\_BASIS}(\nabla g(\widetilde{\boldsymbol{x}}_{i*}; \boldsymbol{v}_{*\xi}))$ ▷ Tangent space for Weingarten matrix (see Algorithm 2)

$\mathcal{W} = \text{solve}\left(\det\left(\boldsymbol{W}(\boldsymbol{x}_{i*}, \boldsymbol{\alpha}, \boldsymbol{\beta}) - \kappa\boldsymbol{I}\right) = 0\right)$ ▷ Eigenvalues of Weingarten matrix (see Eqs. (6)–(8))

$\kappa^* = \max\ \mathcal{W}$ ▷ Maximum principal curvature

$\kappa_* = \min\ \mathcal{W}$ ▷ Minimum principal curvature

return $\lambda_1, \lambda_2, \lambda_3, \boldsymbol{U}, \rho_1, \ldots, \rho_{10}, \boldsymbol{V}, \kappa^*, \kappa_*, \nabla g(\widetilde{\boldsymbol{x}}_{i*}; \boldsymbol{v}_{*\xi}), \mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*}; \boldsymbol{v}_{*\xi})$

why, despite the rank check, the TW algorithm applies Tikhonov regularization and includes an explicit search process to determine the index of the smallest reliable generalized eigenvalue.

Finally, note that if the `POINT_WISE_WEIGHTS` function in Algorithm 3 is used to compute custom weights (e.g., through RBF or IDW), the centroid must be updated and the neighborhood recentered by subtracting the new weighted centroid. Note that if the weights correspond to a standard mean, the new centroid will be zero because the neighborhood was already centered. Consequently, the recentering of the neighborhood's coordinates will involve a translation by the null vector, i.e., the coordinates will remain the same.

2.5. Second-order geometric descriptors

Computing Algorithm 3 enables the extraction of several geometric features directly from its output. In this section we detail many of these features, categorizing them into eigenvalue features (those that involve the generalized eigenvalues of the quadric fit), polynomial features (those that involve the straightforward evaluation or coefficients of the polynomial), differential features (those computed from the gradient and/or the Hessian of the quadratic model), and curvature features (those derived from the principal curvatures).

The eigenvalue features described in Table 2 use the generalized eigenvalues to describe the geometry of the neighborhood. Some of them, like the quadratic deviation, eigensum, eigenentropy, omnivariance, hypersphericity, and hyperanisotropy, are straightforward generalizations of typical first-order geometric descriptors but computed with the generalized eigenvalues of the quadratic model $\rho_2, \ldots, \rho_{10}$ instead of the eigenvalues of the linear model $\lambda_1, \lambda_2, \lambda_3$. Recall that ρ_2 is considered as the maximum generalized eigenvalue because we have a bias term in our model, which leads to a trivial ρ_1 eigenvalue. For the sake of understanding, let us clarify that the quadratic deviation generalizes the surface variation, while the hypersphericity and hyperanisotropy generalize the sphericity and anisotropy, respectively. The remaining eigenvalue features can be seen as the norms of the system. The squared spectral norm is the maximum generalized eigenvalue, which is the square of the corresponding maximum singular value (that corresponds to the square roots of the eigenvalues). The Frobenius norm is the square root of the summation of the squares of the elements in the matrix of monomials $T(\widetilde{\boldsymbol{X}}_i)$, which matches the square root of the sum of its squared singular values, which correspond to the eigenvalues of its Gram matrix. The Schatten feature refers to the Schatten norm $\|\boldsymbol{M}\|_p = \left(\sum_{k=2}^{\xi} \sigma_k^p\right)^{1/p}$ for $p = 1$ where σ_k is the kth singular value satisfying $\sigma_k = \sqrt{\rho_k}$, leading to $\|\boldsymbol{M}\|_{p=1} = \sum_{k=2}^{\xi} \sqrt{\rho_k}$. In the equation of the normalized eigenentropy $\widehat{\rho}_k = (\rho_2 + \cdots + \rho_\xi)^{-1}\rho_k$.

The polynomial features described in Table 2 involve the polynomial evaluation and coefficients to characterize the geometry of the neighborhood. The absolute and squared algebraic distances can be used to measure how well the neighborhood fits a general implicit quadratic polynomial model. On top of that, the norms involving the coefficients of the monomials (which include the absolute value of the bias term) characterize the neighborhood by quantifying the contribution of the different subsets of the monomials (e.g., those of the linear terms or the squares) to the polynomial model.

The differential features described in Table 3 involve the gradient and the Hessian of the quadratic polynomial model. Note that many of these features can be computed as closed-form expressions directly from the coefficients of the fit polynomial. For example, the Laplacian is simply $2(v_{7\xi} + v_{8\xi} + v_{9\xi})$, the Frobenius of the Hessian is given by $\sqrt{4v_{7\xi}^2 + 4v_{8\xi}^2 + 4v_{9\xi}^2 + 2v_{4\xi}^2 + 2v_{5\xi}^2 + 2v_{6\xi}^2}$, and the

Table 2
Second-order geometric features based on the generalized eigenvalues of the quadric fit and the quadratic polynomial model.

Eigenvalue feature	Equation	Polynomial feature	Equation
Quadratic deviation	$\left(\sum_{k=2}^{\xi}\rho_k\right)^{-1}\rho_\xi$	Absolute algebraic distance	$\lvert T(\widetilde{\boldsymbol{x}}_{i*})^{\intercal}\boldsymbol{v}_{*\xi}\rvert$
Eigensum	$\sum_{k=2}^{\xi}\rho_k$	Squared algebraic distance	$(T(\widetilde{\boldsymbol{x}}_{i*})^{\intercal}\boldsymbol{v}_{*\xi})^2$
Eigenentropy	$-\sum_{k=2}^{\xi}\rho_k\ln(\rho_k)$	Coefficient's norm	$\lVert\boldsymbol{v}_{*\xi}\rVert$
Omnivariance	$\left(\prod_{k=2}^{\xi}\rho_k\right)^{1/(\xi-1)}$	Linear norm	$\lVert(v_{1\xi},v_{2\xi},v_{3\xi})\rVert$
Hypersphericity	$\rho_2^{-1}\rho_\xi$	Normalized linear norm	$\lVert\boldsymbol{v}_{*\xi}\rVert^{-1}\lVert(v_{1\xi},v_{2\xi},v_{3\xi})\rVert$
Hyperanisotropy	$\rho_2^{-1}(\rho_2-\rho_\xi)$	Cross norm	$\lVert(v_{4\xi},v_{5\xi},v_{6\xi})\rVert$
Squared spectral	ρ_2	Normalized cross norm	$\lVert\boldsymbol{v}_{*\xi}\rVert^{-1}\lVert(v_{4\xi},v_{5\xi},v_{6\xi})\rVert$
Frobenius	$\sqrt{\sum_{k=2}^{\xi}\rho_k}$	Square norm	$\lVert(v_{7\xi},v_{8\xi},v_{9\xi})\rVert$
Schatten	$\sum_{k=2}^{\xi}\sqrt{\rho_k}$	Normalized square norm	$\lVert\boldsymbol{v}_{*\xi}\rVert^{-1}\lVert(v_{7\xi},v_{8\xi},v_{9\xi})\rVert$
Normalized eigenentropy	$-\sum_{k=2}^{\xi}\widehat{\rho}_k\ln(\widehat{\rho}_k)$	Degree-2 norm	$\lVert(v_{4\xi},\ldots,v_{9\xi})\rVert$
		Normalized degree-2 norm	$\lVert\boldsymbol{v}_{*\xi}\rVert^{-1}\lVert(v_{4\xi},\ldots,v_{9\xi})\rVert$
		Absolute bias	$\lvert v_{10,\xi}\rvert$

Table 3
Second-order geometric features based on the derivatives of the quadratic polynomial model.

Differential feature	Equation	Differential feature	Equation
Laplacian	$\frac{\partial^2 g}{\partial x^2}+\frac{\partial^2 g}{\partial y^2}+\frac{\partial^2 g}{\partial z^2}$	Mean quadratic deviation	$m_i^{-1}\sum_{j=1}^{m_i}\left(\frac{\nabla g(\widetilde{\boldsymbol{x}}_{i*})^{\intercal}(\widetilde{\boldsymbol{x}}_{j*}-\widetilde{\boldsymbol{x}}_{i*})}{\lVert\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rVert}\right)^2$
Gradient norm	$\lVert\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rVert$	Min projected gradient (minpg)	$\min\{\lvert\boldsymbol{u}_{*k}^{\intercal}\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rvert\}_{k=1}^{3}$
Max projected gradient (maxpg)	$\max\{\lvert\boldsymbol{u}_{*k}^{\intercal}\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rvert\}_{k=1}^{3}$	Normalized minpg	$\min\left\{\frac{\lvert\boldsymbol{u}_{*k}^{\intercal}\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rvert}{\lVert\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rVert}\right\}_{k=1}^{3}$
Normalized maxpg	$\max\left\{\frac{\lvert\boldsymbol{u}_{*k}^{\intercal}\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rvert}{\lVert\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rVert}\right\}_{k=1}^{3}$	Saddleness	$-(\mathcal{H}^*)^{-1}\mathcal{H}_*$
Projected gradient norm (PGN)	$\lVert\boldsymbol{U}_{2D}^{\intercal}\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rVert$	Normalized PGN	$\left\lVert\boldsymbol{U}_{2D}^{\intercal}\frac{\nabla g(\widetilde{\boldsymbol{x}}_{i*})}{\lVert\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rVert}\right\rVert$
Gradient axis curvature (GAC)	$\lvert\nabla g(\widetilde{\boldsymbol{x}}_{i*})^{\intercal}\mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*})\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rvert$	Normalized GAC	$\frac{\lvert\nabla g(\widetilde{\boldsymbol{x}}_{i*})^{\intercal}\mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*})\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rvert}{\lVert\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rVert^2}$
Absolute quadraticity	$\left\lvert\det\left(D\left(\frac{\nabla g(\widetilde{\boldsymbol{x}}_{i*})}{\lVert\nabla g(\widetilde{\boldsymbol{x}}_{i*})\rVert}\right)\right)\right\rvert$	Hessian Frobenius	$\lVert\mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*})\rVert_{\mathrm{F}}$
Hessian absolute determinant	$\lvert\det(\mathbf{H}g(\widetilde{\boldsymbol{x}}_{i*}))\rvert$		

absolute determinant of the Hessian matrix is $\lvert 8v_{7\xi}v_{8\xi}v_{9\xi}+2(v_{4\xi}v_{5\xi}v_{6\xi}-v_{4\xi}^2v_{9\xi}-v_{5\xi}^2v_{8\xi}-v_{6\xi}^2v_{7\xi})\rvert$. The saddleness inside the interval $[-1,1]$ is computed from the minimum h_* and maximum h^* eigenvalues of the Hessian matrix, dividing the one with the lowest absolute value $\mathcal{H}_* = \arg\min_{h\in\{h_*,h^*\}}\lvert h\rvert$ by that with the highest one $\mathcal{H}^* = \arg\max_{h\in\{h_*,h^*\}}\lvert h\rvert$. Note that we take the eigenvalues, not their absolute values, and then multiply by (-1) to have saddle points in the positive side of the interval. Thus, saddleness will be positive if $\boldsymbol{x}_{i*}$ is a saddle point, with a maximum value of one corresponding to a symmetric saddle point, i.e., one with principal curvatures of equal magnitude. Negative values quantify how far $\boldsymbol{x}_{i*}$ is from being a saddle point. The rationale is that a saddle point must have a Hessian with some negative and some positive eigenvalues, which for our saddleness feature implies values greater than zero. If these are equal in magnitude, the final value will be one (the maximum value).

Recall the Gauss map explained in Section 2.3. When computed as the matrix of partial derivatives instead of a directional derivative, its determinant allows us to classify a point in a regular surface. When the determinant is positive, the point is said to be elliptic; if it is negative, the point is said to be hyperbolic [33, see pp. 178–186]. In our case, the sign ambiguity of fitting polynomial models as eigenvalue problems forces us to consider the absolute value of the determinant, and thus we have the absolute quadraticity. Many other differential features can be seen as projections of the gradient or the Hessian on alternative directions to the principal curvatures. When using $\boldsymbol{U}_{2D}\in\mathbb{R}^{3\times 2}$ to project the gradient, note that it refers to the basis matrix with the eigenvectors associated with the two maximum eigenvalues from the linear model.

Finally, the curvature features described in Table 4 involve the principal curvatures of the quadratic polynomial model. Due to the previously explained sign ambiguity of the eigenvalue-based fit of an algebraic model, some of these features, like the Gaussian, mean, and root mean squared curvatures, use absolute or squared values. This modeling decision allows us to deal with the impossibility of

Table 4
Second-order geometric features based on the principal curvatures of the quadratic polynomial model.

Curvature feature	Equation	Curvature feature	Equation
Gaussian curvature	$\lvert\kappa_*\kappa^*\rvert$	Mean curvature	$\frac{\lvert\kappa_*+\kappa^*\rvert}{2}$
Max absolute curvature	$\max\{\lvert\kappa_*\rvert,\lvert\kappa^*\rvert\}$	Min absolute curvature	$\min\{\lvert\kappa_*\rvert,\lvert\kappa^*\rvert\}$
Full umbilic deviation	$\kappa^*-\kappa_*$	Umbilicality	$\exp(\kappa_*-\kappa^*)$
Gaussian umbilicality	$\exp(-(\kappa^*-\kappa_*)^2)$	Shape index	$\frac{2}{\pi}\left\lvert\arctan\left(\frac{\kappa^*+\kappa_*}{\kappa_*-\kappa^*}\right)\right\rvert$
Root mean squared curvature	$\sqrt{\frac{(\kappa_*)^2+(\kappa^*)^2}{2}}$		

distinguishing when our surface is looking upwards or downwards in a universal way. Features measuring the umbilicity of a point are based on the definition of umbilic points, i.e., those whose principal curvatures are the same. In our case, the umbilic deviation will be zero for umbilic points and will be greater than zero the less umbilic a point is. For umbilicality, umbilic points will take a value of one, and non-umbilic points will go farther from one the less umbilic a point is. For Gaussian umbilicality, umbilic points have a value of one, and other points have a value closer to zero, the less umbilic they are.

3. Validation framework

In this section, we introduce the evaluation framework to assess the quality of the TW algorithm. First, we study the theoretical error and correlations with respect to analytical solutions and other approaches. Then, we perform experiments with scientific datasets for object part segmentation and large-scale semantic scene segmentation. Finally, we quantify the speedup of an open-source C++ implementation on the CESGA FinisTerrae-III supercomputer to analyze the algorithm's scalability.

3.1. Theoretical evaluation

For the theoretical evaluation of our algorithm we use Sagemath [34] to derive the mean and Gaussian curvatures from parametric surface models $\boldsymbol{\phi}(u,v)=(\hat{x}(u,v),\hat{y}(u,v),\hat{z}(u,v))$. We do this by computing the normal vector as shown in Eq. (9). Then we obtain the first and second fundamental forms $\boldsymbol{F}_1,\boldsymbol{F}_2\in\mathbb{R}^{2\times2}$ using the expressions in Eq. (10) and compute the principal curvatures κ_1,κ_2 as the eigenvalues of $\boldsymbol{F}_1^{-1}\boldsymbol{F}_2$. Finally, we derive the Gaussian curvature as $K=\kappa_1\kappa_2$ and the mean curvature as $H=2^{-1}(\kappa_1+\kappa_2)$.

$$\boldsymbol{n}=\left\|\frac{\partial\boldsymbol{\phi}}{\partial u}\times\frac{\partial\boldsymbol{\phi}}{\partial v}\right\|^{-1}\left(\frac{\partial\boldsymbol{\phi}}{\partial u}\times\frac{\partial\boldsymbol{\phi}}{\partial v}\right) \tag{9}$$

$$\boldsymbol{F}_1=\begin{bmatrix}\frac{\partial\boldsymbol{\phi}}{\partial u}^{\intercal}\frac{\partial\boldsymbol{\phi}}{\partial u} & \frac{\partial\boldsymbol{\phi}}{\partial u}^{\intercal}\frac{\partial\boldsymbol{\phi}}{\partial v}\\ \frac{\partial\boldsymbol{\phi}}{\partial u}^{\intercal}\frac{\partial\boldsymbol{\phi}}{\partial v} & \frac{\partial\boldsymbol{\phi}}{\partial v}^{\intercal}\frac{\partial\boldsymbol{\phi}}{\partial v}\end{bmatrix},\quad \boldsymbol{F}_2=\begin{bmatrix}\boldsymbol{n}^{\intercal}\frac{\partial^2\boldsymbol{\phi}}{\partial u^2} & \boldsymbol{n}^{\intercal}\frac{\partial^2\boldsymbol{\phi}}{\partial u\partial v}\\ \boldsymbol{n}^{\intercal}\frac{\partial^2\boldsymbol{\phi}}{\partial u\partial v} & \boldsymbol{n}^{\intercal}\frac{\partial^2\boldsymbol{\phi}}{\partial v^2}\end{bmatrix} \tag{10}$$

We compare the TW algorithm with PCA to quantify its improvement over a naive approach that does not compensate for the algebraic distance bias. We also compare it against SVD to see if it is a better alternative than a numerically more stable method for the naive strategy. Finally, we compare our method with the one implemented in the open-source scientific software Cloud Compare (CC) for the mean and Gaussian curvatures [35]. This method is based on the work of Zvi Har'el [36]. It assumes that, in an appropriate basis, the neighborhood of a point in the surface can be expressed in Monge's form $z=\hat{z}(x,y)$. Then it computes the curvatures from the partial derivatives of an osculating paraboloid.

We conduct comparisons considering the following varieties: sphere, ellipsoid, one-sheet hyperboloid, hyperbolic paraboloid, elliptical cylinder, toroid, and catenoid. We quantify the absolute errors at each point and aggregate them to calculate the **mean absolute error (MAE)** and **median absolute error (MeAE)**. Additionally, we compute the **Pearson** [37] **and Spearman** [38] **correlation coefficients** to assess the linear and monotonic correlations between the curvature estimates and the theoretical values. Note that we exclude constant values, like the curvatures of a sphere, from the correlations.

The errors and the correlations are also evaluated for different levels of Gaussian noise. More concretely, we consider a Gaussian random variable $d\sim\mathcal{N}(0,\sigma^2)$ to represent the distance of a translation along the direction given by the normalized gradient. Thus, each point $\boldsymbol{x}\in\mathbb{R}^3$ of the variety is translated to $\boldsymbol{x}'=\boldsymbol{x}+d\boldsymbol{n}$, where $\boldsymbol{n}\in\mathbb{R}^3$ is the normalized gradient vector as given in Eq. (9). We repeat the process considering different standard deviations σ ranging from 10^{-5} to 1.

3.2. Experimental evaluation

We evaluate point-wise classification tasks using Random Forest and computing the geometric features in multiscale spherical neighborhoods. More concretely, we evaluate object-part segmentation and large-scale semantic scene segmentation. For each experiment, we quantify the Overall Accuracy (OA), Precision (P), Recall (R), F1-score (F1) [39], and Intersection-over-Union (IoU) [40].

We also calculate their weighted versions to account for class imbalance. Additionally, we include Matthew's Correlation Coefficient (MCC) [41] and Cohen's Kappa score (κ) [42].

For the task of **object-part segmentation**, we use eight of the largest datasets from the ShapeNet Part benchmark [43]: 1) Airplane with wing, body, tail, and engine classes; 2) car with wheel, hood, and roof classes; 3) motorbike with wheel, handle, gas tank, light, and seat classes; 4) chair with leg, arm, back, and seat classes; 5) guitar with head, body, and neck classes; 6) lamp with canopy, lampshade, and base classes; 7) knife with handle and blade classes; and 8) pistol with trigger and guard, handle, and barrel classes.

For **large-scale semantic scene segmentation**, we consider the ALS dataset DALES [44]. This dataset covers $10\,\mathrm{km}^2$ in the city of Dayton (Ohio, USA) with approximately 505 million points and includes eight different classes: ground (impervious surfaces and grass), vegetation (trees, shrubs, hedges, and bushes), car, truck, power line, pole (power line poles, light poles, and transmission towers), fence (residential fences and highway barriers), and building (houses, high-rises and warehouses). We compute the evaluation metrics globally and also the F1 separately for each class, allowing us to assess which objects benefit more from second-order geometric descriptors.

3.3. Feature analysis

We conduct an ablation study on the features and a sensitivity analysis of their importance as a function of the numerical parameters of the TW algorithm. For the **ablation study**, we consider the chair, pistol, motorbike, and car point clouds of the ShapeNet Part benchmark. In these experiments, the second-order geometric features are sorted from most to least important. The object-part segmentation task is repeated using one to eleven features per neighborhood with four multi-scale spherical neighborhoods, i.e., 4 to 44 features in total.

For the **sensitivity analysis**, we follow a *ceteris paribus* strategy and explore two different cases. First, we change the eigenthreshold parameter ϵ governing the degeneracy check while keeping everything else constant. Then, we change the Tikhonov regularization parameter τ while keeping everything else constant (including the threshold for the degeneracy check). In doing so, the importance of second-order geometric features is aggregated across the categories introduced in Section 2.5. The large-scale ALS DALES dataset is used for these experiments. The features are computed in five multi-scale spherical neighborhoods.

3.4. Scalability analysis

Algorithms designed for 3D point clouds acquired by laser scanning or photogrammetry must be capable of handling massive datasets comprising millions of points. Therefore, they must be able to reduce their runtime by increasing the number of cores. Otherwise, they are not well-suited for real-world applications due to their lack of scalability. We conduct numerous experiments with varying numbers of cores on the HPC nodes of the FinisTerrae-III supercomputer [1]. These nodes have an Intel Xeon Ice Lake 8352Y CPU with 32 cores at 2.2 GHz, along with 256GB of RAM. Measuring the runtimes enables us to derive the **speedup** as $t_c^{-1}t_1$ where $t_1 \in \mathbb{R}_{>0}$ is the runtime of the sequential execution and $t_c \in \mathbb{R}_{>0}$ is the runtime when using c cores. We compute each experiment five times and take the mean value to have a robust representation of the runtime.

4. Results and discussion

In this section, we discuss the results from the theoretical validation, the experiments on point-wise classification datasets, and the scalability analysis.

4.1. Theoretical error and correlations

The theoretical error quantifications are presented in Table 5. These results clearly show that correcting the algebraic distance bias is more important than merely improving the numerical stability, as both PCA and SVD yield similar results and perform significantly worse than TW. Compared to the CC implementation, the TW algorithm produces similar or even better estimations for surface curvatures, depending on the case. The correlations in Table 6 confirm that, in our experiments, the surface curvature estimations of TW have systematically stronger linear and monotonic correlations than those of the other methods.

Fig. 2 represents the point-wise estimations of the mean curvature for the different methods considered in our experiments. It can be observed that PCA and SVD both produce unreliable estimations, while TW and CC are visually very similar. Nevertheless, in some cases, the TW algorithm offers some improvements over CC. For example, in the case of the hyperbolic paraboloid, the overall behavior of the mean curvature from TW is smoother than that from CC. The latter has a sharp transition as we move from the center to the extremes. This pattern is likely to happen because the parametric model of the hyperbolic paraboloid increases the separation between points at different rates depending on the direction as we move further from the center. Therefore, the estimated basis where the osculating paraboloid in Monge's form must be computed becomes problematic in these neighborhoods. For the toroid, the TW algorithm slightly overestimates the minimum mean curvature region near the center of the torus. However, it consistently outperforms the CC approach, which might be suffering from a basis-related bias again. This bias leads to overestimating the minimum curvatures when the points lie in subspaces passing through the torus center in directions nearly aligned with the canonical basis.

[1] FinisTerrae-III online documentation https://cesga-docs.gitlab.io/ft3-user-guide/overview.html

Table 5
Theoretical error quantification in terms of mean absolute error (MAE) and median absolute error (MeAE). The minimum error is highlighted in bold for each variety-curvature pair.

		CC		TW		PCA		SVD	
Variety	Curvature	MAE	MeAE	MAE	MeAE	MAE	MeAE	MAE	MeAE
Sphere	Gaussian	0.0783	0.0647	**0.0663**	**0.0523**	0.9099	0.9702	0.9099	0.9704
	Mean	0.0393	0.0320	**0.0328**	**0.0264**	0.7789	0.8272	0.7789	0.8279
Ellipsoid	Gaussian	0.0490	0.0271	**0.0299**	**0.0226**	0.7056	0.6179	0.7056	0.6180
	Mean	0.0292	0.0186	**0.0185**	**0.0145**	0.7362	0.6992	0.7362	0.6992
One-sheet hyperboloid	Gaussian	0.0504	0.0388	**0.0115**	**0.0077**	0.1540	0.1539	0.1540	0.1540
	Mean	0.0581	0.0271	**0.0072**	**0.0056**	0.2475	0.2128	0.2474	0.2127
Hyperbolic paraboloid	Gaussian	0.0311	0.0198	**0.0159**	**0.0135**	498.5807	0.3619	556.5291	0.3614
	Mean	0.0232	**0.0088**	**0.0117**	0.0100	14.8054	0.3952	15.1556	0.3941
Elliptical cylinder	Gaussian	0.0434	0.0355	**0.0020**	**0.0015**	0.0093	0.0045	0.0093	0.0045
	Mean	0.1132	0.0749	**0.0186**	**0.0159**	0.3229	0.3577	0.3229	0.3574
Toroid	Gaussian	0.2813	0.2246	**0.1975**	**0.1018**	46.5025	0.6366	47.2577	0.6371
	Mean	0.1741	0.1178	**0.0674**	**0.0635**	1.4137	0.5425	1.4203	0.5425
Catenoid	Gaussian	0.0260	**0.0173**	**0.0203**	**0.0173**	3245.8689	0.3067	5227.0772	0.3045
	Mean	0.0299	**0.0140**	**0.0228**	0.0176	24.2118	0.7353	26.6470	0.7322

Table 6
Theoretical correlations in terms of Pearson (r) and Spearman (ρ) coefficients. The highest correlation is highlighted in bold for each variety-curvature pair.

		CC		TW		PCA		SVD	
Variety	Curvature	r	ρ	r	ρ	r	ρ	r	ρ
Ellipsoid	Gaussian	0.9798	0.9833	**0.9929**	**0.9872**	0.6585	0.6298	0.6586	0.6300
	Mean	0.9795	0.9847	**0.9925**	**0.9878**	0.6999	0.6498	0.7001	0.6509
One-sheet hyperboloid	Gaussian	0.9867	0.9862	**0.9987**	**0.9963**	0.9749	0.8750	0.9749	0.8747
	Mean	0.9633	0.9792	**0.9995**	**0.9981**	0.8533	0.6031	0.8532	0.6029
Hyperbolic paraboloid	Gaussian	0.9994	0.9978	**0.9998**	**0.9994**	-0.0307	0.5064	-0.0215	0.5067
	Mean	0.9565	0.9606	**0.9917**	**0.9920**	0.0281	0.3363	0.0231	0.3373
Elliptical cylinder	Mean	0.7887	0.5706	**0.9912**	**0.9961**	0.9391	0.8826	0.9390	0.8827
Toroid	Gaussian	0.9493	0.9520	**0.9903**	**0.9969**	0.0683	0.8580	0.0681	0.8582
	Mean	0.8772	0.8554	**0.9925**	**0.9951**	-0.0900	0.3327	-0.0900	0.3329
Catenoid	Gaussian	0.9931	0.9770	**0.9981**	**0.9968**	-0.0022	0.5147	0.0116	0.5168

The errors and correlations for different levels of Gaussian noise are represented in Fig. 3. In general, the TW algorithm is robust to noise in the studied varieties up to standard deviations of about $\sigma = 10^{-2}$. At this level of noise, a few points may exhibit abnormal values, but, in general, most point-wise curvature estimations remain reliable in terms of error and correlation quantifications. Visually, the distribution of values remains consistent. The noise becomes problematic at some point between $\sigma = 10^{-2}$ and $\sigma = 10^{-1}$.

The state-of-the-art laser-scanning simulator HELIOS++ [45] handles noise by adding a Gaussian-distributed random variable to the simulated range measurement. This method is similar to our theoretical approach, which applies a Gaussian random variable along the gradient direction at each point in the discrete representation of the studied varieties. On the one hand, typical values for TLS simulations include $\sigma = 0.005$ (RIEGL VZ-400) or $\sigma = 0.008$ meters (RIEGL VZ-1000, RIEGL VQ-450). These scanners produce 3D point clouds that are close to our theoretical experiments, as their geometric information is fine-grained enough to enable analysis of small spherical neighborhoods with subunitary radii, which matches the radius $r = 0.4$ used in our theoretical quantifications. On the other hand, ALS simulations often use higher standard deviations, such as $\sigma = 0.01$ (Optech ALTM 2033), $\sigma = 0.02$ (LIVOX Mid-70), or $\sigma = 0.05$ meters (LEICA ALS50). In these cases, the radii of the spherical neighborhoods are often greater than one, as the geometric information is significantly less fine-grained than for terrestrial point clouds. Consequently, we argue that the TW algorithm is well-suited for practical applications with real noise. This claim is also supported by the empirical experiments conducted on real data.

Furthermore, we explore the relationship between the radius of the spherical neighborhood and the MeAE for realistic noise conditions with $\sigma = 0.01$ and $\sigma = 0.05$. The results of our analysis are represented in Fig. 4. In general, we observe that considering larger neighborhoods helps overcome noise-induced problems in the computation of second-order geometric descriptors. For the particular case of $\sigma = 0.05$, the geometric quantifications that are inconsistent when using a radius $r = 0.4$ (see Fig. 3) become more stable and much closer to what is expected when using $r = 1.0$. Numerically speaking, comparing $r = 0.55$ to $r = 0.4$ for $\sigma = 0.01$ reduces the MeAE to a 47.42 % of the original error for the hyperbolic paraboloid, increases to a 164.87 % for the toroid, reduces to a

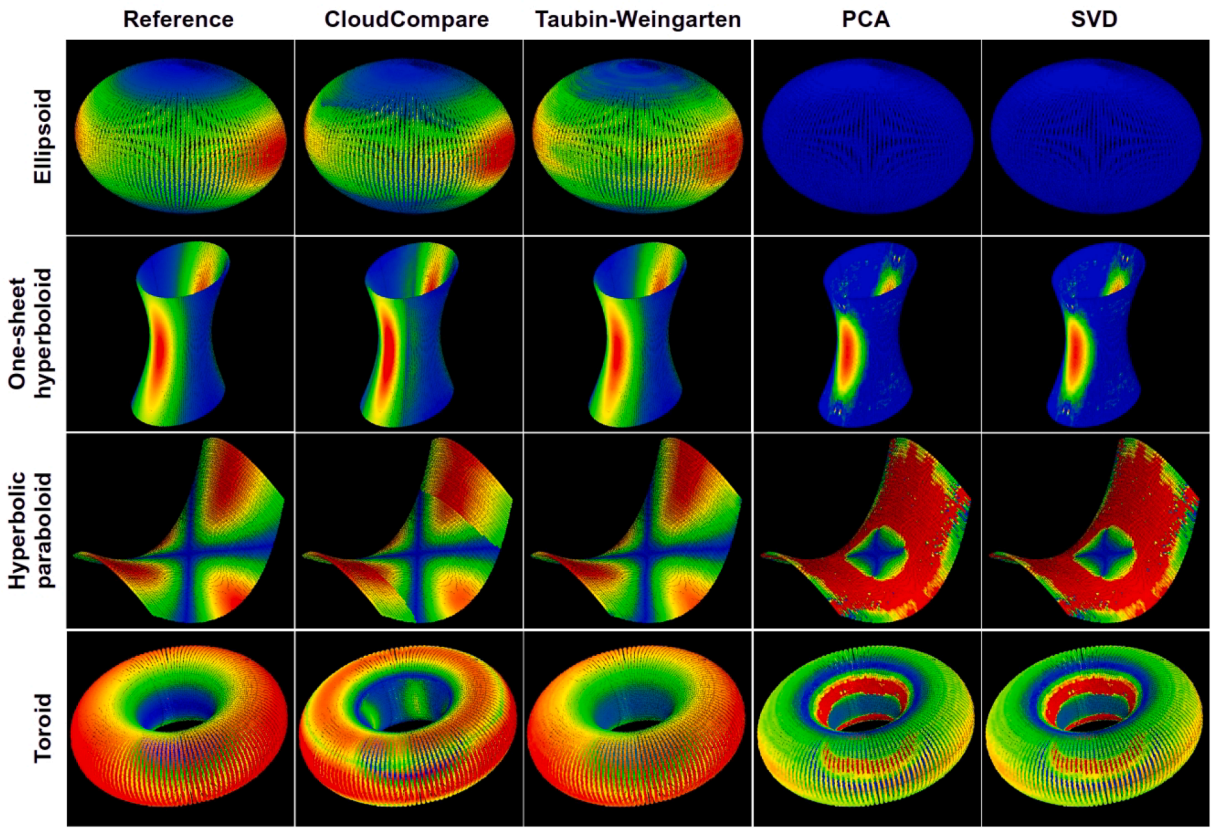

Fig. 2. Visualization of the mean curvature computed with SageMath (reference), CloudCompare, Taubin-Weingarten, PCA, and SVD. The color map uses blue for the lowest values and red for the highest ones with intermediate values in green. All the results are colored with the color map of the reference case.

42.70 % for the one-sheet hyperboloid, and 48.63 % for the catenoid. Considering $\sigma = 0.05$ and comparing $r = 1.0$ to $r = 0.4$, the MeAE is reduced to a 8.84 % for the hyperbolic paraboloid, 39.26 % for the toroid, 9.25 % for the one-sheet hyperboloid, and 16.28 % for the catenoid.

4.2. Point-wise classification

The point-wise classification experiments clearly demonstrate that using second-order geometric descriptors, especially when combined with first-order geometric descriptors, consistently yields better results than the typical approach based solely on first-order geometric features. This improvement holds for both object part segmentation and large-scale semantic scene segmentation tasks.

4.2.1. Object part segmentation

The quantitative results of object part segmentation on the ShapeNet Part benchmark are summarized in Table 7. The combination of first and second-order geometric descriptors leads to better results in terms of F1 and IoU for all objects. Moreover, it is also clear that using second-order geometric descriptors alone leads to better results than using only first-order geometric features. These results suggest that the descriptive power of second-order geometric descriptors both outperforms and complements that of first-order geometric features. The difference between the worst (first-order) and best (hybrid) models in terms of F1 ranges from 1.59 % and 2.57 % for the knife and guitar cases to 9.55 % and 9.99 % for the motorbike and pistol cases.

Fig. 5 allows for qualitative comparison of object part segmentation results in representative examples. It shows one case for each type of object, representing each part with a different color. In general, the hybrid model yields fewer misclassified points than the first-order model. This visualization, together with the quantifications in Table 7, suggests that incorporating second-order geometric features into machine learning models enhances the point-wise classification of objects. This result is important because ground and buildings are often adequately addressed with current models based on first-order geometric features. However, comparatively small objects are still problematic. In other words, second-order geometric descriptors tackle the problem of small and scarcely represented objects (compared to the most frequent classes), which is especially relevant for large-scale semantic scene segmentation.

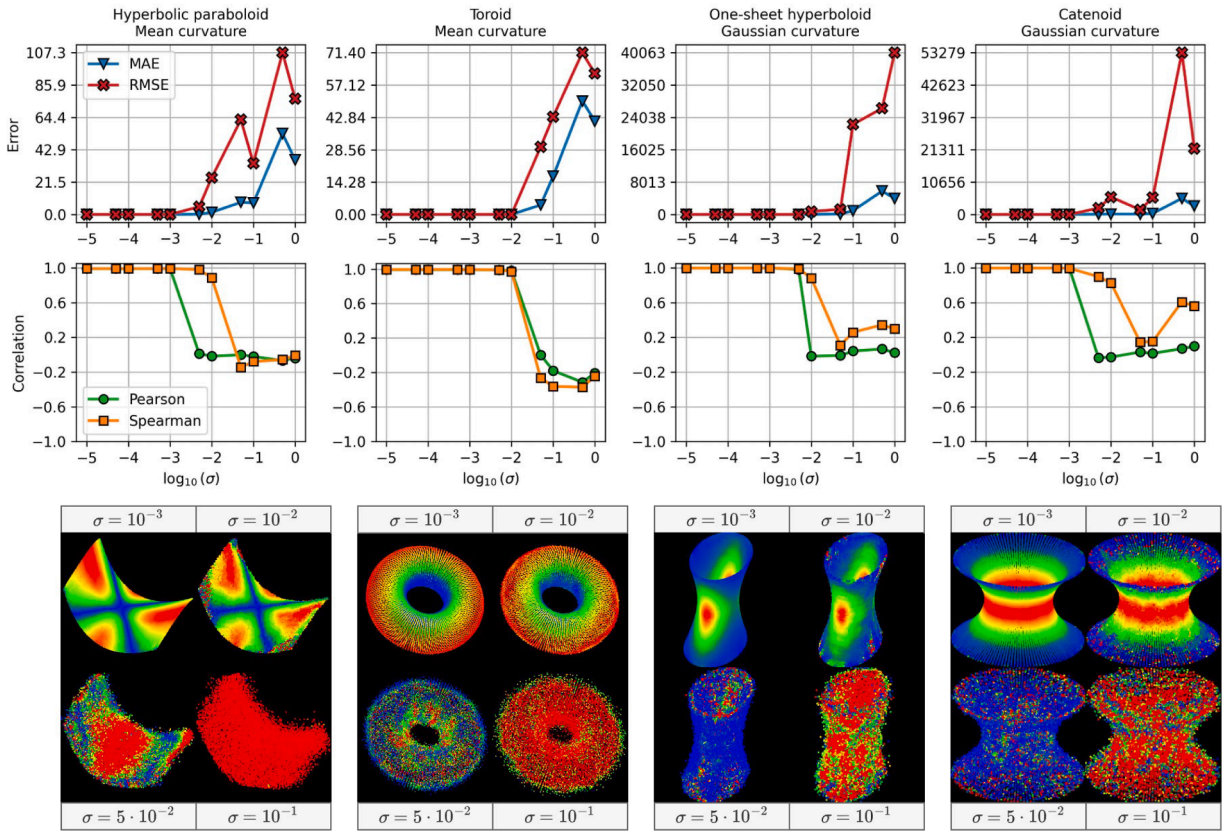

Fig. 3. Visualization of the mean and Gaussian curvatures computed with the Taubin-Weingarten algorithm under different normal noise conditions. The top row represents the mean absolute error (MAE) and the root mean squared error (RMSE). The mid row represents the Pearson and Spearman correlation coefficients. The bottom row shows the point-wise curvatures for different values of sigma, from blue (low) to red (high). The radius of the point-wise spherical neighborhoods is $r = 0.4$ for all cases.

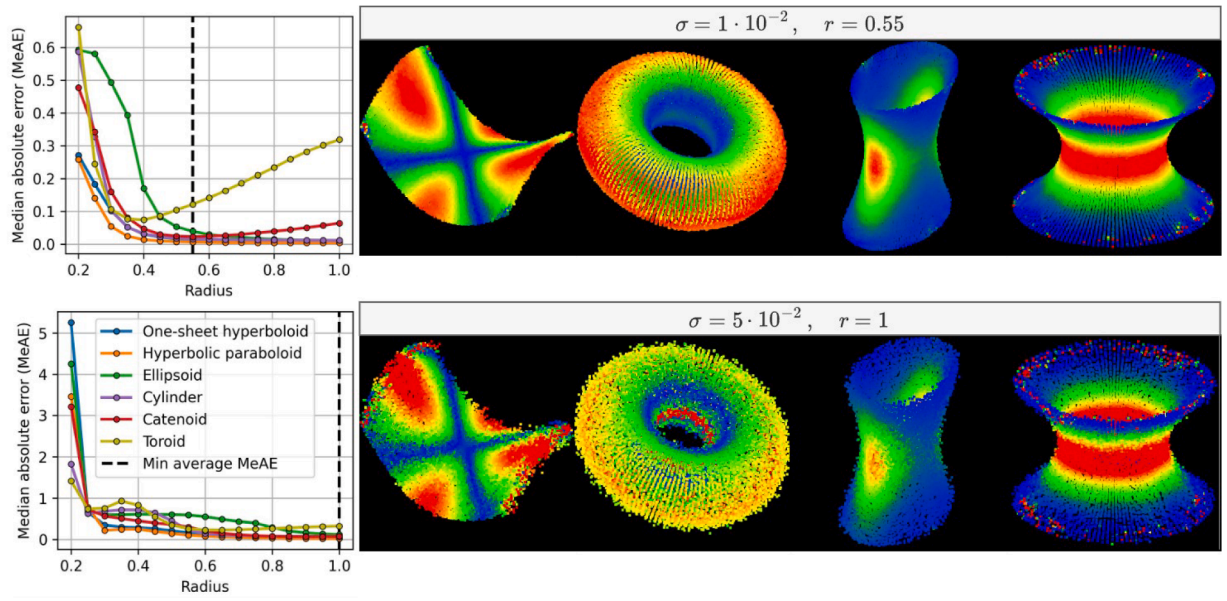

Fig. 4. Visualization of the median absolute error (MeAE) as a function of the spherical neighborhood radius under normal noise conditions with $\sigma \in \{10^{-2}, 5 \cdot 10^{-2}\}$. The vertical dashed line represents the radius corresponding to the minimum average error considering the six studied varieties. The geometries on the right side represent the mean and Gaussian curvatures computed by the Taubin-Weingarten algorithm using the radius that minimizes the average error.

Table 7
Comparison between first and second-order geometric descriptors for object part segmentation with Random Forest on the ShapeNet Part benchmark. The highest score for each dataset is highlighted in bold.

		Scores (%)										
Dataset	Model	OA	P	R	F1	IoU	wP	w*R*	wF1	wIoU	MCC	κ
Airplane	RF 1st order	86.16	84.45	82.50	83.42	71.80	86.10	86.16	86.10	75.78	78.53	78.52
	RF 2nd order	87.85	86.71	84.22	85.39	74.72	87.80	87.85	87.79	78.39	81.12	81.09
	RF hybrid	**89.19**	**88.36**	**86.21**	**87.23**	**77.50**	**89.16**	**89.19**	**89.15**	**80.53**	**83.22**	**83.21**
Car	RF 1st order	93.17	92.90	91.26	92.04	85.59	93.13	93.17	93.12	87.36	87.71	87.67
	RF 2nd order	**96.11**	95.77	**94.88**	95.31	91.12	**96.10**	**96.11**	**96.09**	**92.54**	**93.02**	**93.01**
	RF hybrid	96.08	**96.02**	94.83	**95.41**	**91.32**	96.07	96.08	96.06	92.49	92.96	92.94
Motorbike	RF 1st order	81.11	71.90	55.00	59.60	47.04	79.27	81.11	79.08	68.54	56.85	55.41
	RF 2nd order	84.18	81.43	58.40	63.45	50.75	83.26	84.18	81.99	72.67	64.44	62.96
	RF hybrid	**85.93**	**82.89**	**63.68**	**69.15**	**56.15**	**85.16**	**85.93**	**84.49**	**75.51**	**68.81**	**67.74**
Chair	RF 1st order	74.71	69.96	67.30	68.19	53.10	74.59	74.71	74.54	59.67	61.79	61.75
	RF 2nd order	82.31	78.53	70.92	73.09	59.89	82.23	82.31	82.00	70.05	73.13	73.07
	RF hybrid	**82.63**	**79.14**	**74.70**	**76.38**	**63.11**	**82.61**	**82.63**	**82.48**	**70.45**	**73.68**	**73.60**
Guitar	RF 1st order	94.26	90.92	89.05	89.87	82.07	94.18	94.26	94.15	89.37	86.76	86.64
	RF 2nd order	94.13	91.69	89.25	90.32	82.70	94.07	94.13	94.00	89.03	86.44	86.27
	RF hybrid	**95.50**	**93.44**	**91.60**	**92.44**	**86.19**	**95.47**	**95.50**	**95.44**	**91.50**	**89.65**	**89.57**
Lamp	RF 1st order	88.19	73.33	59.32	63.19	51.70	87.38	88.19	87.54	79.74	59.52	59.11
	RF 2nd order	89.61	**81.18**	59.35	64.17	53.13	88.93	89.61	88.83	81.64	63.77	63.04
	RF hybrid	**90.37**	79.27	**62.14**	**66.84**	**55.62**	**89.76**	**90.37**	**89.79**	**82.94**	**67.01**	**66.54**
Knife	RF 1st order	82.90	83.27	83.01	82.87	70.78	83.37	82.90	82.87	70.78	66.28	65.86
	RF 2nd order	84.00	84.32	84.05	83.97	72.39	84.37	84.00	83.97	72.39	68.37	68.02
	RF hybrid	**84.52**	**84.91**	**84.55**	**84.46**	**73.25**	**84.96**	**84.52**	**84.47**	**73.26**	**69.46**	**69.03**
Pistol	RF 1st order	77.46	73.69	62.00	65.92	50.93	76.86	77.46	76.69	63.50	49.88	49.43
	RF 2nd order	82.11	81.39	68.15	72.92	58.68	81.92	82.11	81.28	69.48	60.09	59.03
	RF hybrid	**83.13**	**82.90**	**71.53**	**75.91**	**62.10**	**82.95**	**83.13**	**82.53**	**71.06**	**62.70**	**61.94**

Table 8
Comparison between first and second-order geometric descriptors for semantic scene segmentation with Random Forest on the large-scale DALES dataset. The highest score is highlighted in bold.

	Scores (%)										
Model	OA	P	R	F1	IoU	wP	w*R*	wF1	wIoU	MCC	κ
RF 1st order	94.35	64.10	78.13	68.07	58.23	95.18	94.35	94.64	90.25	90.89	90.85
RF 2nd order	95.41	67.57	78.49	70.40	60.81	96.01	95.41	95.61	91.94	92.59	92.56
RF hybrid	**95.66**	**68.66**	**80.93**	**72.35**	**62.77**	**96.19**	**95.66**	**95.84**	**92.33**	**92.98**	**92.96**

Table 9
Class-wise scores of first and second-order geometric descriptors for semantic scene segmentation with Random Forest on the large-scale DALES dataset. The highest score for each class is highlighted in bold.

	F1-score per class (%)							
Model	ground	veget.	car	truck	power	fence	pole	building
RF 1st order	97.16	92.09	54.76	19.17	83.22	50.49	56.39	93.59
RF 2nd order	97.71	93.74	62.12	19.55	84.57	50.95	61.85	94.82
RF hybrid	**97.87**	**93.77**	**64.54**	**24.97**	**85.75**	**53.74**	**64.89**	**95.41**

4.2.2. Large-scale semantic scene segmentation

The quantitative results from the global evaluation of the models in the DALES dataset for large-scale semantic scene segmentation are summarized in Table 8. The model with second-order geometric features alone improves the F1 and IoU in 2.33 % and 2.58 % respectively, compared to the model with first-order geometric descriptors. The hybrid model outperforms the first-order model by 4.28 % in F1 and 4.54 % in IoU. Analyzing the class-wise scores in Table 9 clarifies that the most significant improvements come from small objects like cars (9.78 % better F1), trucks (5.80 % better F1), and poles (8.50 % better F1). These results agree with the expectations from the object part segmentation results in Section 4.2.1. Furthermore, the results of overrepresented classes also improve. More concretely, in terms of F1, the ground class improves 0.71 % when using the hybrid model. The vegetation class improves 1.68 % and the building class 1.82 %. A qualitative visual inspection of the results is available in Fig. 6.

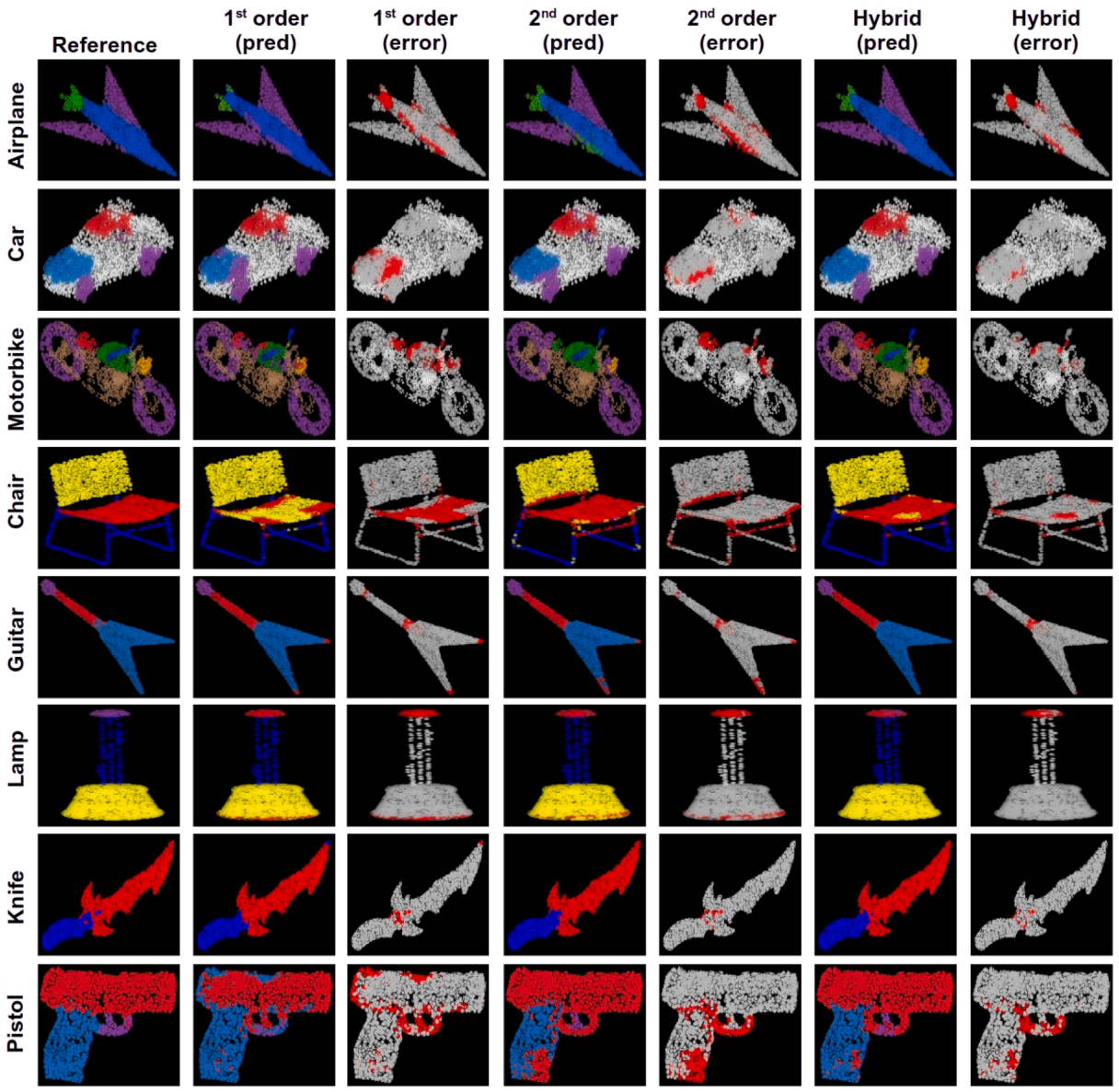

Fig. 5. Visualization of the results of first, second-order, and hybrid models for object part segmentation on some examples of the ShapeNet Part benchmark. In the predictions, each part is represented with a different color. The error mask represents misclassified points in red color and correctly classified ones in light gray.

4.3. Feature analysis

The features for each case in the ablation study are ordered by the feature importances derived from the Random Forest ensemble, without permutations. For the chair dataset, the five most important features are the gradient norm, the normalized GAC, the mean quadratic deviation, the Gaussian curvature, and the saddleness. For the pistol dataset, they are the mean quadratic deviation, the saddleness, the Laplacian, the shape index, and the gradient norm. For the motorbike dataset, they are the mean quadratic deviation, the saddleness, the Laplacian, the normalized GAC, and the gradient norm. For the car dataset, they are the normalized GAC, the gradient norm, the absolute algebraic distance, the shape index, and the Laplacian.

The impact of the number of features on the OA, F1, and MCC is depicted in Fig. 7. The best results are achieved with 8 to 11 features per neighborhood. As four multi-scale spherical neighborhoods are used, the total number of features lies between 32 and 44. Both the importance of each feature and the model's marginal improvement from adding new features depend strongly on the input geometries.

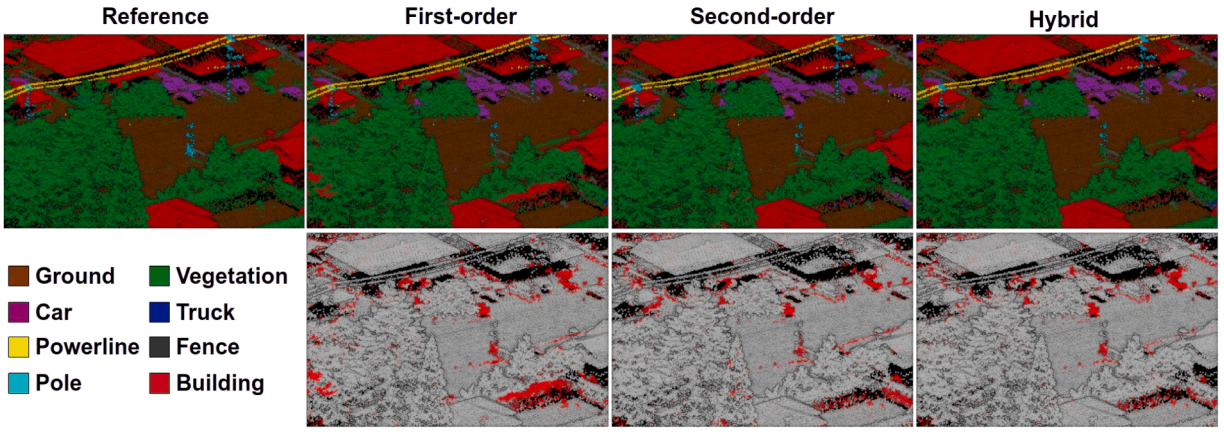

Fig. 6. Visualization of the results of first, second-order, and hybrid models for large-scale semantic segmentation on the DALES dataset. The top row shows the predictions with each part represented using a different color. Note that there are no trucks in this example. The error mask in the bottom row represents misclassified points in red color and correctly classified ones in light gray.

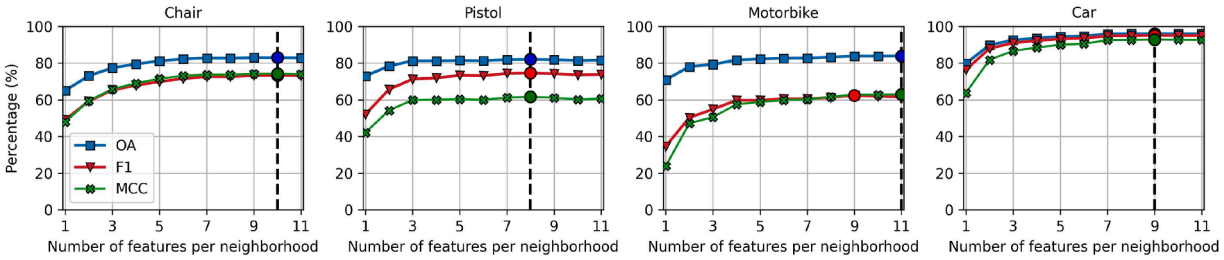

Fig. 7. Results of the ablation study on second-order features only using the ShapeNet Part benchmark. The marker for each score is replaced by a circle at the maximum value. The black, dashed vertical line represents the minimum number of features required to achieve the best result.

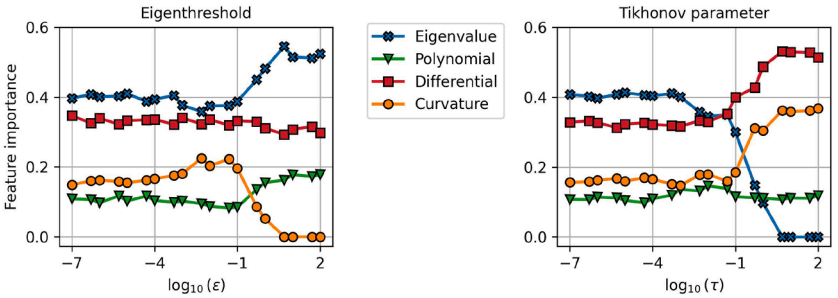

Fig. 8. Results of the sensitivity analysis for the eigenthreshold and Tikhonov parameter on the DALES dataset. Feature importance is aggregated for each family of features according to the categories introduced in Section 2.5.

The impact of the numerical parameters governing the algorithm and the computed features is shown in Fig. 8. The sensitivity analysis of the eigenthreshold for the degeneracy check shows that once it becomes too high ($\epsilon > 10^{-1}$), the curvature descriptors become less valuable and eventually reach zero feature importance. The main reason is that a high eigenthreshold implies that many local surfaces will be interpreted as linear varieties (typically planes embedded in 3D space, but also lines), even when they are not. In this case, there is no quadratic model, so any curvature information is null. The eigenvalue and polynomial features increase their importance consequently, but the model experiences a decrease of around 2.5 % in terms of F1 when curvature features vanish.

Concerning the Tikhonov regularization parameter, the eigenvalue features are the ones that gradually vanish once it becomes too high ($\tau > 10^{-1}$). The reason is that the Tikhonov regularization parameter lifts the eigenvalues, and if it is too big, it becomes impossible to distinguish reliable eigenvalues from numerical artefacts. In this scenario, differential and curvature features that rely on the eigenvectors can still be used, and their feature importance increases to compensate for the loss of eigenvalue-based geometric descriptors. In this case, the model experiences a decrease of around 9.5 % in terms of F1.

The results of the sensitivity analysis suggest that both the Eigenthreshold and the Tikhonov regularization parameter should be set to be inside $(0, 10^{-1}]$. The most common decimal encoding formats are 32-bit and 64-bit floating-point. Following the IEEE standard

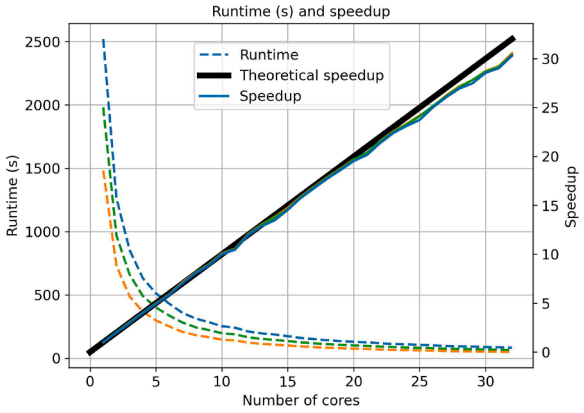

Fig. 9. Results of the scalability analysis from one to 32 cores. Dashed lines show the runtime in seconds, solid lines show the speedup.

for floating-point arithmetic [46], 32-bit precision would correspond to the interval $[10^{-7}, 10^{-1}]$ and 64-bit precision to $[10^{-16}, 10^{-1}]$. The empirical results shown in Fig. 7 support the recommendation to choose τ close to the machine epsilon (10^{-7} in our experiments with 32 bits) given in Section 2.2. In general, thresholds closer to the lowest value of the interval should be preferred over those closer to the highest value.

4.4. Scalability measurements

The results of the scalability analysis are summarized in Fig. 9. The runtime (in seconds) and the speedup are represented for three different large-scale point clouds from the DALES dataset. The point clouds are *5080_54435* with $11,345,691$ points, *5185_54485* with $14,539,130$ points, and *5110_54320* with $17,747,769$ points. When using the maximum number of cores (32) in the Intel Xeon Ice Lake 8352Y CPU, they achieve speedups 30.55, 30.41, and 30.29, respectively. Compared to the ideal speedup of 32, this is an outstanding result that corroborates the embarrassingly parallel nature of the TW algorithm.

5. Conclusions

In this paper, we introduce the Taubin-Weingarten algorithm, a novel method to compute second-order geometric descriptors in 3D point clouds. These new geometric features enhance the performance of machine learning models on object part segmentation and semantic scene segmentation by $1.59 - 9.99\,\%$ F1, depending on the class. The most significant improvements occur in the critical case of small, underrepresented objects, such as cars or poles, which are less frequent than ground or vegetation. These small objects are challenging to handle when using classical machine learning models for point-wise classification, such as Random Forest with first-order geometric features, and they are especially important in practical applications.

On top of that, we quantify the theoretical error of the Gaussian and mean curvatures in mathematical varieties with respect to their analytical solutions derived from the parametric form. We find that the Taubin-Weingarten algorithm provides more accurate estimations of surface curvature than the simple PCA and SVD approaches. Moreover, it can yield even better results than some standard methods, as the algorithm to estimate the Gaussian and mean curvatures from a quadric fit in Monge's form, which is implemented in the open-source scientific software Cloud Compare.

The theoretical noise analysis indicates that the proposed algorithm is robust enough to operate under the expected noise conditions of 3D point clouds. The ablation study conducted across different geometries shows that selecting 8 to 11 second-order geometric descriptors per neighborhood is sufficient to achieve successful generalization with machine learning models. Studying the sensitivity of the two numerical parameters of the proposed algorithm reveals that selecting values between 10^{-7} and 10^{-4} does not affect the quality of the geometric information, whereas significant problems arise for values above 10^{-1}. The scalability analysis shows that our method achieves good speedups as the number of cores increases (systematically above 30 for 32 cores). Consequently, the Taubin-Weingarten algorithm is well-suited to deal with dense 3D point clouds from real data acquisition campaigns, which often lead to large datasets in terms of surface area and point resolution.

There are two main limitations of our method. First, it is limited to offline 3D point cloud processing due to its high computational burden. Although the method achieves a significant speedup and can be computed within a reasonable time on modern hardware, many real-time systems lack the high-performance cores available in supercomputers and high-end PCs. Consequently, the method is not yet suitable for real-time 3D point cloud processing using embedded hardware. Second, the extreme numerical values that can arise in geometrically intricate neighborhoods pose challenges for machine learning models that perform numerical computations on input features. In our experiments, we used a random forest model that avoids arithmetic operations on the feature vectors and thus is robust to extreme numerical values. However, in its current form, the TW algorithm is likely problematic when used with numerically intensive models such as neural networks.

Future work could extend the method to address its current limitations, for example, by switching from a point-wise to a leaf-wise paradigm. Moving the computation to one neighborhood per leaf of an advanced data structure, such as Octrees [47] or KDTrees [48], might lead to a significant performance improvement. Moreover, the proposed method could also be used as a feature extraction procedure within 3D geometric neural networks. These models are often based on hierarchical feature extraction architectures [49–53] that differ in the feature extraction operator (e.g., MLPs, point convolutions, transformers). We hypothesize that the Taubin-Weingarten algorithm could be used as a feature extractor for 3D geometric neural networks. In doing so, numerical stability must be carefully addressed, as a single NaN or a few extreme values could be enough to invalidate the computation of the loss function and, consequently, the gradient-based update of the weights. Custom regularization strategies will likely need to be developed as well.

Open-access software and data availability

All the scripts needed to reproduce the results are provided as open-source software at https://github.com/albertoesmp/twalg_scripts. The C++ implementation of the algorithm is available in the open-source scientific software VirtuaLearn3D++ at https://doi.org/10.5281/zenodo.17911279. All the datasets used in this study are open-access and can be downloaded from their sources. We also provide two Jupyter notebooks for Sagemath that can be used to reproduce the reference values for the theoretical error quantifications and the robustness to noise experiments.

Data availability

The datasets are already open access. The code is available in a public repository as open-source scientific software.

Acknowledgments

This work has received financial support from the Agencia Estatal de Investigación (Spain) grants grant PID2020-115155GB-I00, PID2022-141623NB-I00 and PID2024-155502NB-I00 (financial support from MCIN/AEI/10.13039/501100011033 and from FEDER, UE) and the Xunta de Galicia - Conselleria de Educación, Ciencia, Universidades e Formación Profesional (CiTIUS-Research Center in Intelligent Technologies a Research Center of the Galician University System accreditation 2024–2027 ED431G-2023/04, and Reference Competitive Group accreditations ED431C-2022/016 and ED431C 2023/31) and the European Union (European Regional Development Fund - ERDF).

Moreover, this work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in the frame of the project "VirtuaLearn3D" (project number: 496418931), and the project “Fostering a community-driven and sustainable HELIOS++ scientific software” (project number: 528521476, programme: “Research Software – Quality Assured and Re-usable”).

We also acknowledge the Galician Supercomputing Center (CESGA) for providing us with access to the FinisTerrae-III supercomputer for accurate speedup measurements and an exhaustive sensitivity analysis on a large-scale dataset.

References

[1] C. Şahin, Applications of Point Cloud Technology, IntechOpen, Rijeka, Rijeka, 2024. https://doi.org/10.5772/intechopen.1001660
[2] S. Oehmcke, L. Li, K. Trepekli, J.C. Revenga, T. Nord-Larsen, F. Gieseke, C. Igel, Deep point cloud regression for above-ground forest biomass estimation from airborne liDAR, Remote Sens. Environ. 302 (2024) 113968. https://doi.org/10.1016/j.rse.2023.113968
[3] B. Xiang, M. Wielgosz, T. Kontogianni, T. Peters, S. Puliti, R. Astrup, K. Schindler, Automated forest inventory: analysis of high-density airborne liDAR point clouds with 3D deep learning, Remote Sens. Environ. 305 (2024) 114078. https://doi.org/10.1016/j.rse.2024.114078
[4] H. Sun, Q. Ye, Q. Chen, L. Fu, Z. Xu, C. Hu, Tree canopy volume extraction fusing ALS and TLS based on improved pointnext, Remote Sens. 16, 2641 (14) (2024). https://doi.org/10.3390/rs16142641
[5] S. Jiang, P. Qi, L. Han, L. Liu, Y. Li, Z. Huang, Y. Liu, X. He, Navigation system for orchard spraying robot based on 3D LiDAR SLAM with NDT_ICP point cloud registration, Comput. Electron. Agric. 220 (2024) 108870. https://doi.org/10.1016/j.compag.2024.108870
[6] A. Crisan, M. Pepe, D. Costantino, S. Herban, From 3D point cloud to an intelligent model set for cultural heritage conservation, Heritage 7 (3) (2024) 1419–1437. https://doi.org/10.3390/heritage7030068
[7] M. Doneus, B. Höfle, D. Kempf, G. Daskalakis, M. Shinoto, Human-in-the-loop development of spatially adaptive ground point filtering pipelines-an archaeological case study, Archaeol. Prospect. 29 (4) (2022) 503–524. https://doi.org/10.1002/arp.1873
[8] X. Li, L. Wang, H. Guan, K. Chen, Y. Zang, Y. Yu, Urban tree species classification using UAV-based multispectral images and LiDAR point clouds, J. Geovisualiz. Spatial Anal. 8 (1) (2024) 5. https://doi.org/10.1007/s41651-023-00167-9
[9] A.M. Esmorís, D.L. Vilariño, D.F. Arango, F.-A. Varela-García, J.C. Cabaleiro, F.F. Rivera, Characterizing zebra crossing zones using LiDAR data, Comput.-Aided Civ. Infrastruct. Eng. 38 (13) (2023) 1767–1788. https://doi.org/10.1111/mice.12968
[10] Y. Gao, H. Li, W. Fu, C. Chai, T. Su, Damage volumetric assessment and digital twin synchronization based on liDAR point clouds, Autom. Constr. 157 (2024) 105168. https://doi.org/10.1016/j.autcon.2023.105168
[11] N. Wang, D. Ma, X. Du, B. Li, D. Di, G. Pang, Y. Duan, An automatic defect classification and segmentation method on three-dimensional point clouds for sewer pipes, Tunnelling Underground Space Technol. 143 (2024) 105480. https://doi.org/10.1016/j.tust.2023.105480
[12] C. Yang, Z. Li, P. Xu, H. Huang, Recognition and optimisation method of impact deformation patterns based on point cloud and deep clustering: applied to thin-walled tubes, J. Ind. Inf. Integ. 40 (2024) 100607. https://doi.org/10.1016/j.jii.2024.100607
[13] C. Zhang, J. Shu, H. Zhang, Y. Ning, Y. Yu, Estimation of load-carrying capacity of cracked RC beams using 3D digital twin model integrated with point clouds and images, Eng. Struct. 310 (2024) 118126. https://doi.org/10.1016/j.engstruct.2024.118126
[14] P. Dąbek, J. Wodecki, P. Kujawa, A. Wróblewski, A. Macek, R. Zimroz, 3D Point cloud regularization method for uniform mesh generation of mining excavations, ISPRS J. Photogramm. Remote Sens. 218 (2024) 324–343. https://www.sciencedirect.com/science/article/pii/S0924271624004015. https://doi.org/10.1016/j.isprsjprs.2024.10.024
[15] Y. Li, Z. Xiao, J. Li, T. Shen, Integrating vision and laser point cloud data for shield tunnel digital twin modeling, Autom. Constr. 157 (2024) 105180. https://doi.org/10.1016/j.autcon.2023.105180
[16] J. Shan, C.K. Toth, Topographic Laser Ranging and Scanning: Principles and Processing, CRC Press, 2nd edition, 2018. https://doi.org/10.1201/9781315154381

[17] M. Weinmann, S. Urban, S. Hinz, B. Jutzi, C. Mallet, Distinctive 2D and 3D features for automated large-scale scene analysis in urban areas, Comput. Graph. 49 (2015) 47–57. https://doi.org/10.1016/j.cag.2015.01.006

[18] T. Hackel, J.D. Wegner, K. Schindler, Fast semantic segmentation of 3d point clouds with strongly varying density, ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci. III-3 (2016) 177–184. https://doi.org/10.5194/isprs-annals-III-3-177-2016

[19] H. Thomas, F. Goulette, J.-E. Deschaud, B. Marcotegui, Y. LeGall, Semantic classification of 3D point clouds with multiscale spherical neighborhoods, in: 2018 International Conference on 3D Vision (3DV), 2018, pp. 390–398. https://doi.org/10.1109/3DV.2018.00052

[20] V. Zahs, K. Anders, J. Kohns, A. Stark, B. Höfle, Classification of structural building damage grades from multi-temporal photogrammetric point clouds using a machine learning model trained on virtual laser scanning data, Int. J. Appl. Earth Obs. Geoinf. 122 (2023) 103406. https://doi.org/10.1016/j.jag.2023.103406

[21] A.M. Esmorís, H. Weiser, L. Winiwarter, J.C. Cabaleiro, B. Höfle, Deep learning with simulated laser scanning data for 3D point cloud classification, ISPRS J. Photogramm. Remote Sens. 215 (2024) 192–213. https://doi.org/10.1016/j.isprsjprs.2024.06.018

[22] W. Albert, H. Weiser, R. Tabernig, B. Höfle, Wind during terrestrial laser scanning of trees: simulation-based assessment of effects on point cloud features and leaf-wood classification, ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci. X-G-2025 (2025) 25–32. https://doi.org/10.5194/isprs-annals-X-G-2025-25-2025

[23] G. Taubin, Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 13 (11) (1991) 1115–1138. https://doi.org/10.1109/34.103273

[24] N. Lehmann, U. Reif, Notes on the curvature tensor, Graph. Models 74 (6) (2012) 321–325. Special Issue of selected papers from the 8th Dagstuhl seminar on Geometric Modeling. https://doi.org/10.1016/j.gmod.2012.04.003

[25] X.-D. Zhang, Matrix Analysis and Applications, Cambridge University Press, 2017. https://doi.org/10.1017/9781108277587

[26] G. Golub, W. Kahan, Calculating the singular values and pseudo-inverse of a matrix, J. Soc. Ind. Appl. Math. Series B Numer. Anal. 2 (1964) 205–224. https://epubs.siam.org/doi/10.1137/0702016.

[27] R.L. Burden, D.J. Faires, A.M. Burden, Numerical Analysis, Cengage, 10 edition, 2016.

[28] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.

[29] D.W. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, J. Soc. Ind. Appl. Math. 11 (2) (1963) 431–441. http://www.jstor.org/stable/2098941.

[30] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes. The Art of Scientific Computing, Cambridge University Press, 3rd (version 7) edition, 2023.

[31] C.B. Moler, G.W. Stewart, An algorithm for generalized matrix eigenvalue problems, SIAM J. Numer. Anal. 10 (2) (1973) 241–256. https://doi.org/10.1137/0710024

[32] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, Inc., 2nd edition, 2019.

[33] T.F. Banchoff, S. Lovett, Differential Geometry of Curves and Surfaces, Differential geometry of curves and surfaces, CRC Press, Boca ratón, FL, 3rd edition, Boca ratón, FL, 2023.

[34] The Sage Developers, SageMath, the Sage Mathematics Software System (Version 10.5), 2025. https://www.sagemath.org.

[35] D. Girardeau-Montaut, CloudCompare (Version 2.13.2) [GPL software], 2024, (http://www.cloudcompare.org). Open-source 3D point cloud processing software.

[36] Z. Har'el, Curvature of curves and surfaces – a parabolic approach, 1995, Accessed: 2025-08-27, http://harel.org.il/zvi/docs/parabola.pdf.

[37] K. Pearson, O.M. F.E. Henrici, VII. Mathematical contributions to the theory of evolution. – III. regression, heredity, and panmixia, Philosoph. Trans. R. Soc. London. Series A, Containing Papers Math. Phys. Character 187 (1896) 253–318. https://doi.org/10.1098/rsta.1896.0007

[38] C. Spearman, The proof and measurement of association between two things, Am. J. Psychol. 15 (1) (1904) 72–101. http://www.jstor.org/stable/1412159.

[39] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Inf. Process. Manag. 45 (4) (2009) 427–437. https://doi.org/10.1016/j.ipm.2009.03.002

[40] P. Jaccard, Étude comparative de la distribution florale dans une portion des Alpes et du Jura, Bulletin de la Société Vaudoise des Sciences Naturelles 37 (1901) 547–579.

[41] B.W. Matthews, et al., Comparison of the predicted and observed secondary structure of T4 phage lysozyme, Biochimica et Biophysica Acta (BBA) - Protein Structure 405 (2) (1975) 442–451. https://doi.org/10.1016/0005-2795(75)90109-9

[42] J. Cohen, A coefficient of agreement for nominal scales, Educ. Psychol. Meas. 20 (1) (1960) 37–46. https://doi.org/10.1177/001316446002000104

[43] A.X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, F. Yu, ShapeNet: An Information-Rich 3D Model Repository, 2015, arXiv:1512.03012

[44] N.M. Singer, V.K. Asari, DALES Objects: a large scale benchmark dataset for instance segmentation in aerial lidar, IEEE Access 9 (2021) 97495–97504. https://doi.org/10.1109/ACCESS.2021.3094127

[45] L. Winiwarter, A.M. Esmorís Pena, H. Weiser, K. Anders, J. Martínez Sánchez, M. Searle, B. Höfle, Virtual laser scanning with HELIOS++: a novel take on ray tracing-based simulation of topographic full-waveform 3D laser scanning, Remote Sens. Environ. 269, 112772 (2022). https://doi.org/10.1016/j.rse.2021.112772

[46] IEEE Computer Society, IEEE Standard for floating-Point arithmetic, IEEE Std 754–2008 (2008) 1–70. https://doi.org/10.1109/IEEESTD.2008.4610935

[47] D. Meagher, Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer, Technical Report, Rensselaer Polytechnic Institute, Image Processing Laboratory, 1980. https://www.researchgate.net/publication/238720460_Octree_Encoding_A_New_Technique_for_the_Representation_Manipulation_and_Display_of_Arbitrary_3-D_Objects_by_Computer.

[48] J.L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (1975) 509–517. https://doi.org/10.1145/361002.361007

[49] G. Qian, Y. Li, H. Peng, J. Mai, H. Hammoud, M. Elhoseiny, B. Ghanem, Pointnext: revisiting pointnet++ with improved training and scaling strategies, in: NeurIPS'22, 2022. https://doi.org/10.48550/arXiv.2206.04670

[50] X. Ma, C. Qin, H. You, H. Ran, Y. Fu, Rethinking network design and local geometry in point cloud: a simple residual MLP framework, in: ICLR 2022, 2022. https://doi.org/10.48550/arXiv.2202.07123

[51] X. Wu, Y. Lao, L. Jiang, X. Liu, H. Zhao, Point transformer V2: grouped vector attention and partition-based pooling, in: NeurIPS'22, 2022. https://doi.org/10.48550/arXiv.2210.05666

[52] X. Li, Z. Zhang, Y. Li, M. Huang, J. Zhang, SFL-NET: slight filter learning network for point cloud semantic segmentation, IEEE Trans. Geosci. Remote Sens. 61 (2023) 1–14. https://doi.org/10.1109/TGRS.2023.3313876

[53] H. Thomas, Y.-H.H. Tsai, T.D. Barfoot, J. Zhang, KPConvX: modernizing kernel point convolution with kernel attention, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 5525–5535.