



**HAL**  
open science

# Corrected Laplace–Beltrami Operators for Digital Surfaces

Colin Weill-Duflos, David Coeurjolly, Jacques-Olivier Lachaud

► **To cite this version:**

Colin Weill-Duflos, David Coeurjolly, Jacques-Olivier Lachaud. Corrected Laplace–Beltrami Operators for Digital Surfaces. *Journal of Mathematical Imaging and Vision*, 2025, 67 (2), pp.11. 10.1007/s10851-024-01226-6 . hal-04913546

**HAL Id: hal-04913546**

**<https://hal.science/hal-04913546v1>**

Submitted on 3 Feb 2025

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License


# Corrected Laplace-Beltrami operators for digital surfaces

Colin Weill–Duflos<sup>1\*</sup>, David Coeurjolly<sup>2</sup> and Jacques-Olivier Lachaud<sup>3</sup>

<sup>1</sup> Université Savoie Mont Blanc, CNRS, LAMA, F-73000 Chambéry France .

<sup>2</sup> Université Lyon, CNRS, INSA, LIRIS, F-69000 Lyon France .

<sup>3</sup> Université Savoie Mont Blanc, CNRS, LAMA, F-73000 Chambéry France

 0000-0003-4236-2133.

\*Corresponding author(s). E-mail(s): [colin.weill-duflos@univ-smb.fr](mailto:colin.weill-duflos@univ-smb.fr);

Contributing authors: [david.coeurjolly@cnrs.fr](mailto:david.coeurjolly@cnrs.fr); [jacques-olivier.lachaud@univ-smb.fr](mailto:jacques-olivier.lachaud@univ-smb.fr);

## Abstract

Defining consistent calculus frameworks on discrete meshes is useful for processing the geometry of meshes or model numerical simulations and variational problems onto them. However digital surfaces (boundary of voxels) cannot benefit directly from the classical mesh calculus frameworks, since their vertex and face geometry is too poor to capture the geometry of the underlying smooth Euclidean surface well enough. This paper proposes three new calculus frameworks dedicated to digital surfaces, which exploit a *corrected normal field*, in a manner similar to the recent digital calculus of [1]. First we build a *corrected interpolated calculus* by defining inner products with position and normal interpolation in the Grassmannian. Second we present a *corrected finite element method* which adapts the standard Finite Element Method with a corrected metric per element. Third we present a *corrected virtual refinement method* adapting the method of [2]. Experiments show that these digital calculus frameworks seem to converge toward the continuous calculus, offer a valid alternative to classical mesh calculus, and induce effective tools for digital surface processing tasks. We then use these corrected Laplace-Beltrami operators in order to build a regularization method for digital surface, using geometric information given by discrete normal and curvature estimators.

**Keywords:** Digital calculus, Laplacian operator, Differential operators

## 1 Introduction

When solving differential equations on a mesh, it is often required to build a set of differential operators for this mesh. Perhaps the most commonly found is the Laplace-Beltrami operator as it is used in a wide variety of applications such as mesh editing [3, 4], mesh smoothing [5] or geodesic path approximation [6]. Building a simple graph Laplacian or discrete Laplacian does not suffice, since the mesh geometry must be taken into account.

Using a subdivision scheme and building the operators on it (as done in [7]) do not suffice either, as the limit surface does not solve the metric issues (staircase effects induced by the grid). On triangular and polygonal surfaces, several calculus frameworks produce these differential operators, such as the Finite Element Method (FEM) [8], Discrete Exterior Calculus (DEC) [9], the Virtual Element Method [10], etc (see [11] for a comparative evaluation).

Usually these frameworks operate under the assumption that the mesh interpolates the underlying “true” smooth geometry. In the case of digital surfaces made of surfels (boundary of voxels), which are frequent when processing 3D images, this assumption is false, and these frameworks fail at yielding convergent operators. However several geometric quantities can be evaluated with convergence properties, such as surface area [12], or the normal field and the curvature tensor [13, 14] on digital surfaces. We are aware of only two digital analogs to differential operators: Caissard *et al.* [15] proposed a digital Laplacian based on the heat kernel, while two of the authors have adapted in [1] the polygonal calculus of [16], by correcting its normal vector field. The digital “*Heat kernel*” Laplacian of [15] is the only one that is proven convergent and its convergence is observed through experiments. The digital “*Projected PolyDEC*” Laplacian of [1] is not pointwise convergent, but yet provides meaningful results in variational problems.

This paper proposes three new digital calculus frameworks that are constructed with a tangent space corrected by a prescribed normal vector field (e.g., the II normal estimator [13]). Tangent space correction has proven to be effective for tasks such as estimating curvatures [14] and reconstructing a piecewise smooth surface from a digital surface [17]. The first one, called “*interpolated corrected calculus*”, embeds the digital surface into the Grassmannian with a vertex-interpolated corrected normal vector field: the resulting surface is thus continuous in positions and normals. It is thus more consistent than the Projected PolyDEC, whose embedding is discontinuous between surfels. The second one, called “*corrected FEM*”, adapts the Finite Element Method with metrics tailored to a constant corrected normal vector per element. The third one adapts the virtual refinement method [2], where each face gets subdivided into triangles thanks to the addition of a virtual vertex. The three constructions are consistent with classical calculus constructions, and we hope they will allow proving the convergence of operators. For now, we conducted experiments which show that these frameworks build a consistent Laplacian, convergent when slightly diffused. We achieve results on par with [15] while retaining the ease of build and sparsity from [1].

We also build a regularization method for digital surface using corrected Laplace-Beltrami operator and geometric estimators: since we can estimate the normals  $\mathbf{n}$  and the mean curvature  $H$ , using the equality  $\Delta \mathbf{p} = -2H\mathbf{n}$  we can recover the position  $\mathbf{p}$ . We compare our method with a former method for regularization [17] based solely upon normals, and we explore the modification of curvatures during the reconstruction.

## 2 Digital calculus with corrected tangent space

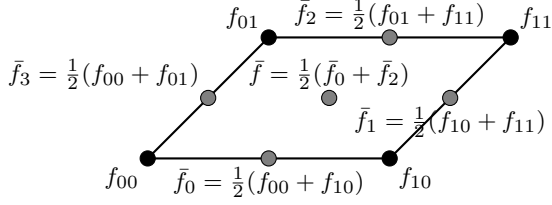
We demonstrate here that the same approach of corrected lengths and areas used in [1] can be used to build differential operators with other methods. The approach can be summarized as a correction of lengths and areas based on how orthogonal they are to the true normal. Assuming we have a vector  $\mathbf{v}$  and a normal  $\mathbf{u}$ , the corrected length of  $\mathbf{v}$  is given by  $\|\mathbf{v} \times \mathbf{u}\|$ . The corrected area of a parallelogram defined by two vectors  $\mathbf{v}$  and  $\mathbf{w}$  with normal  $\mathbf{u}$  is given by  $\det(\mathbf{u}, \mathbf{v}, \mathbf{w})$ . These can be seen as the length/areas of the projected vector/parallelogram onto the tangent plane.

Our data here will be defined by values at vertices, meaning that each face has 4 degrees of freedom. We use these degrees of freedom to build a base of functions on the mesh. These functions are bilinear on mesh elements here but other basis functions are possible (e.g., the Virtual Elements Method [10] requires to know solely the behavior of functions on edges). The methods we present, similarly to [1], use a per face construction of sparse operators.

### Notations

The parameter space of each surfel is a unit square  $\square := [0, 1]^2$  parameterized by  $s$  and  $t$ . We denote by  $\mathbf{n}$  the natural or naive normal of a surfel  $\sigma$ , that can be computed with a cross product of two consecutive edges. The corrected normal field will be denoted  $\mathbf{u}$ . Inside a surfel, we can decompose these normals in the natural base of the surfel into  $\mathbf{u} = (\mathbf{u}^x, \mathbf{u}^y, \mathbf{u}^z)$ .

A function  $f$  in a surfel  $\sigma$  is assumed to be bilinearly interpolated. We denote then by  $[f_{\square}(\sigma)] := [f_{00}(\sigma), f_{10}(\sigma), f_{11}(\sigma), f_{01}(\sigma)]^T$  the degrees of freedom of  $f$ , corresponding to its values at each vertex when circulating around  $\sigma$ . We will



**Fig. 1:** Notations for the interpolations of the values of a function  $f$  on a surfel.

often write simply  $[f_{\square}]$  when the surfel is obvious from the context. We sometimes use averages of these values, whose notations are illustrated in Figure 1.

## 2.1 Interpolated corrected calculus

We propose here a calculus where the corrected normal vector field  $\mathbf{u}(x)$  is continuous over the mesh: corrected normal vectors are given at vertices; these vectors are bilinearly interpolated within each face. Hence, within a surfel,  $\mathbf{u}(s, t) = \mathbf{u}_{00}(1-s)(1-t) + \mathbf{u}_{10}s(1-t) + \mathbf{u}_{01}(1-s)t + \mathbf{u}_{11}st$ . Although this naive bilinear interpolation does not respect the condition that normals need to be unitary vectors, it yields much simpler formulas in calculation. Furthermore, experiments show that a more complex interpolation yielding almost unit normals does not improve the results, while increasing the complexity of formulas.

The construction of the calculus is similar to the polygonal calculus of [16], building inner products, sharp and flat operators on a per face basis. However the correction of the geometry does not follow [1], but instead use an embedding of the mesh into the Grassmannian to correct the area/length measures. The Grassmannian is a way to represent affine subspaces, hence tangent spaces here. Within this space, one can define differential forms that are invariant to rigid motions (Lipschitz-Killing forms). We exploit here the *corrected area 2-form* (see [18, 14]):  $\omega_0^{\mathbf{u}}(\mathbf{x})(\mathbf{v}, \mathbf{w}) := \det(\mathbf{u}(\mathbf{x}), \mathbf{v}, \mathbf{w})$ , for  $\mathbf{v}$  and  $\mathbf{w}$  tangent vectors. As one can see, thanks to the embedding in the Grassmannian, the corrected area form can be expressed as a simple volume form (i.e. a determinant). Note that it falls back to the usual area measure  $\|\mathbf{v} \times \mathbf{w}\|$  when  $\mathbf{v}$  and  $\mathbf{w}$  are indeed orthogonal to a unit normal vector  $\mathbf{u}(\mathbf{x})$ , while it gets smaller if there is a mismatch between tangent and normal information.

We first define how we integrate a quantity  $g$  defined at vertices. In the case of a surfel  $\sigma$  with constant normal  $\mathbf{n}$  aligned with  $z$ -axis wlog, and with  $\mathbf{v} = \frac{\partial \mathbf{x}}{\partial s}$  and  $\mathbf{w} = \frac{\partial \mathbf{x}}{\partial t}$ , the corrected area form reduced on  $\square$  to  $\omega_0^{\mathbf{u}}(s, t) = \langle \mathbf{n} \mid \mathbf{u}(s, t) \rangle = \mathbf{u}^z(s, t)$ . We can now compute the integral of  $g$  inside a surfel:

$$\begin{aligned} \iint_{\square} g \omega_0^{\mathbf{u}} &:= \iint_{\square} g(s, t) \mathbf{u}^z(s, t) ds dt \\ &= [\mathbf{u}^z_{\square}]^{\top} \frac{1}{36} \begin{bmatrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{bmatrix} [g_{\square}]. \end{aligned}$$

We study now the integral quantity  $\iint \nabla \Phi \omega_0^{\mathbf{u}}$ , which is an integrated gradient corrected by the normal vector field  $\mathbf{u}$ . First of all, the scalar field  $\Phi$  will generally be defined as the bilinear interpolation of a scalar field  $\phi$  defined over the domain. Thus  $\phi(s, t) = \Phi(\mathbf{x}(s, t))$ . We relate the gradient of  $\Phi$  with the partial derivatives of  $\phi$  by writing the standard chain rule with Jacobian matrices:

$$\begin{aligned} J_{\phi}(s, t) &= J_{\Phi}(\mathbf{x}(s, t)) J_{\mathbf{x}}(s, t) \\ \Leftrightarrow \begin{bmatrix} \frac{\partial \phi}{\partial s} & \frac{\partial \phi}{\partial t} \end{bmatrix} (s, t) &= (\nabla \Phi)^{\top}(\mathbf{x}(s, t)) \begin{bmatrix} \frac{\partial \mathbf{x}}{\partial s} & \frac{\partial \mathbf{x}}{\partial t} \end{bmatrix} (s, t). \end{aligned}$$

We quite naturally extend  $\phi$  as constant along the  $\mathbf{u}$  direction. The preceding relation can now be inverted given that  $(\frac{\partial \mathbf{x}}{\partial s} = [1 \ 0 \ 0]^{\top}, \frac{\partial \mathbf{x}}{\partial t} = [0 \ 1 \ 0]^{\top}, \mathbf{u} = [\mathbf{u}^x \ \mathbf{u}^y \ \mathbf{u}^z]^{\top})$  forms a basis ( $(s, t)$  is omitted for conciseness):

$$\nabla \Phi(\mathbf{x}) = \underbrace{\begin{bmatrix} \mathbf{u}^z & 0 & 0 \\ 0 & \mathbf{u}^z & 0 \\ -\mathbf{u}^x & -\mathbf{u}^y & 1 \end{bmatrix}}_C \begin{bmatrix} \frac{\partial \phi}{\partial s} \\ \frac{\partial \phi}{\partial t} \\ 0 \end{bmatrix}.$$

It follows that  $\iint_{\square} \nabla \Phi \omega_0^{\mathbf{u}} = \iint_{\square} C \begin{bmatrix} \frac{\partial \phi}{\partial s} & \frac{\partial \phi}{\partial t} & 0 \end{bmatrix}^{\top} \mathbf{u}^z ds dt$ . Below, we explicit the vector  $\begin{bmatrix} \frac{\partial \phi}{\partial s} & \frac{\partial \phi}{\partial t} & 0 \end{bmatrix}^{\top}$  involving derivatives of  $\phi$  as

$$\begin{bmatrix} (1-t)(\phi_{10} - \phi_{00}) + t(\phi_{11} - \phi_{01}) \\ (1-s)(\phi_{01} - \phi_{00}) + s(\phi_{11} - \phi_{10}) \\ 0 \end{bmatrix}$$



$$= \underbrace{\begin{bmatrix} 1-t & 0 & -t & 0 \\ 0 & s & 0 & s-1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_B \underbrace{\begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix}}_{D_0} [\phi_{\square}].$$

The matrix  $D_0$  is the differential operator, and is common to all quad faces. We get:

$$\iint_{\square} \nabla \Phi \omega_0^{\mathbf{u}} = \underbrace{\iint_{\square} CB \mathbf{u}^z ds dt}_{\mathcal{G}_{\sigma}} D_0 [\phi_{\square}],$$

where  $\mathcal{G}_{\sigma}$  is a  $3 \times 4$  matrix whose expression is (note the use of averages):

$$\mathcal{G}_{\sigma} = \frac{1}{3} \begin{bmatrix} \bar{\mathbf{u}}^z & 0 & -\bar{\mathbf{u}}^z & 0 \\ 0 & \bar{\mathbf{u}}^z & 0 & -\bar{\mathbf{u}}^z \\ -\bar{\mathbf{u}}^x & -\bar{\mathbf{u}}^y & \bar{\mathbf{u}}^x & \bar{\mathbf{u}}^y \end{bmatrix} + \frac{1}{6} \begin{bmatrix} \bar{\mathbf{u}}_0^z & 0 & -\bar{\mathbf{u}}_2^z & 0 \\ 0 & \bar{\mathbf{u}}_1^z & 0 & -\bar{\mathbf{u}}_3^z \\ -\bar{\mathbf{u}}_0^x & -\bar{\mathbf{u}}_1^y & \bar{\mathbf{u}}_2^x & \bar{\mathbf{u}}_3^y \end{bmatrix}.$$

The (corrected) area  $a_{\sigma}$  of such a surfel  $\sigma$  has a simple expression, while a pointwise expression of the gradient  $\mathbf{G}_{\sigma}$  is obtained by normalizing  $\mathcal{G}_{\sigma}$  by the corrected area leading to:

$$a_{\sigma} := \iint_{\square} \omega_0^{\mathbf{u}} = \iint_{\square} \mathbf{u}^z ds dt = \bar{\mathbf{u}}^z, \\ \mathbf{G}_{\sigma} := \frac{1}{a_{\sigma}} \mathcal{G}_{\sigma} D_0.$$

### Sharp and flat operators.

The sharp operator transform a 1-form into a vector field. We use the expression of the pointwise gradient to raise any 1-form as a representative vector per surfel. Within a surfel, a 1-form associates a scalar value to each (oriented) edge. Let  $\beta$  be a 1-form, and  $[\beta_{\square}(\sigma)] := [\beta_0 \ \beta_1 \ \beta_2 \ \beta_3]^{\top}$  its values on the 4 edges of  $\sigma$ . Omitting the differential operator  $D_0$  in the pointwise gradient gives the representative 3D vector of  $\beta$  on surfel  $\sigma$ :

$$\beta^{\sharp}(\sigma) := \frac{1}{a_{\sigma}} \mathcal{G}_{\sigma} [\beta_{\square}(\sigma)].$$

The discrete sharp operator on  $\sigma$  is thus the  $3 \times 4$  matrix  $U_{\sigma} := \frac{1}{a_{\sigma}} \mathcal{G}_{\sigma}$ .

The flat operator projects a vector field onto the tangent plane and computes its circulation

along each edge. The 1-form  $\mathbf{v}^b$  associated with vector  $\mathbf{v}$  is thus:

$$[\mathbf{v}_{\square}^b] := \oint_{\partial f} \mathbf{t}^{\top} (I - \mathbf{u} \mathbf{u}^{\top}) \mathbf{v} \\ = \int_0^1 \begin{bmatrix} [1 \ 0 \ 0] (I - \mathbf{u}(r, 0) \mathbf{u}^{\top}(r, 0)) \mathbf{v} \\ [0 \ 1 \ 0] (I - \mathbf{u}(1, r) \mathbf{u}^{\top}(1, r)) \mathbf{v} \\ [-1 \ 0 \ 0] (I - \mathbf{u}(r, 1) \mathbf{u}^{\top}(r, 1)) \mathbf{v} \\ [0 \ -1 \ 0] (I - \mathbf{u}(0, r) \mathbf{u}^{\top}(0, r)) \mathbf{v} \end{bmatrix} dr.$$

By linearity, the flat operator  $V_{\sigma}$  is a  $4 \times 3$  matrix (see appendix for details).

### Inner products for discrete forms (i.e metrics).

The inner product between 0-forms is simply the integration of their product on the surfel  $\sigma$ . For any bilinearly interpolated functions  $\phi, \psi$ , we obtain on the surfel  $\sigma$  the scalar:

$$\langle \phi | \psi \rangle_0(\sigma) := \iint_{\sigma} \phi \psi \omega_0^{\mathbf{u}} \\ = [\phi_{\square}(\sigma)]^{\top} M_{0, \sigma} [\psi_{\square}(\sigma)].$$

The associated metric matrix is a  $4 \times 4$  symmetric matrix, called *mass matrix*, whose expression is given in the appendix. If the corrected normal vector  $\mathbf{u}$  is consistent with the naive surfel normal  $\mathbf{n}$  (i.e.  $\langle \mathbf{u}(s, t) | \mathbf{n} \rangle > 0$ ), then  $M_{0, \sigma}$  is positive definite.

We would like the inner product between 1-forms  $\beta$  and  $\gamma$  to be defined by emulating the continuous case. We integrate the scalar product between the vectors associated with the 1-forms on the surfel  $\sigma$ :

$$\langle \beta | \gamma \rangle_1(\sigma) := \iint_{\square} \langle \beta^{\sharp} | \gamma^{\sharp} \rangle \omega_0^{\mathbf{u}} \\ = [\beta_{\square}(\sigma)]^{\top} M_{1, \sigma}^{\text{naive}} [\gamma_{\square}(\sigma)].$$

Using above relations we have:

$$\langle \beta | \gamma \rangle_1(\sigma) = a_{\sigma} (U_{\sigma} [\beta_{\square}(\sigma)])^{\top} (U_{\sigma} [\gamma_{\square}(\sigma)] \gamma) \\ = [\beta_{\square}(\sigma)]^{\top} \left( \frac{1}{a_{\sigma}} \mathcal{G}_{\sigma}^{\top} \mathcal{G}_{\sigma} \right) [\gamma_{\square}(\sigma)].$$

Hence  $M_{1, \sigma}^{\text{naive}} = \frac{1}{a_{\sigma}} \mathcal{G}_{\sigma}^{\top} \mathcal{G}_{\sigma}$ ; it is a symmetric matrix. It can be verified that, if  $\mathbf{u}$  is a unit constant vector over the surfel  $\sigma$  and  $\langle \mathbf{u} | \mathbf{n} \rangle > 0$ ,

then this matrix is symmetric positive semidefinite. However, it is not definite. To remedy this, we follow [16] and complement the definition to get the *stiffness matrix* as

$$M_{1,\sigma} := \frac{1}{a_\sigma} \mathcal{G}_\sigma^\top \mathcal{G}_\sigma + \lambda(I - U_\sigma V_\sigma). \quad (1)$$

### Calculus on the whole mesh.

Let  $n$ ,  $m$  and  $k$  be respectively the number of vertices, edges and faces of the mesh. Let  $\mathcal{V}$  be the space of all sampled functions (an  $n$ -dimensional vector space), and  $\mathcal{E}$  be the space of all discrete 1-forms (an  $m$ -dimensional vector space). Global operators sharp  $U$  (size  $3k \times m$ ), flat  $V$  (size  $m \times 3k$ ), mass matrix  $M_0$  (size  $n \times n$ ) and stiffness matrix  $M_1$  (size  $m \times m$ ), differential  $D_0$  (size  $m \times n$ ) are obtained by merging the corresponding local operators  $U_\sigma, V_\sigma, M_{0,\sigma}, M_{1,\sigma}, D_0$  on the corresponding rows and columns.

### Codifferentials and Laplacian.

We build the 1-codifferential  $\delta_1 : \mathcal{E} \rightarrow \mathcal{V}$  by adjointness in our inner products.

$$\begin{aligned} \forall f \in \mathcal{V}, \forall \alpha \in \mathcal{E}, \\ \langle D_0 f \mid \alpha \rangle_1 = - \langle f \mid \delta_1 \alpha \rangle_0 \Leftrightarrow (D_0 f)^\top M_1 \alpha \\ = -f^\top M_0 \delta_1 \alpha. \end{aligned}$$

Being true for all pairs  $(f, \alpha)$ , it follows that  $\delta_1 := -M_0^{-1} D_0^\top M_1$ . The *Laplacian operator*  $\Delta_0$  is the composition of the codifferential and the differential, i.e.

$$\Delta_0 := \delta_1 D_0 = -M_0^{-1} D_0^\top M_1 D_0.$$

Since it is very costly to build the matrix  $M_0^{-1}$ , we will generally not use the two operators  $\delta_1$  and  $\Delta_0$  as is when solving numerical problems, but we will rather work with their “integrated” version ( $M_0 \delta_1$  and  $M_0 \Delta_0$ ). We can now see another approach to compute a Laplace-Beltrami operator coming from the Finite Elements framework.

## 2.2 Generalization to Finite Element Method

We show here how to adapt the standard Finite Element Method (FEM), e.g. see [8], in order to

solve a Poisson problem. The method builds a stiffness matrix  $L$  and a mass matrix  $M$  to transform the Poisson problem into a linear problem. We will see also that a Laplace operator can be obtained with the same method. Our adaptation consist in correcting the metric used, changing the formulas used for derivatives and dot products. While we only demonstrate here how to correct FEM on a Poisson problem, other problems can also be corrected with the same metric.

The Poisson problem is formulated as solving for  $g$  in  $\Delta g = f$ , with a given boundary constraint for  $g$  if the domain has a boundary, or with a fixed value somewhere if the domain has no boundary. The weak formulation of this problem is given by: solve for  $f$

$$\int_\Omega \nabla g \cdot \nabla \Phi = - \int_\Omega f \Phi + \int_{\partial\Omega} \Phi \langle \nabla g, \mathbf{n} \rangle, \quad (2)$$

for any  $\Phi$ . In our case, we will evaluate against  $\Phi$  the locally bilinear functions inside each element. The third term is dependent on the boundary condition, and we will make it vanish here for now.

The FEM approach consists in discretizing the problem at nodes and splitting the domain into elements bordered by nodes (quads here): functions  $g$  (say) are discretized at these nodes as vectors  $\mathbf{g}$  of their values at nodes. FEM assumes bilinear interpolation of functions within elements. It builds a stiffness matrix  $L$  and a mass matrix  $M$  such that  $L\mathbf{g} = M\mathbf{b}$ . This corresponds to the first two terms in (2). Boundary constraints are integrated in this linear problem, either by removing rows and columns or by setting equalities. We can then solve the Poisson problem by solving the linear system  $L\mathbf{g} = M\mathbf{b}$ , but we can also deduce a Laplacian operator  $\Delta := M^{-1}L$ .

The matrices are built quad by quad, so here per surfel. We start by defining a metric  $G$  per surfel, since it depends on the corrected normal  $\mathbf{u}$ , then using this metric in the formulas for derivatives and scalar products when building the matrices. Our reference element is a unit square in the plane  $\square$ . We obtain:

$$G = \begin{bmatrix} 1 - (\mathbf{u}^x)^2 & -\mathbf{u}^x \mathbf{u}^y \\ -\mathbf{u}^x \mathbf{u}^y & 1 - (\mathbf{u}^y)^2 \end{bmatrix}.$$

Since we assume now that our corrected normal field is constant on the surfel, the metric is also constant. This is an arbitrary choice we make in order to keep formulas simple. It becomes easy

to compute the gradient and Laplacian. We use the formula  $df(\mathbf{w}) = \langle \nabla f, \mathbf{w} \rangle_G$  for any vector  $\mathbf{w}$ , with  $\langle \cdot, \cdot \rangle_G$  the inner product. It follows that  $\nabla f = G^{-1} \left[ \frac{\partial f}{\partial s} \quad \frac{\partial f}{\partial t} \right]^\top$ . For the Laplacian, since the metric is constant, we use  $\Delta f = \nabla \cdot \nabla f$ . We write them more explicitly as:

$$\nabla f = \frac{1}{(\mathbf{u}^z)^2} \left[ \begin{array}{c} (1 - u_y^2) \frac{\partial f}{\partial s} + \mathbf{u}^x \mathbf{u}^y \frac{\partial f}{\partial t} \\ \mathbf{u}^x \mathbf{u}^y \frac{\partial f}{\partial s} + (1 - (\mathbf{u}^x)^2) \frac{\partial f}{\partial t} \end{array} \right] \quad (3)$$

$$\Delta f = \frac{1}{(\mathbf{u}^z)^2} \left( (1 - (\mathbf{u}^y)^2) \frac{\partial^2 f}{\partial s^2} + 2\mathbf{u}^x \mathbf{u}^y \frac{\partial^2 f}{\partial s \partial t} + (1 - (\mathbf{u}^x)^2) \frac{\partial^2 f}{\partial t^2} \right) \quad (4)$$

We choose a basis of bilinear functions on the square as our basis functions. This choice can be disputed: while linear functions are still harmonic regarding to the Laplacian in (4), bilinear functions are no longer harmonics in this setting. However, finding a way to build hat functions that stay harmonic in this setting is not obvious. Yet bilinear functions are still used on quad meshes that are not rectangular and where the same reasoning can be applied to show that they are not harmonic. We define the four basis functions as  $f_0 = (1 - s)(1 - t)$ ,  $f_1 = s(1 - t)$ ,  $f_2 = st$ ,  $f_3 = (1 - s)t$ .

In order to build our stiffness matrix we evaluate:  $\int_{\square} \langle \nabla f, \nabla p \rangle_G$  for any bilinear  $f$  and  $p$ . The local stiffness matrix  $L_M$  is given in (5) of Figure 2. The global stiffness matrix  $L$  is then obtained by summing over all the local stiffness matrices. The local mass matrix  $M_M$  is computed from  $\int_{\Omega} fp$ , with  $f$  and  $p$  bilinear:

$$M_M = \frac{\mathbf{u}^z}{36} \begin{bmatrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{bmatrix}. \quad (6)$$

We recognize the standard mass matrix for quad mesh with a factor correcting the area of the surfel. It is the same as  $M_0$  for constant  $\mathbf{u}$ . The global mass matrix  $M$  is obtained similarly by summing over all the local mass matrices.

## 2.3 Generalization to Virtual Refinement Method

Here we show how we can adapt the method from Bunge et al. [2] for surfels with corrected normals.

The method can be summarized as: each face has a virtual point added to it, that is then used as a basis for the triangulation the face. The point is chosen as the one minimizing the squared areas of the created triangles. Then, the point is expressed as a linear combination of the face. The weights used in the linear combination are used to create a prolongation operator. The stiffness matrix are then built on the triangulation, and the polygonal versions of those matrices are deducted using the prolongation.

First, we find the position of the virtual point inside each face, by minimizing the squared areas of the 4 created triangles. For the corrected normals, we use normals interpolated at vertices, otherwise a constant normal per face would lead to a point at the center of the face every time and wouldn't take advantage of this step. Thus our goal is to find  $s_0$  and  $t_0$ . Our goal is to minimize the sum of the squared areas of the triangles  $A_0^2 + A_1^2 + A_2^2 + A_3^2$  (figure 3). The area of these triangles can be expressed as an integral, recalling that the corrected area form  $\omega_0^{\mathbf{u}}(\mathbf{x}(s, t))$  reduces to  $\mathbf{u}^z(s, t)$  on a surfel:

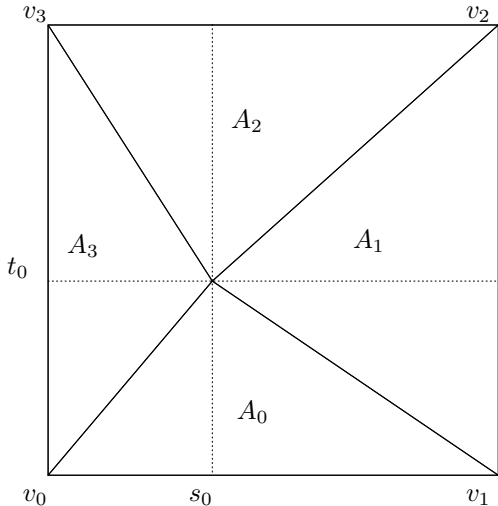
$$A_0 = \int_0^{s_0} \int_0^{\frac{s_0}{s_0} t_0} u_z(s, t) dt ds + \int_{s_0}^1 \int_0^{\frac{1-s_0}{1-s_0} t_0} u_z(s, t) dt ds. \quad (7)$$

The three other areas have similar form. The component  $u_z$  being bilinear in  $s$  and  $t$ , the total area to optimize is given by a 6th order polynomial, and the exact solution is hard to obtain. Instead, we use a simple gradient descent to minimize it: using randomly generated values for the normals, we see that it generally converges in a few steps and always stays inside the surfel.

Once we have  $s_0$  and  $t_0$ , we have our virtual point  $p$  as a bilinear combination of the vertices of the face. We can directly deduce the weights and build the projection operator  $P$ . For the vertices  $v_0, v_1, v_2, v_3$  that make up the surfel, the corresponding weights are  $w_0 = (1 - s_0)(1 - t_0)$ ,  $w_1 = s_0(1 - t_0)$ ,  $w_2 = s_0 t_0$ ,  $w_3 = (1 - s_0)t_0$ . The

$$L_M = \frac{1}{6\mathbf{u}^z} \begin{bmatrix} 3\mathbf{u}^x\mathbf{u}^y + 2 + 2(\mathbf{u}^z)^2 & 2(\mathbf{u}^y)^2 - 1 - (\mathbf{u}^x)^2 & 1 - 3\mathbf{u}^x\mathbf{u}^y - (\mathbf{u}^z)^2 & 2(\mathbf{u}^x)^2 - 1 - (\mathbf{u}^y)^2 \\ 2(\mathbf{u}^y)^2 - 1 - (\mathbf{u}^x)^2 & 2 - 3\mathbf{u}^x\mathbf{u}^y + 2(\mathbf{u}^z)^2 & 2(\mathbf{u}^x)^2 - 1 - (\mathbf{u}^y)^2 & 3\mathbf{u}^x\mathbf{u}^y + 1 - (\mathbf{u}^z)^2 \\ 2 - 3\mathbf{u}^x\mathbf{u}^y - (\mathbf{u}^z)^2 & 2(\mathbf{u}^x)^2 - 1 - (\mathbf{u}^y)^2 & 3\mathbf{u}^x\mathbf{u}^y + 2 + 2(\mathbf{u}^z)^2 & 2(\mathbf{u}^y)^2 - 1 - (\mathbf{u}^x)^2 \\ 2(\mathbf{u}^x)^2 - 1 - (\mathbf{u}^y)^2 & 3\mathbf{u}^x\mathbf{u}^y + 1 + (\mathbf{u}^z)^2 & 2(\mathbf{u}^y)^2 - 1 - (\mathbf{u}^x)^2 & 2 - 3\mathbf{u}^x\mathbf{u}^y + 2(\mathbf{u}^z)^2 \end{bmatrix}. \quad (5)$$

**Fig. 2:** Local stiffness matrix  $L_M$  for the corrected finite element method.



**Fig. 3:** We find  $s_0$  and  $t_0$  such that the sum of the squared areas  $A_0^2 + A_1^2 + A_2^2 + A_3^2$  is minimized.

prolongation operator then needs to assign values to vertices and the virtual vertices from the vertices: for a single face, it would be a  $5 \times 4$  matrix of the form

$$P_f = \begin{bmatrix} w_0 & w_1 & w_2 & w_3 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The global prolongation operator  $P \in \mathbb{R}^{n_v+n_f \times n_v}$  is built on the same principle: the first  $n_f$  rows have non-zero values on the columns whose indices correspond to the vertices of the face, with the values being the  $w_i$  of the virtual point of this face associated to the vertex corresponding to the row. The following  $n_v$  rows correspond to the identity matrix.

For the mass matrix, we simply apply the area formula of (7). For the stiffness matrix, we could try to again use a finite element approach, however trying to evaluate  $\int \nabla g \cdot \nabla f dA$  for  $f$  and  $g$  two linear functions, and using the correction metric, is

quite complex. Instead, we adapt the metric version of the cotan formula. For a triangle with edges  $l_0$ ,  $l_1$  and  $l_2$  (opposite to vertices  $v_0$ ,  $v_1$  and  $v_2$  respectively), the coefficient between two different vertices  $i$  and  $j$  is given by  $(-l_k^2 + l_i^2 + l_j^2)/(4A)$ .

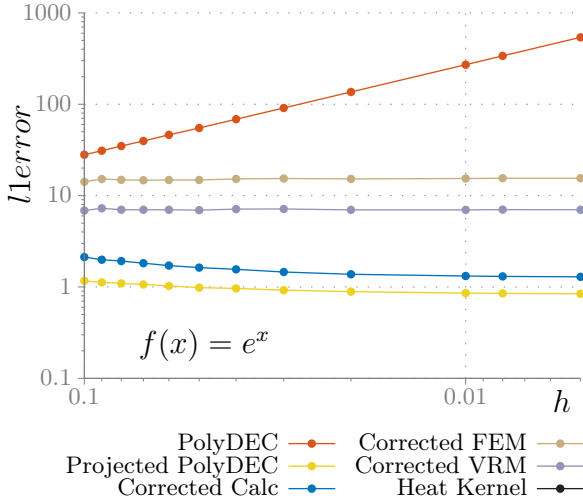
Once we have built the mass matrix  $M_\Delta$  and  $L_\Delta$  on the triangulation, using our prolongation operator we compute  $M = P^\top M_\Delta P$  and  $L = P^\top L_\Delta P$ . We now have three ways of building a sparse Laplace-Beltrami operator. We see now how they compare against operators from previous works. We compare the method ‘‘PolyDEC’’ [16], its corrected version ‘‘Projected PolyDEC’’ [1], the interpolated corrected calculus ‘‘Corrected Calculus’’, the corrected finite element method ‘‘Corrected FEM’’, the corrected virtual refinement method ‘‘Corrected VRM’’ and the heat kernel method ‘‘Heat Kernel’’ [15].

## 3 Evaluations and comparisons

We compare the resulting operators and the ones from previous works on several use cases: first on the sphere, with forward evaluation (compute the Laplacian of a function), backward evaluation (solve a Poisson problem getting a function back from its Laplacian), eigenvalue comparisons, and then on a standard mesh by comparing with the results obtained on an underlying triangle mesh. Plots related to digitized spheres are the means of the results of 32 computations for each step, each conducted with a different center to better take into account the variability in sphere discretizations.

### 3.1 Forward evaluation

Several previous works tried to evaluate the quality and convergence of the Laplacian operator when used in a forward manner: from  $f$  defined on the mesh, we compute  $\Delta f$  both analytically and with a discrete Laplacian, then compare the two results. In other words, if our stiffness matrix

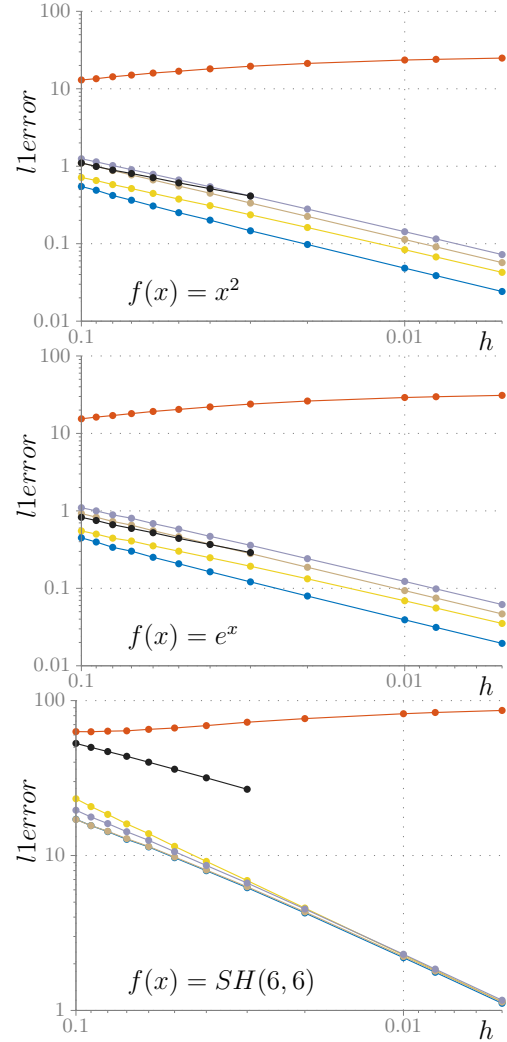


**Fig. 4:** Naive forward evaluation of the mesh Laplacian on an exponential function. No convergence is observed.

is called  $L$  and our mass matrix  $M$ , we solve the equation  $LF = MX$  where  $X$  is the unknown.

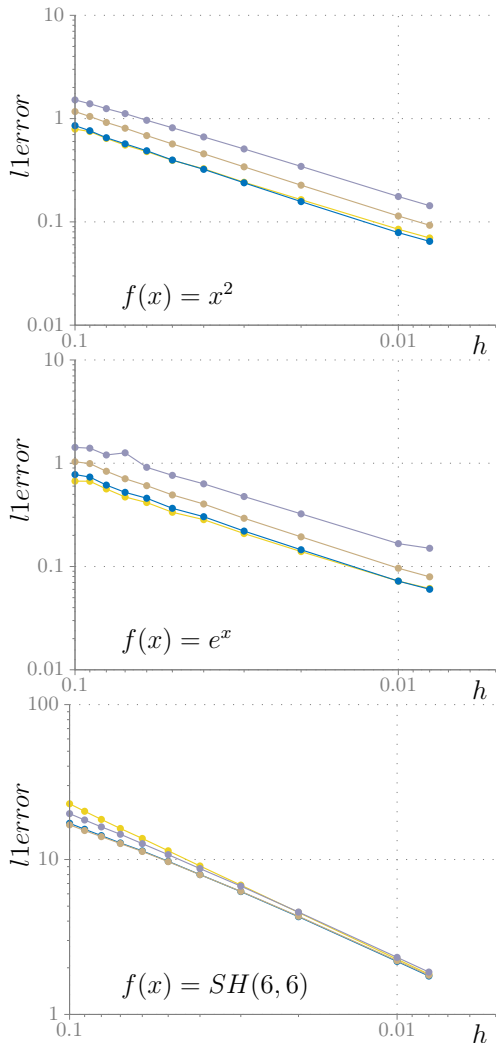
A naive approach consists in computing  $X = M^{-1}LF$ . This is the one that was used for evaluation in previous works, and was not convergent when using sparse operators. We reproduce this behavior by computing the Laplacian of  $f(x) = e^x$  using various methods, none of which seem to converge (see Figure 4). This is disappointing since we expect the Laplacian operator to have linear convergence when evaluated in forward manner, as observed in [15] and proven for the mesh Laplacian [19] on triangle mesh.

Our idea for improving the convergence consists of adding a small diffusion step to the result. It suffices to replace the mass matrix  $M$  by  $M - dtL$ . In other words, instead of evaluating  $X = M^{-1}LF$ , we evaluate  $X = (M - dtL)^{-1}LF$ . The result depends on the choice of parameter  $dt$ : we found that we approach linear convergence when  $dt$  is in the order of  $h$ , and the best quality for  $dt = 0.035h$ . This means that we add a diffusion step with a characteristic length of order  $h^{\frac{1}{2}}$ , which is coherent with results from other works. Using this method, we achieve what seems to be linear convergence on different functions (Figure 5), with results comparable or even higher quality than in [15]. We run the same experiment as figure 5, with estimated normal vectors (using Integral



**Fig. 5:** Forward evaluation of the mesh Laplacian with added diffusion on a quadratic function (top), exponential function (middle) and the sixth spherical harmonic (bottom). Adding diffusion significantly improves the results, achieving linear convergence on the sphere. We achieve similar rates of convergence as Heat Kernel [15], with a better quality on less smooth functions. The Heat Kernel method is, however, limited to a grid step of  $h \geq 0.03$ , due to its enormous memory usage.

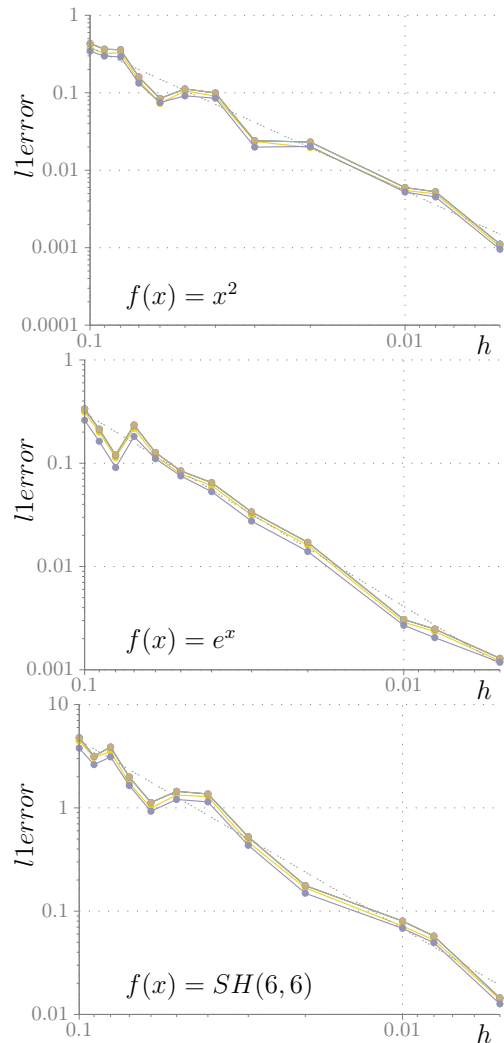
Invariant [13]) instead of ground truth. Results are shown in Figure 6, and also approach linear convergence.



**Fig. 6:** Evaluation of the Laplacian using the integral invariant normal estimator. Estimated normals give slightly worse results than with true normals, yet the Laplacian seem to converge too. Computations are performed for grid steps  $h \geq 0.008$  because integral invariant normal estimator is very slow on finer grids.

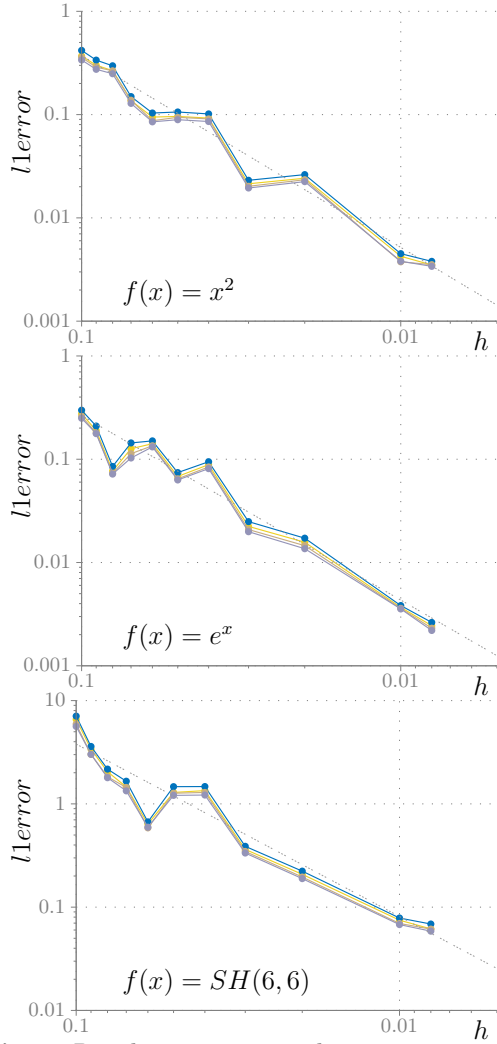
### 3.2 Backward evaluation

A Laplacian is often built to solve a Poisson problem. We evaluate a function on our digital surface, we also evaluate its Laplacian using an exact formula, then we compute an approximation of the original function that we compare to the exact original. It is a criterion used for Laplacian evaluation (see [11]), which has not yet been done for digital Laplacians. It also makes more sense



**Fig. 7:** Error when solving a Poisson problem with different Laplacians. We approach a quadratic convergence rate.

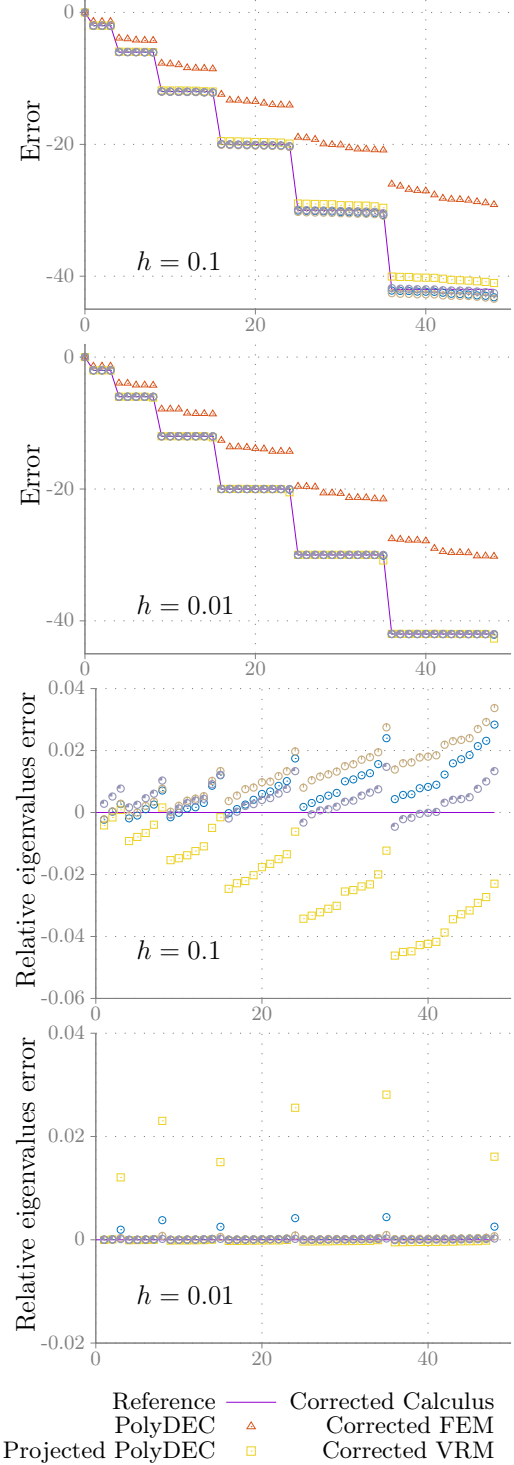
to evaluate the Finite Element Methods in this case than in forward evaluation, as this is the problem the operator is built for and is proven (in the case of standard regular meshes) to converge. We find that all methods give roughly the same results (figure 7). They seem to be convergent, with a rate around  $h^{1.9}$ , which is coherent with the theoretical quadratic rate. Again, we run the experiment using the Integral Invariant estimators and approach similar rate of convergence (Figure 8).



**Fig. 8:** Results using integral invariant estimated normals for solving a Poisson problem.

### 3.3 Eigenvalues

We follow the evaluation of eigenvalues on the spherical harmonics used in [11]. Since the spherical harmonics have analytic expressions, we can compare the eigenvalues of our operator to the exact eigenvalues of the Laplace-Beltrami operator on the sphere. To obtain these eigenvalues, we solve for  $\lambda$  in the following generalized eigenvalue problem  $L\mathbf{u} = \lambda M\mathbf{u}$ . Figure 9 shows the first eigenvalues of our Laplacians on the unit sphere with discretization steps  $h = 0.1$  and  $h = 0.01$ . The PolyDEC method [16] is not accurate, but corrected methods are, with accuracy increased at finer resolutions.



**Fig. 9:** The smallest 49 eigenvalues of the Laplacian on unit sphere with discretization step 0.1 and 0.01. Relative error is given by  $(\hat{\lambda} - \lambda)/\lambda$ , where  $\hat{\lambda}$  is the approximated eigenvalue and  $\lambda$  the correct value.



### 3.4 Comparison to the cotan Laplacian

Until now all our comparisons were made on a digitized sphere: this is because there are some closed form expressions of Laplacians, and its eigen decomposition is well studied. However, the sphere is a very specific case, and our evaluations may not reflect well more general cases. We compare here our operators to the results obtained on a regular, high quality triangle mesh with the standard cotan Laplacian. To do so, we use a refined version of a triangle mesh (at 100000 vertices), and equivalent digital surfaces at different resolutions ( $128^3$ ,  $256^3$ ,  $512^3$ ). Then we built a projection operator allowing us to map values on the high resolution mesh to the digital one (orthogonal projection and linear interpolation). We also use this projection operator to map the normal vector field computed on the mesh to the surfels or vertices in the calculus frameworks. We then compute a function on the triangle mesh as well as its Laplacian using the cotan Laplacian on the triangle mesh [4], and then their projection on the digital surface, which we use as "ground truth". Forward evaluation results are shown on figure 10. We use the same diffusion constant as previously ( $0.035h$ ). Error is significantly reduced with the discretization step of the digital surface, suggesting that our operator is convergent toward the result given by the cotan Laplacian.

As a more complex testbed for our corrected differential operators, we propose in the following section a method for mesh regularization that relies on differential calculus.

## 4 Surface regularization from normal vectors and curvatures

Since we can estimate normal vectors, curvatures, and build a corrected Laplace-Beltrami operator even with bad positions, we can leverage the equality  $\Delta \mathbf{p} = -2H\mathbf{n}$  in order to recover consistent positions after estimating normals and curvatures. Another way to view this approach is to say that we use our mean curvature estimator along with the normal estimator in order to build delta coordinates which encode the deviation of each point

to the barycenter of its neighbors. In this situation, following the argument from [20], we need less precision in order to recover the result since encoding delta coordinates requires less precision than encoding the absolute position. Our approach can also include curvature modifications steps if needed.

### 4.1 Digital surface regularization





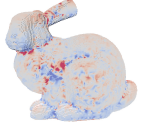





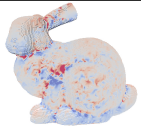




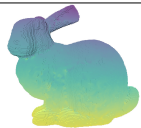
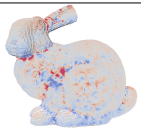
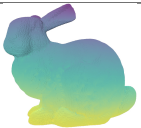
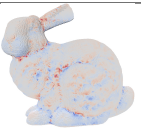


We describe here a process of regularization for digital surfaces. The steps are the following: we first compute integral invariants [13] (with  $\alpha = 1/3$ ) for normals  $\mathbf{n}$  on the digital surface, and then use corrected curvature measures [18] (with radius  $r = 0.1$ ) for estimation of the mean curvature  $H$ . We also build a corrected Laplacian from the corrected normals and the digital surfaces. We could then minimize the energy  $\|\Delta \mathbf{p}' - H\mathbf{n}\|^2$ . However, similarly to some high pass quantization that does not preserve low frequencies, the results we have shown good features but tends to deviate from the original surface (figure 11). In the case of digital surfaces, the error in position is in high frequencies, so it makes sense to use this energy to fix the high frequencies: in order to also match in lower frequencies by minimizing

$$\|\Delta \mathbf{p}' - H\mathbf{n}\|^2 + \alpha \|\mathbf{p}' - \mathbf{p}\|^2,$$

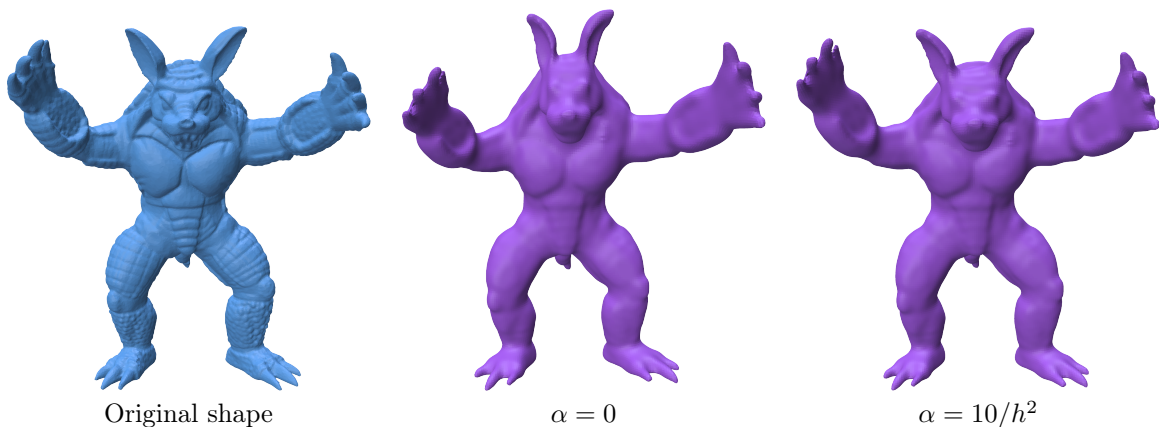
where  $\mathbf{p}$  are the original positions and  $\alpha$  is a weight between the two energies. The  $\alpha$  coefficient has to be homogeneous to the inverse of a square length, so we take it of the form  $\alpha = \frac{\alpha_0}{h^2}$  ( $h$  being the discretization step) and we use  $\alpha_0 = 10$ . This energy is quadratic so it can be minimized in a single step by computing

$$\mathbf{p}' = -\frac{1}{2}(L^\top M^{-1}L + \alpha M)^{-1}(LH\mathbf{n} - \alpha M\mathbf{p}).$$

We compare our results with the ones obtained using the method of Coeurjolly et al. [17] (figure 12). Their method also uses an input corrected normal field, and also includes a term attaching positions to the original ones: however, in their case, it is used in order to avoid shrinkage (the rest of the energy can be minimized by just shrinking the surface) while we only use this term in order to avoid a low frequencies deviation (removing this term yields a non-degenerate solution). Our results are visually similar to the one obtained

	resolution: 128	$l_1$ error	resolution: 256	$l_1$ error	resolution: 512	$l_1$ error
Ground truth						
FEM		 0.0484979		 0.0299988		 0.0194369
Corrected Calculus		 0.0473037		 0.0311042		 0.0197104
Poly Proj		 0.0474556		 0.0301771		 0.0195026

**Fig. 10:** Comparison between classical cotan Laplacian of a function defined on a triangle mesh and the digital Laplacian operators. All three operators have comparable performances. The error decreases as the resolution increases, suggesting that all three operators are convergent.



**Fig. 11:** Regularization of a  $256 \times 256 \times 256$  voxelization of the armadillo mesh with and without a position attachment term. While most of the features of the shape are obtained, the members highly deviate from the original position when  $\alpha = 0$

by [17]. We compare the use of non-corrected and corrected operators, by using non-corrected ones built using the method from de Goes et al. [16] and the normal corrected variants from Coeurjolly and Lachaud [1] (figure 13). Compared to the results obtained using corrected operators, the non-corrected one look inflated: this is explained

by the lack of correction in the mass matrix, meaning that the estimated area of the shape is taken to be the same as the area of the digital surface which is significantly larger than the area of the actual surface. However, since the non-corrected Laplacian does not change the geometry of the surfels, the quads resulting from the regularization

have similar edge lengths: if this is a desired feature for the result,  $\alpha$  can be increased in order to make the result match more closely to the digital surface and counter the expansion effect caused by overestimated area.

A drawback of our method is that the recovered surface tends to be smoother than the original (figure 14), due to the nature of the normal and curvature estimators which smoothes the actual curvature and normals.

We do some experiments on known surfaces in order to measure the accuracy of the method: we first voxelize the surface, then we regularize the digital surface and then measure the distance of the regularized surface to the original one. We measure the l1 distance and the difference between the of the regularization and of the original surface: both errors are reduced as the grid step diminishes (figure 15).

## 4.2 Mesh reconstruction from normal fields

We explore the possibility of using corrected operators when using other geometries than digital surface, or other normal fields than one estimated from an estimator ran on positions. Such a case arises when using surfaces with normal maps. We use a high resolution surface in order to build a lower resolution surface with a normal map. The latter is a much more compact data structure than the former. Our goal is to build back the original surface from the lower resolution one and the normal map.

As a proof of concept, we start from a coarse mesh with a normal map. We then build a voxelization of this coarse mesh, which we then subdivide. We compute a projection for the center of each surfel onto the coarse mesh in order to sample the normal map on the coarse mesh, thus emulating a normal map directly defined on the digital surface. The procedure is illustrated on Figure 16. This case illustrates a situation where the positions highly deviate from the underlying surface, and where the reconstruction tends to be unstable. As a way to stabilize this procedure, we clamp the curvature estimator as it introduces very high incorrect values at some points. Our method is able to recover the original surface with only a few defects visible. Using non-corrected operators induce a far worse reconstruction.

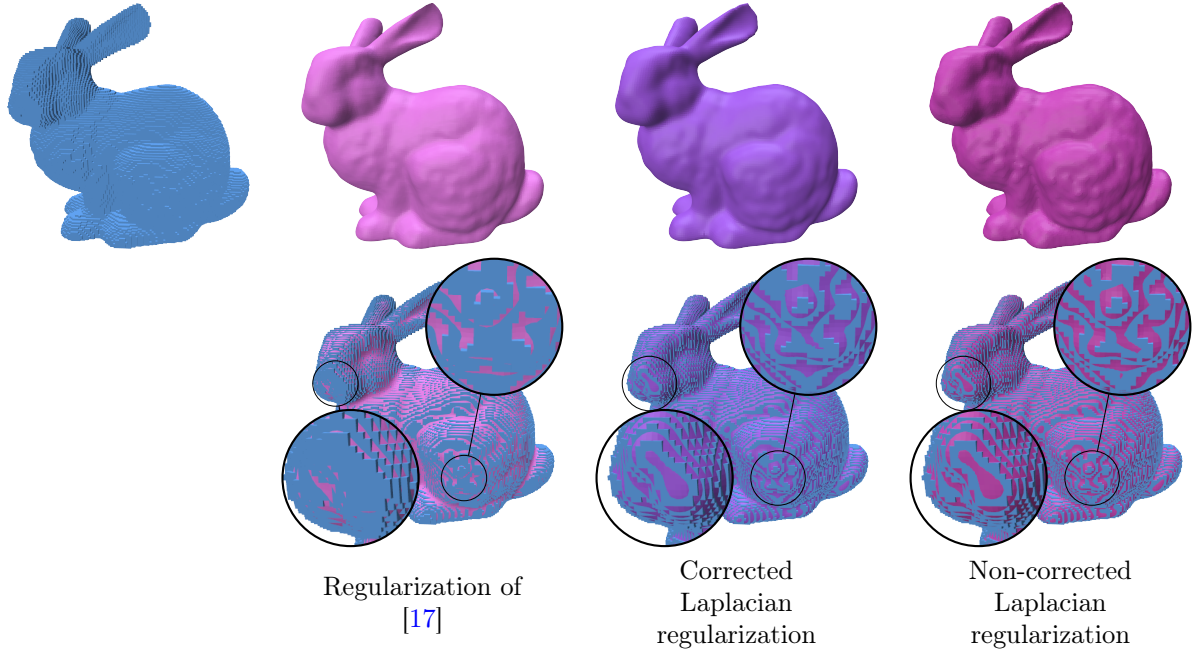
## 4.3 Curvature edition

Since we use normals and curvature to recover positions, we can modify the curvature and recover a shape with this curvature. A simple way to do this is through the modification of the  $H$  component. Since we do not modify the normals and we still try to match the original positions, the resulting surface doesn't exactly have the target curvature: as an example, we can multiply the curvature by a constant, which in theory only rescales the shape by the same amount, while in our case this exaggerates the features of the shape.

Thus, our reconstruction can include various curvature modifications such as curvature amplification or curvature clamping (figure 17). This can be used in order to exaggerate some features, or instead to smooth some part of the shape.

As a way to further modify the curvature, we can directly manipulate the shape operator  $S := d\mathbf{n}$ . It is a symmetric  $3 \times 3$  matrix, with at most two non-zeros eigenvalues  $k_1$  and  $k_2$  corresponding to the principal curvatures. Their eigenvectors correspond to the principal curvature directions. It can usually be obtained from classical curvature estimators, such as corrected curvature measure. The last eigenvalue of the resulting matrix is generally not zero however (corresponding to the normal direction): this is fixed by computing instead  $S + K\mathbf{nn}^\top$ , where  $K$  is a big constant (like  $10^6$ ) and then ignoring the largest eigenvalue. The two remaining eigenvalues can then be modified, and a shape operator  $S'$  can be built. Since  $d\mathbf{n} = S$ , we can integrate our shape operator  $S'$  in order to build  $\mathbf{n}'$  (again, we also add a term to attach the new normals to the original normals), which we then normalize. We then evaluate  $\mathbf{p}'$  from  $\mathbf{n}'$  and  $H' = \text{Tr}(S')$ . The procedure is summarized in algorithm 4.3.

This procedure does not make the curvature exactly match the input: for example, if we try to set Gaussian curvature to 0 everywhere, since the Gauss-Bonnet theorem states that  $\int_{\Omega} K dA = 2 - 2g$  with  $g$  the genus, a surface homeomorphic to a sphere ( $g = 0$ ) cannot have 0 Gaussian curvature everywhere. In practice, setting one of the two curvature components to 0 everywhere gives a smoother surface. Curvatures can also be modified in order to, for example, exaggerate the strongest principal curvature (see figure 18).



**Fig. 12:** We use several regularization methods on a voxelization of the bunny shape on a  $256 \times 256 \times 256$  grid. Left: original shape, middle-left: regularization of [17], middle-right: our regularization with a corrected Laplacian, right with a non-corrected Laplacian. The bottom row shows that our results do not shrink compared to the ones obtained by Coeurjolly et al. [17]. Here, the difference between corrected and non-corrected is visually small.

---

**Algorithm 1:** Normal and surface regularization

---

```

Data:  $P, F, N, S, \alpha$  //digital surface of
        positions  $P$  and faces  $F$ , normal
        fields  $N$ , shape operator field  $S$ 
Result:  $P'$  //Corrected positions
 $L, M \leftarrow$ 
    BuildCorrectedLaplacian( $P, F, N$ );
 $Div \leftarrow$ 
    BuildCorrectedDivergence( $P, F, N$ ); //Build
    corrected operators
 $dS \leftarrow Div(S)$ ;
//Minimize  $\|M^{-1}LN' - dS\|^2 + \alpha\|N' - N\|^2$ 
 $N' \leftarrow (LM^{-1}L + \alpha M)^{-1}(-2LdS + \alpha MN)$ ;
 $H \leftarrow \text{trace}(S)$  //Compute mean curvature
 $N' \leftarrow \text{normalize}(N')$ ;
//Minimize  $\|M^{-1}LP' - HN'\|^2 + \alpha\|P' - P\|^2$ 
 $P' \leftarrow (LM^{-1}L + \alpha M)^{-1}(-2LHN' + \alpha MP)$ ;
return  $P'$ ;

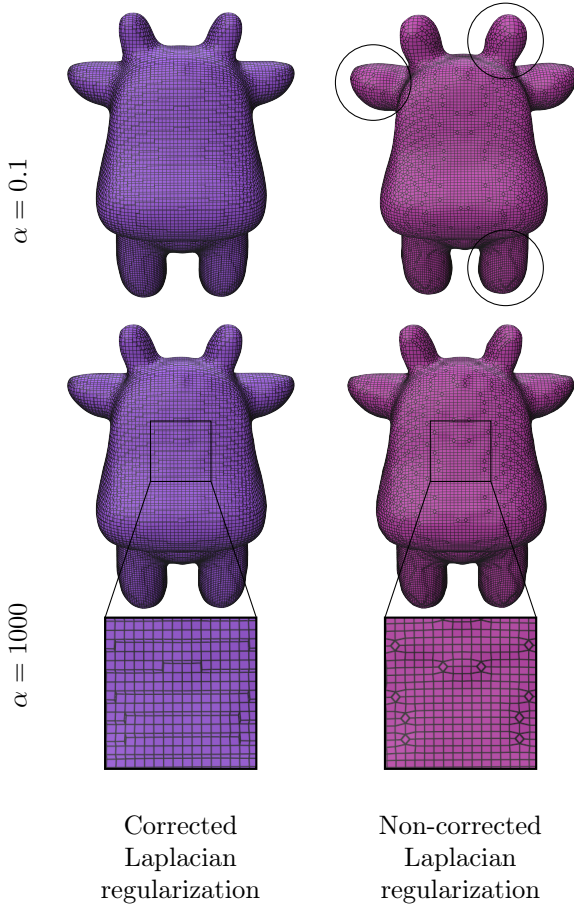
```

---

## 5 Conclusion

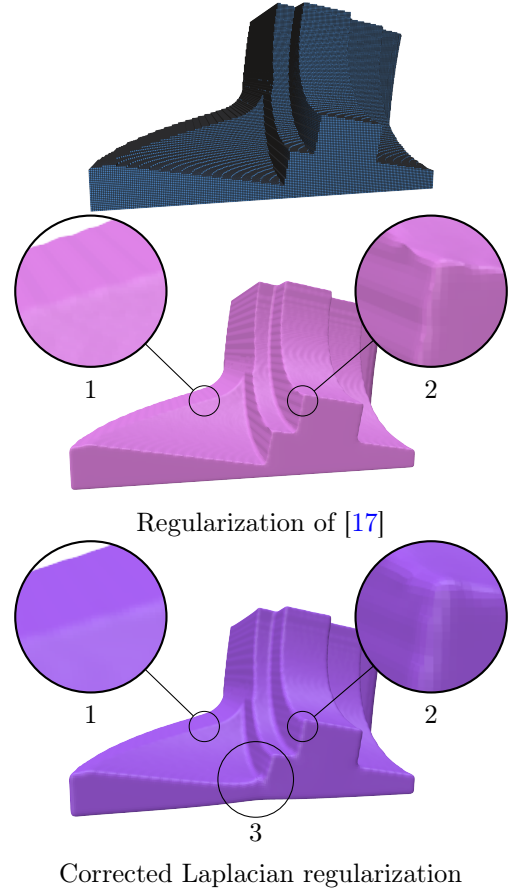
We show that, similarly to the corrected PolyDEC method [1], a corrected normal field can be inserted within discrete calculus frameworks yielding different Laplace-Beltrami operators. All these operators seem to converge when solving Poisson problems, and when used in a forward evaluation, the addition of a slight diffusion also seems to make them convergent. A limit of our study is that these results are only experimental: only the Heat kernel Laplacian of [15] is yet proven to converge on digital surfaces (strong consistency). However, since results on a common framework (the finite element method) seem promising, it may be interesting to see if the proof of convergence from this framework can be adapted to digital surfaces. The same type of calculus construction could also be tested outside digital surfaces, for instance on a triangle mesh with a corrected normal field or a normal field of much higher resolution such as a normal map (as done in [18]). The idea of adding diffusion and modifying the mass matrix





**Fig. 13:** Left: regularization with a corrected Laplacian vs right with a non-corrected one, both on a  $256 \times 256 \times 256$  voxelization of the spot shape. The corrected Laplacian shows the pattern given if the digital surface was projected onto the underlying surface, while the non-corrected one provides a surface where each edge has approximately the same length. The top row uses  $\alpha = 0.1$  and the bottom one  $\alpha = 1000$ : when using low  $\alpha$  (so a weak attach to the original positions), the non-corrected Laplacian regularization is visually worse than the corrected one. However, if the regularity of edge lengths is a desired effect,  $\alpha$  can be increased to decrease the influence of the Laplacian.

can be seen as similar to the approach of [21]. However Caissard *et al.* [15] were not able to reproduce their experiments and expected convergence, probably because digital surfaces do not have the mesh regularity required by the proof. Indeed, from our metric  $G$  it is easy to find that mesh



**Fig. 14:** Top: original shape (voxelization of the fan disk shape on a  $256 \times 256 \times 256$  grid). Middle: regularization from [17]. Bottom: our regularization using a corrected Laplacian. Our method exhibits less sharp angles (2), but tends to be smoother and to avoid stair-like effects on parts of the surface that should be flat (1). Our method is also sensible to artifacts from curvature estimations: the bottom side isn't flat and there is a visible bump inside an angle (3).

regularity means that  $\frac{1}{|u^z|}$  is bounded. Such a condition can be fulfilled for some specific meshes (such as a digital plane), but is not guaranteed on surface digitization in general (such as a sphere).

#### Acknowledgments

This work is supported by the French National Research Agency in the framework of the « France 2030 » program (ANR-15-IDEX-0002),

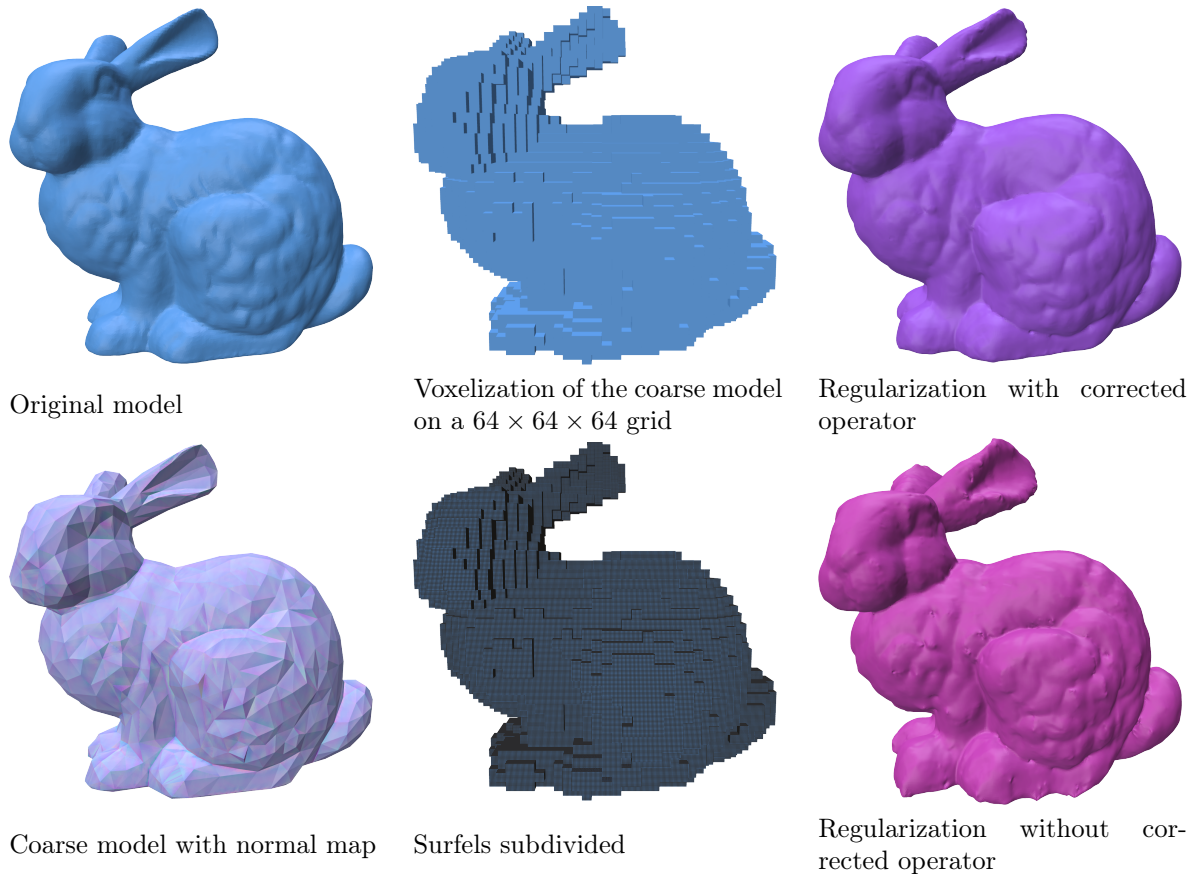
grid size :		64x64x64	128x128x128	256x256x256	512x512x512
sphere	position error	0.00583	0.00289	0.00138	0.000688
	normal error	0.0548	0.0416	0.0333	0.0233
bunny	position error	0.00457	0.00235	0.00118	0.000184
	normal error	0.239	0.158	0.0972	0.0614
spot	position error	0.00370	0.00188	0.000942	0.000510
	normal error	0.152	0.1014	0.0783	0.0569

**Fig. 15:** Error obtained on the regularization of digital surfaces by comparing with the surface that was voxelized. The error is first obtained pointwise by computing the distance of each point of the regularization to the original surface (and by taking the difference of the normals for the normal error) and then integrated into the  $l_1$  error.

by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) and by the StableProxies project (ANR-22-CE46-0006).

## References

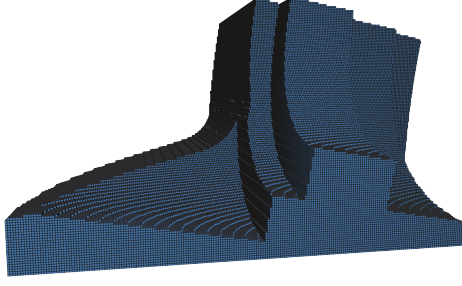
- [1] Coeurjolly, D., Lachaud, J.-O.: A simple discrete calculus for digital surfaces. In: Baudrier, É., Naegel, B., Krähenbühl, A., Tajine, M. (eds.) *Discrete Geometry and Mathematical Morphology. Lecture Notes in Computer Science*, vol. 13493, pp. 341–353. Springer, Cham (2022)
- [2] Bunge, A., Herholz, P., Kazhdan, M., Botsch, M.: Polygon Laplacian made simple. *Computer Graphics Forum* **39**(2), 303–313 (2020) <https://doi.org/10.1111/cgf.13931>
- [3] Sorkine, O.: Laplacian mesh processing. In: Chrysanthou, Y., Magnor, M. (eds.) *Eurographics 2005 - State of the Art Reports*, pp. 53–70. The Eurographics Association, Goslar, DEU (2005). <https://doi.org/10.2312/egst.20051044>
- [4] Lévy, B., Zhang, H.: Spectral mesh processing. In: *ACM SIGGRAPH 2010 Courses*, pp. 1–312 (2010)
- [5] Nealen, A., Igarashi, T., Sorkine, O., Alexa, M.: Laplacian mesh optimization. In: *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia. GRAPHITE '06*, pp. 381–389. Association for Computing Machinery, New York, NY, USA (2006)
- [6] Crane, K., Weischedel, C., Wardetzky, M.: The heat method for distance computation. *Commun. ACM* **60**(11), 90–99 (2017)
- [7] Goes, F., Desbrun, M., Meyer, M., DeRose, T.: Subdivision exterior calculus for geometry processing. *ACM Trans. Graph.* **35**(4), 1–11 (2016)
- [8] Reddy, J.N.: *Introduction to the Finite Element Method*. McGraw-Hill Education, New York (2019). <https://www.accessengineeringlibrary.com/content/book/9781259861901>
- [9] Hirani, A.N.: *Discrete exterior calculus*. PhD thesis, California Institute of Technology, Pasadena, CA, USA (2003). AAI3086864
- [10] Veiga, L., Brezzi, F., Cangiani, A., Manzini, G., Marini, L., Russo, A.: Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences* **23** (2012)
- [11] Bunge, A., Botsch, M.: A survey on discrete Laplacians for general polygonal meshes. *Computer Graphics Forum* **42**(2), 521–544 (2023)
- [12] Lachaud, J.-O., Thibert, B.: Properties of Gauss digitized shapes and digital surface integration. *Journal of Mathematical Imaging and Vision* **54**(2), 162–180 (2016)
- [13] Lachaud, J.-O., Coeurjolly, D., Levallois, J.: Robust and convergent curvature and normal estimators with digital integral invariants. In: Najman, L., Romon, P. (eds.) *Modern Approaches to Discrete Curvature. Lecture Notes in Mathematics*, vol. 2184, pp. 293–348. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-58002-9\\_9](https://doi.org/10.1007/978-3-319-58002-9_9) . [https://doi.org/10.1007/978-3-319-58002-9\\_9](https://doi.org/10.1007/978-3-319-58002-9_9)
- [14] Lachaud, J.-O., Romon, P., Thibert, B.: Corrected curvature measures. *Discrete & Computational Geometry* **68**(2), 477–524 (2022)



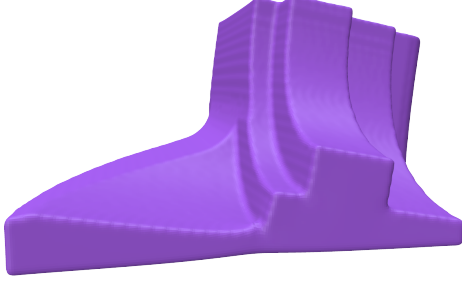
**Fig. 16:** Meshes used in order to emulate a normal map on a surfel mesh, and then the use of our method to recover the original geometry. Using non-corrected operators results in a geometry with bad visual artifacts, suggesting that the corrected operator stabilizes the method. Surfels have been subdivided 3 times, the radius used for corrected curvature estimation is 0.2 and the curvature is clamped to be at most 10 in absolute value.

- [15] Caissard, T., Coeurjolly, D., Lachaud, J.-O., Roussillon, T.: Laplace–Beltrami operator on digital surfaces. *Journal of Mathematical Imaging and Vision* **61**(3), 359–379 (2019)
- [16] De Goes, F., Butts, A., Desbrun, M.: Discrete differential operators on polygonal meshes. *ACM Transactions on Graphics (TOG)* **39**(4), 110–1 (2020)
- [17] Coeurjolly, D., Lachaud, J.-O., Gueth, P.: Digital surface regularization with guarantees. *IEEE Trans. Vis. Comput. Graph.* **27**(6), 2896–2907 (2021)
- [18] Lachaud, J.-O., Romon, P., Thibert, B., Coeurjolly, D.: Interpolated corrected curvature measures for polygonal surfaces. *Comput. Graph. Forum* **39**(5), 41–54 (2020)
- [19] Belkin, M., Sun, J., Wang, Y.: Discrete laplace operator on meshed surfaces. In: *Proceedings of the Twenty-Fourth Annual Symposium on Computational Geometry. SCG '08*, pp. 278–287. Association for Computing Machinery, New York, NY, USA (2008). <https://doi.org/10.1145/1377676.1377725> . <https://doi.org/10.1145/1377676.1377725>
- [20] Sorkine, O., Cohen-Or, D., Toledo, S.: High-pass quantization for mesh encoding. In: *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing. SGP '03*, pp. 42–51. Eurographics Association, Goslar, DEU (2003)

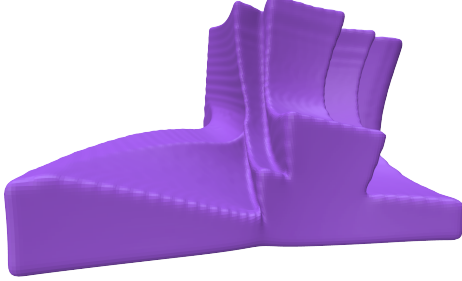




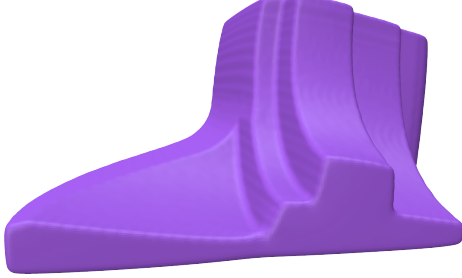
Original shape



No modification



$H \leftarrow 2H$



$H > 50 \Rightarrow H \leftarrow 50$

**Fig. 17:** Various reconstructions of the fandisk shape with mean curvature manipulations.

[21] Hildebrandt, K., Polthier, K.: On approximation of the Laplace-Beltrami operator and the Willmore energy of surfaces. Computer Graphics Forum (2011)

## A Details on the interpolated corrected calculus

Let  $\sigma$  be a surfel aligned with  $x$  and  $y$  and with normal aligned with  $z$ . The flat operator has the following expression:

$$V_\sigma := \frac{1}{6} [V_1 \ V_2 \ V_3] \quad \text{with:}$$

$$V_1 := \begin{bmatrix} 6 - 2((\mathbf{u}_{00}^x)^2 + \mathbf{u}_{00}^x \mathbf{u}_{10}^x + (\mathbf{u}_{10}^x)^2) \\ -(2\mathbf{u}_{10}^x + \mathbf{u}_{11}^x) \mathbf{u}_{10}^y - (\mathbf{u}_{10}^x + 2\mathbf{u}_{11}^x) \mathbf{u}_{11}^y \\ 2((\mathbf{u}_{01}^x)^2 + \mathbf{u}_{01}^x \mathbf{u}_{11}^x + (\mathbf{u}_{11}^x)^2) - 6 \\ (2\mathbf{u}_{00}^x + \mathbf{u}_{01}^x) \mathbf{u}_{00}^y + (\mathbf{u}_{00}^x + 2\mathbf{u}_{01}^x) \mathbf{u}_{01}^y \end{bmatrix}$$

$$V_2 := \begin{bmatrix} -(2\mathbf{u}_{00}^x + \mathbf{u}_{10}^x) \mathbf{u}_{00}^y - (\mathbf{u}_{00}^x + 2\mathbf{u}_{10}^x) \mathbf{u}_{10}^y \\ 6 - 2((\mathbf{u}_{10}^y)^2 + \mathbf{u}_{10}^y \mathbf{u}_{11}^y + (\mathbf{u}_{11}^y)^2) \\ (2\mathbf{u}_{01}^x + \mathbf{u}_{11}^x) \mathbf{u}_{01}^y + (\mathbf{u}_{01}^x + 2\mathbf{u}_{11}^x) \mathbf{u}_{11}^y \\ 2((\mathbf{u}_{00}^y)^2 + \mathbf{u}_{00}^y \mathbf{u}_{01}^y + (\mathbf{u}_{01}^y)^2) - 6 \end{bmatrix}$$

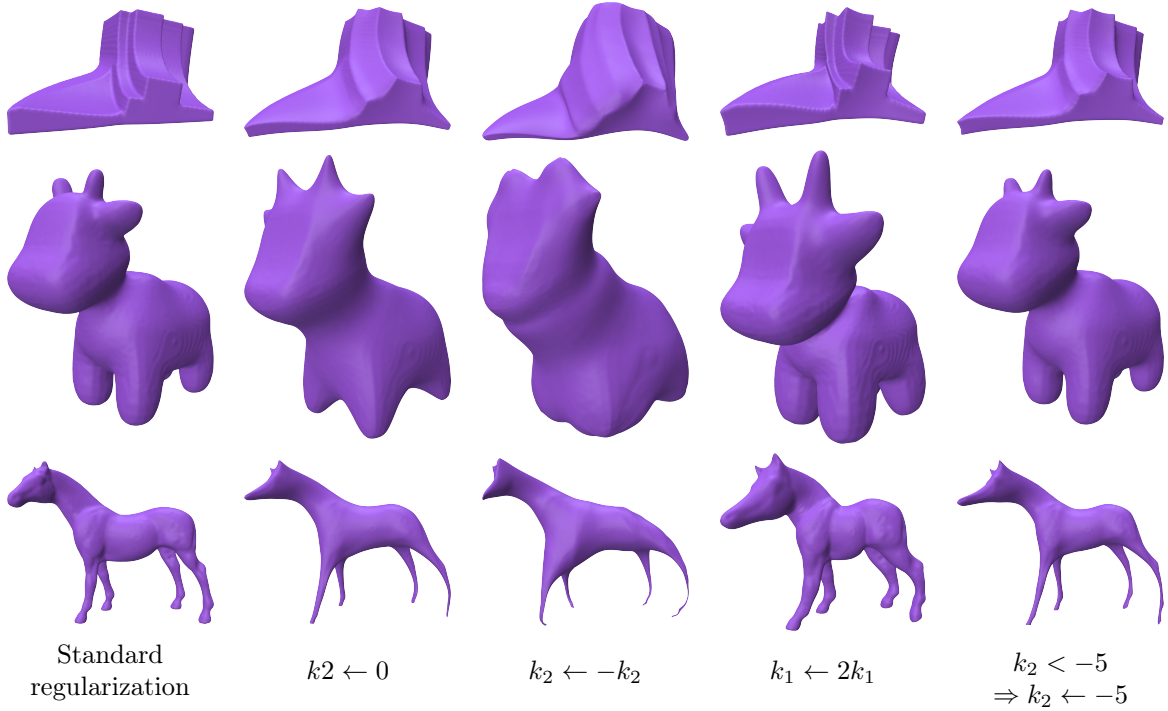
$$V_3 := \begin{bmatrix} -(2\mathbf{u}_{00}^x + \mathbf{u}_{10}^x) \mathbf{u}_{00}^z - (\mathbf{u}_{00}^x + 2\mathbf{u}_{10}^x) \mathbf{u}_{10}^z \\ (2\mathbf{u}_{10}^y + \mathbf{u}_{11}^y) \mathbf{u}_{10}^z - (\mathbf{u}_{10}^y + 2\mathbf{u}_{11}^y) \mathbf{u}_{11}^z \\ (2\mathbf{u}_{01}^x + \mathbf{u}_{11}^x) \mathbf{u}_{01}^z + (\mathbf{u}_{01}^x + 2\mathbf{u}_{11}^x) \mathbf{u}_{11}^z \\ (2\mathbf{u}_{00}^y + \mathbf{u}_{01}^y) \mathbf{u}_{00}^z + (\mathbf{u}_{00}^y + 2\mathbf{u}_{01}^y) \mathbf{u}_{01}^z \end{bmatrix}$$

The metric matrix for 0-forms is defined as the matrix such that, for any bilinearly interpolated functions  $\phi, \psi$ , we obtain on surfel  $\sigma$  the scalar:

$$\langle \phi | \psi \rangle_0(\sigma) := \iint_\sigma \phi \psi \omega_0^{\mathbf{u}} = [\phi_{\square}(\sigma)]^\top M_0 [\psi_{\square}(\sigma)].$$

Let us now define weighted sums for components of  $\mathbf{u}$  over the quad. We number the edges when turning along the boundary of the surfel  $\sigma$  from 0 to 3, such that edges 0,1,2,3 connect vertex pairs  $(\mathbf{x}_{00}, \mathbf{x}_{10}), (\mathbf{x}_{10}, \mathbf{x}_{11}), (\mathbf{x}_{01}, \mathbf{x}_{11}), (\mathbf{x}_{01}, \mathbf{x}_{00})$ , respectively. We define

$$\begin{aligned} \bar{\mathbf{u}}_{00} &:= 9\mathbf{u}_{00} + 3\mathbf{u}_{10} + \mathbf{u}_{11} + 3\mathbf{u}_{01} \\ \bar{\mathbf{u}}_{10} &:= 3\mathbf{u}_{00} + 9\mathbf{u}_{10} + 3\mathbf{u}_{11} + \mathbf{u}_{01} \\ \bar{\mathbf{u}}_{11} &:= \mathbf{u}_{00} + 3\mathbf{u}_{10} + 9\mathbf{u}_{11} + 3\mathbf{u}_{01} \\ \bar{\mathbf{u}}_{01} &:= 3\mathbf{u}_{00} + \mathbf{u}_{10} + 3\mathbf{u}_{11} + 9\mathbf{u}_{01} \\ \bar{\mathbf{u}}_{00,10} &:= 3\mathbf{u}_{00} + 3\mathbf{u}_{10} + \mathbf{u}_{11} + \mathbf{u}_{01} \\ \bar{\mathbf{u}}_{10,11} &:= \mathbf{u}_{00} + 3\mathbf{u}_{10} + 3\mathbf{u}_{11} + \mathbf{u}_{01} \\ \bar{\mathbf{u}}_{11,01} &:= \mathbf{u}_{00} + \mathbf{u}_{10} + 3\mathbf{u}_{11} + 3\mathbf{u}_{01} \\ \bar{\mathbf{u}}_{01,00} &:= 3\mathbf{u}_{00} + \mathbf{u}_{10} + \mathbf{u}_{11} + 3\mathbf{u}_{01} \end{aligned}$$



**Fig. 18:** Various curvature editions of  $k_1, k_2$  the principal curvatures, with  $k_1 > k_2$ . These are applied during the regularization process starting from voxelized shapes on a  $256 \times 256 \times 256$  grid.

By integration of the left-hand side, we obtain for a surfel with normal  $z$ :

$$\mathbf{M}_0 = \frac{1}{144} \begin{bmatrix} \bar{\mathbf{u}}_{00}^z & \bar{\mathbf{u}}_{00,10}^z & 4\bar{\mathbf{u}}^z & \bar{\mathbf{u}}_{01,00}^z \\ \bar{\mathbf{u}}_{00,10}^z & \bar{\mathbf{u}}_{10}^z & \bar{\mathbf{u}}_{10,11}^z & 4\bar{\mathbf{u}}^z \\ 4\bar{\mathbf{u}}^z & \bar{\mathbf{u}}_{10,11}^z & \bar{\mathbf{u}}_{11}^z & \bar{\mathbf{u}}_{11,01}^z \\ \bar{\mathbf{u}}_{11,01}^z & 4\bar{\mathbf{u}}^z & \bar{\mathbf{u}}_{11,01}^z & \bar{\mathbf{u}}_{01}^z \end{bmatrix}.$$