UNIVERSITÉ SAVOIE MONT BLANC

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ SAVOIE MONT BLANC

Spécialité : **Mathématiques-Informatique**

Arrêté ministériel : 25 Mai 2016

Présentée par

Colin WEILL--DUFLOS

Thèse dirigée par **Jacques-Olivier LACHAUD** et
codirigée par **David COEURJOLLY**

préparée au sein du **LAMA, Université Savoie Mont Blanc**
dans **l'École Doctorale MSTII**

L'opérateur de Laplace-Beltrami et ses applications au traitement de géométrie discrète

Thèse soutenue publiquement le **12 décembre 2024**,
devant le jury composé de :

M. Dmitry SOKOLOV
Professeur des universités Université de Lorraine, Loria, Rapporteur
M. Hugues TALBOT
Professeur CentraleSupelec, Rapporteur
Mme Céline ROUDET
Maîtresse de conférences Université de Bourgogne, LIB, Examinateur
M. Boris THIBERT
Professeur Université Grenoble Alpes, LJK, Examinateur
M. Jacques-Olivier LACHAUD
Professeur Université de Savoie Mont-Blanc, LAMA, Directeur de Thèse
M. David COEURJOLLY
Directeur de recherche CNRS, LIRIS, Directeur de Thèse

UNIVERSITÉ
SAVOIE
MONT BLANC

Contents

French overview

0.1 Introduction

L'opérateur de Laplace-Beltrami, utilisé dans de nombreuses EDP telles que l'équation de la chaleur ou l'équation des ondes, a été discrétisé très tôt à l'aide de réseau éléctriques afin d'approximer des solutions de ces EDP. Ici, nous nous concentrons sur ses usages en traitement de géométrie, où l'on utilise des outils et des concepts mathématiques afin de manipuler des objets 3D comme des maillages surfaciques, des maillages volumiques ou des nuages de points. Un problème qui représente bien ceux que l'on trouve dans ce domaine est celui de la reconstruction d'une surface depuis un nuage de points scannés, où les points peuvent par exemple être groupés en triangles avec des méthodes telles que le plane fitting, ou bien où le bruit des données scannées peut être retiré avec des méthodes similaires à l'analyse de Fourier. Dans ce champs de recherche, l'opérateur de Laplace-Beltrami fait partie des objets mathématiques les plus employés, avec des applications dans de nombreuses méthodes telle que le lissage de surface, la compression de maillage, l'approximation de géodésiques, la déscription de formes et la correspondance de formes.

Étant donné que nous travaillons sur des objets discret (et, dans le cadre de cette thèse, des surfaces digitales spécifiquement), il nous faut une version discrète de l'opérateur de Laplace-Beltrami. La discrétisation de cette opérateur présente plusieurs difficultés. Idéalement, l'opérateur approxime son équivalent lisse: si l'on augmente la résolution de la surface, les résultats obtenus avec l'opérateur discret devraient approcher ceux de l'opérateur lisse. Dans le cas de la discrétisation la plus courante pour les surfaces triangulaires, le Laplacian cotan, c'est effectivement vrai dans un sens faible pour la plupart des surfaces exceptées certaines comme la lanterne de Schwarz, dont les normales ne convergent pas vers celle de la surface sous-jacente.

Notre travail se concentre sur les surfaces digitales. Ces surfaces se trouvent en géométrie digital, où l'on travail sur des sous-ensemble de $\mathbb{Z}^3$. Alors que dans les cas des surfaces triangulaires des normales non convergentes correspond à un cas pathologique, ici c'est l'inverse: on peut généralement supposer que les normales ne convergent pas. Il en résulte que pour des estimateurs de quantités simple telle que la longueur ou l'aire, obtenir des garanties de convergence n'est pas simple, mais pas impossible non plus: des estimateurs conrgent existent, ainsi que pour les normales et la courbure.

Notre but est d'utiliser ces estimateurs afin de développer des méthodes d'analyse pour les surfaces digitales qui se comportent autant que possible comme dans le cas lisse. De telles méthodes ont déjà été développées, et ont même des garanties de convergence pour l'opérateur de Laplace-Beltrami: elles sont cependant lentes à construire (car requierent la distance géodésique entre chaque pair de point de la surface) et produisent des opérateurs denses et donc occupant beaucoup de mémoire et prenant du temps à inverse. Nous souhaitons donc développer des opérateurs sparses à l'aide de méthode de construction locale qui peuvent, à l'opposer des opérateurs dense, rapidement être construits et facilement inversés.

Nous proposons plusieurs adaptations de méthodes existantes pour des géométrie où l'on fournit un champ de normal additionel. Nous vérifions que ces méthodes se comportent comme voulu. Nous utilisos aussi ces méthodes afin de construire un opérateur de Laplace-Beltrami qui l'on utilise avec des estimateurs de normales et de courbure afin de développer une méthode de régularisation des surfaces digitales.

Nous étudions également l'adaptions de fonctionnelles d'Ambrosio-Tortorelli afin de produire des UV maps. Les UV maps sont des applatissement en 2d de surfaces 3d, principalement utilisées afin d'appliquer des textures (stockées sous forme d'images) sur une surface, avec des discontinuités. ignera en tant que coupures. Lors de la construction de UV maps, le but est de minimiser la distorsion de la parametrisation tout en minimisant également la longueur des coupures. Minimiser la distorsion permet d'éviter d'avoir des triangles dont les répresentations dans le plan sont à des échelles différentes et requiert donc des résolutions dimages différentes, et permet également les modifications faites dirèctement sur la texture à correspondre à la modification résultante sur la surface. On veut également éviter d'avoir trop de longueur de coupure, qui permettent certe des réduire la distorsion mais rende la texture plus dure à manipuler. En optimisant nos version adaptées de la fonctionnelle d'Ambrosio-Tortorelli, on obtient une optimisation simultanée de la distorsion et des coupures de la parametrisation. On observe que différentes discrétisations présentes différents avantages, notamment en terme de vitesse et d'injectivité.

0.2 L'opérateur de Laplace-Beltrami

Définissons d'abord l'opérateur que nous voulons étudier. Rappelons d'abord la définition du Laplacien, valable dans les espaces euclidiens:

$$\Delta = \sum_i \frac{\partial^2}{\partial x_i^2}$$

Cet opérateur peut se généraliser avec l'opérateur de Laplace-Beltrami, pouvant notamment être utilisé sur des surfaces, via la formule $\Delta = \operatorname{div} \circ \operatorname{grad}$. Une autre généralisation plus large encore est celle de l'opérateur de Laplace-de Rahm, faisant intervenir la différentielle d et la codifférentielle δ, deux opérateurs pouvant être définis via le calcul exterieur. Cet opérateur est défini comme $\Delta = d\delta + \delta d$, et se simplifie en $\Delta = \delta d$ lorsqu'appliqué aux 0-formes (correspondants aux champs de scalaires). Il peut alors également se décomposer sous la forme $\Delta = \star d \star d$ où $\star$ est encore un opérateur défini dans le calcul discret, appelé étoile de Hodge.

Cet opérateur intervient dans des équations connues telle que l'équation de la chaleur $-\Delta T = \frac{\partial T}{\partial t}$ (l'opérateur charactérise ainsi la diffusion) et l'équation des ondes $\Delta f = \frac{\partial^2 f}{\partial t^2}$. Une analyse des solutions stationnaires de l'équation des ondes, c'est à dire de la form $f(\mathbf{p}, t) = g(\mathbf{p})h(t)$, révèle que la partie spatiale g de ces solutions doit correspondre à des vecteurs propres de l'opérateur. En étudiant les vecteurs propres de cet opérateur, on se rend compte que dans le cas Euclidien ils correspondent aux éléments qui composent la base de Fourier. L'étude de cet opérateur peut ainsi permettre de généraliser les bases de Fourier à des espaces non Euclidiens. Une question qui peut se poser à partir des valeurs propres/vecteurs propres de l'opérateur de Laplace-Beltrami est: suffisent-ils à retrouver la surface d'où l'opérateur a été construit? Elle a été formulée dans le papier "Can one hear the shape of a drum?" de M. Kac [Kac66], puis une réponse négative a été apportée par Gordon et al. [GW96], qui fournissent plusieurs formes avec le même spectre.

0.2.1 Applications de l'opérateur en traitement de géométrie

Nous nous intéressons ici à l'utilisation de cet opérateur en traitement de géométrie. Le traitement de géométrie désigne le domaine regroupant la géométrie et l'informatique autour des formes 3d généralement utilisées en informatique graphique. On retrouve parmis ces formes les surfaces triangulaires, les surface polygonales, les surfaces implicites, les maillages tétrahédriques, les voxels, les nuages de points... Parmis les problèmes dans ce domaine, on trouve la reconstruction d'une surface depuis un nuage de point, le débruitage de surface et la simplification de surface.

Nous nous concentrerons ici aux applications liées aux surfaces. L'opérateur de Laplace-Beltrami apparaît fréquemment dans ce domaine: cela s'explique notamment par son lien avec l'énergie de Dirichlet. Cette énergie sert à mesurer, pour une fonction u, à quel point cette fonction est lisse et vaut $\int |\nabla u|^2$. Les minimiseurs de cette énergies correspondent, à un terme de bord prêt, aux fonctions harmoniques c'est à dire de Laplacien nul. Nous regardons maintenant quelques usages de cet opérateur en traitement de surface.

Lissage de surface Le lissage de surface peut s'appuyer sur le *flot de courbure moyenne* $\Delta \mathbf{p} = -2H\mathbf{n}$, où $\mathbf{p}$ sont les positions, H la courbure moyenne et $\mathbf{n}$ les normales de la surface. Ce flot correspond au gradient de l'énergie de membrane sur le domaine Ω: $E_A(\Omega) := \int_\Omega dA$. Minimiser cette énergie tend à rapprocher la surface à une sphère (et donc à éliminer les creux et les bosses), avec cependant l'inconvénient de rétrécir la surface. Plusieurs méthodes sont dérivées de cette approche, notamment celle de Taubin [Tau95] et de Desbrun et al. [Des+99].

Compression de surface L'objectif ici est de représenter un ou plusieurs maillages avec le moins de mémoire possible. On peut pour cela employer les *coordonnées delta*, qui encodent la déviation d'un sommet au barycentre de ces voisins. A partir de coordonnées delta et d'un Laplacien, il est possible d'intégrer ces coordonnées afin de retrouver la surface originale. C'est la méthode proposée par Sorkine et al. [SCT03], qui argumentent qu'en quantifiant ces coordonnées sur quelques bits, l'erreur obtenue est peu visible et se concentrent surtout sur des déviations dans les basses fréquences.

Approximation de géodésiques On peut estimer les géodésiques grâce à l'équation de la chaleur. En effet, le relation suivante lie le noyau de la chaleur $k_{t,x}$ et la distance géodésique d:

$$d(x,y) = \lim_{t\to 0} \sqrt{-4t \log kt, x(y)}.$$

Plutôt que d'exploiter directement cette relation (difficile à cause des imprécisions numériques), Crane et al. [CWW17] proposent une méthode d'intégration du flot de la chaleur en 3 étapes:

1. Intégrer le flux de la chaleur $\frac{\partial u}{\partial t} = \Delta u$ pour une durée t fixée

2. Calculer le champs de vecteurs $X = -\nabla u / |\nabla u|$
3. Résoudre l'équation de Poisson $\Delta\Phi = \nabla \cdot X$

Le champs scalaire Φ obtenu donne une approximation de la distance aux sources de la chaleurs choisies pour l'étape 1.

Analyse spectrale Plusieurs approches reposent sur le spectre de l'opérateur de Laplace-Beltrami, qui permet de faire de l'analyse de Fourier sur des surfaces. On peut par exemple utiliser les vecteurs propres comme base afin de décomposer un signal définit sur la surface et de filtrer certaines fréquences. Les outils obtenus à partir de l'analyse spectrale ont l'avantage de ne pas dépendre de la triangulation, uniquement de la surface triangulée. Puisque l'opérateur de Laplace-Beltrami est invariant aux isométries, son spectre est également assez robuste aux variations de pose. Lévy et Zhang [LZ10] décrivent plusieurs de ces applications: la paramétrisation de surface, le remaillage, le clustering, la segmentation, la correspondance de formes...

Signature de points, descripteurs de surface A partir du spectre de l'opérateur de Laplace-Beltrami, ou bien de celui d'opérateurs faisant intervenir celui de Laplace-Beltrami, plusieurs outils de descriptions de surfaces ou de signature de points ont été développé. L'objectif est de charactériser des points ou des surfaces afin de facilement pouvoir faire la correspondance avec d'autres points ou autres surfaces, par exemple pour réaliser un "Shape Google" [Bro+11]. On peut citer le descripteur de surface ShapeDNA [RWP06], basé sur les premières valeurs propres de l'opérateur de Laplace-Beltrami, ou bien Global Point Signature [Rus07] qui échantillonne les valeurs des vecteurs propres à certains endroits de la surface.

Correspondance de surfaces On cherche à déterminer, pour deux maillages triangulaires correspondant à la même surface (mais pas forcément maillée de la même manière, ou bien déformée), une correspondance sommet à sommet entre ces deux surfaces. La méthode de Lombart et al. [Lom+13] effectue ainsi une correspondance via les plus basses fréquences du spectre. Les fonctional maps [Ovs+17; Ovs+12] sont un outil qui permet également de faire de la correspondance de surfaces. Afin de faire correspondre une fonction f définie sur une surface M à une fonction g définie sur une surface N, on détermine une correspondance entre une base de fonctions sur M et une base sur N. Il est courant de prendre les vecteurs propres du Laplacien comme base. La partie cruciale de la construction des fonctional maps est le calcul de la correspondance (sous forme d'une matrice C), souvent faite en minimisant l'énergie qui mesure à quel point le mapping commute avec le Laplacien de chaque domaine.

Ces méthodes se basent donc toutes sur la possibilité de construire un opérateur de Laplace-Beltrami sur des surfaces.

0.3 Discrétisations de l'opérateur de Laplace-Beltrami

Puisque l'on veut employer l'opérateur de Laplace-Beltrami sur des surfaces discrètes, il faut pouvoir construire une discrétisation de cette opérateur.

Lorque l'on manipule une fonction définie sur une surface, on suppose qu'elle est échantillonnée sur plusieurs points de la surface (souvent correspondant aux sommets de la surface). On place toutes les valeurs prises par la fonction dans un

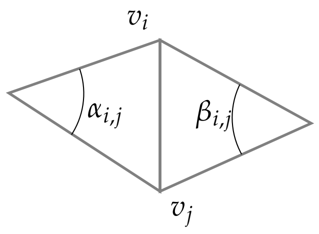


FIGURE 1: Angles utilisés dans la formule cotangente

vecteur **u**, et la construction du Laplacien revient ensuite à déterminer une matrice $\mathbb{L}$ telle que $\mathbb{L}\mathbf{u}$ représente le Laplacien de la fonction **u**. En général, on cherche $\mathbb{L}$ de la forme $\mathbb{L} = M^{-1}L$, où M est appelée *mass matrix* et L *stiffness matrix*. Afin de discrétiser l'opérateur, il existe plusieurs schéma de discrétisation d'EDP que l'on peut employer. On peut notamment citer les méthodes des éléments finis [Red19; DE13; EG04], les méthodes des volumes finis [DUS55; EGH00] et le calcul exterieur discret [Hir03; Des+05]. Dans le cas des maillages triangulaires, on obtient généralement la formule cotangente: les coefficients non diagonaux de la matrice L sont donnés par la formule (en reprenant les coefficients de la figure 1)

$$L_{i,j} = -\frac{\cot\alpha_{i,j} + \cot\beta_{i,j}}{2},$$

et les coefficients digonaux sont l'inverse de la somme des autres coefficients de la ligne.

Le cas des maillages polygonaux est plus complexe, et chaque schéma donne des résultats différents, avec avantages et inconvénients. Il existe également d'autres méthodes, comme les méthodes à raffinement virtuel [Bun+20; Bun+24], où des points sont rajoutés à l'intérieur de chaque polygones pour permettre une triangulation (où l'on peut ensuite calculer l'opérateur à l'aide des méthodes existantes pour surfaces triangulaires). On construit ensuite un opérateur de projection qui permet de passer de l'opérateur construit sur la surface triangulée, avec les sommets rajoutés à la surface originale sans ces sommets.

0.3.1 Propriétés recherchées

Il y a plusieurs propriétés que l'on souhaite avoir sur notre opérateur, en particulier sur la matrice L. On veut que cette matrice soit creuse (et occupe ainsi moins de mémoire, et soit plus facile à inverser), et de la forme suivante:

$$(Lx)_i = \sum_{j \in V(i)} \omega_{i,j}(x_i - x_j),$$

où $V(i)$ est le voisinage du sommet i et $\omega_{i,j}$ sont les coefficients recherchés pour L. Les propriétés recherchées sont données par Wardetzky et al. [War+07], avec un résultat "no free lunch": il n'est pas possible de trouver un opérateur satisfaisant toutes ces proriétés pour toutes surfaces. Ces propriétés sont les suivantes:

- symmétrie: $\omega_{i,j} = \omega_{j,i}$
- localité: $\omega_{i,j} \neq 0 \Rightarrow v_i$ and v_j sont voisins
- précision linéaire: si le domaine est contenu dans le plan, et la fonction u définie sur le domaine est linéaire, alors $Lu = 0$ aux sommets intérieurs

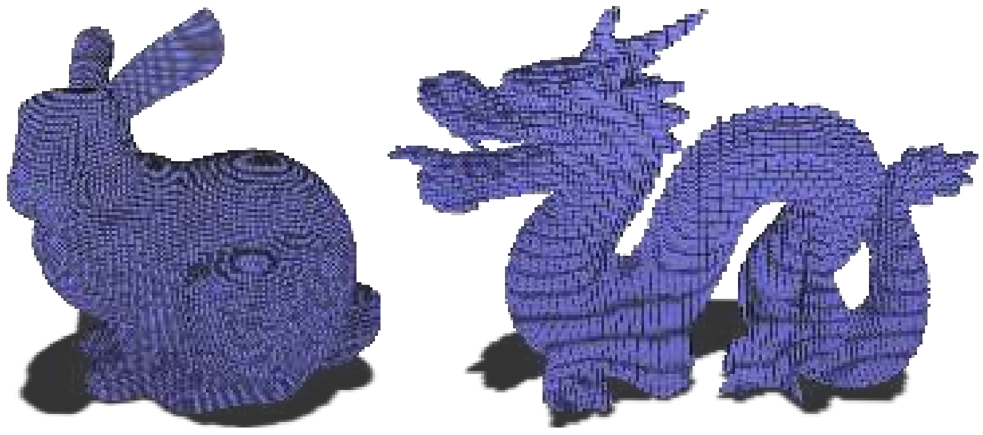


FIGURE 2: Surfaces digitales des meshs bunny et dragon

- poids positifs: $i \neq j \Rightarrow \omega_{i,j} \geq 0$, ce qui garanti le principe du maximum (les fonctions harmoniques n'ont pas d'extremas locaux aux points intérieurs)
- semi définie positive: L doit être SDP
- convergence: lorsqu'une surface lisse S est approximée par une suite de meshs M_i qui convergence (avec une norme appropriée) vers S, les solutions discrètes aux problèmes de dirichlet données par chaque $\mathbb{L}_i$ convergent vers la solution obtenue sur le domaine lisse.

Par exemple, la formule cotangente ne respecte pas forcément les poids positifs. Une variante, basée sur les triangulations intrinsèques [BS07], permet d'obtenir cette propriété mais l'obtient au coût du principe de localité.

0.3.2 Géométrie digitale

Notre objectif ici est de parvenir à construire des opérateurs de Laplace-Beltrami pour surfaces digitales. Ce sont les surfaces que l'on obtient en géométrie digitale, c'est à dire la géométrie construite par des objets qui sont des sous ensembles de $\mathbb{Z}^n$. On s'intéresse dans notre cas à $n = 3$, et à des surfaces. Un exemple connu de géométrie digitale sont les pixels, qui constituent des éléments de $\mathbb{Z}^2$: on étudie ici plutôt des voxels, et plus particulièrement la surface définie par un objet construit en voxels (figure 2).

La construction d'opérateurs de Laplace-Beltrami pour ces surfaces n'est pas évidente. On peut en effet s'intéresser à un cas plus simple pour s'en rendre compte: la convergence d'un estimateur de surface. Simplement prendre la somme des aires des éléments consituants la surface converge dans le cas des surfaces triangulaires (hors cas pathologiques), mais à l'inverse ne converge pas dans les surfaces digitales (figure 3). On peut cependant utiliser les normales à la surfaces afin de corriger cet estimateur et d'obtenir un estimateur convergent. Des estimateurs convergents de normale et de courbure pour ces objets existent déjà, comme la méthode Integral Invariant [LCL17].

On observe le même problème avec la construction de l'opérateur de Laplace-Beltrami: en applicant naïvement une construction d'opérateurs polygonaux, les opérateurs que l'on obtient ne semblement pas convergents. On cherche donc à développer des opérateurs convergents, et pour cela nous allons utiliser une correction sur les normales similaire à celle employée pour corriger l'estimateur d'aire.

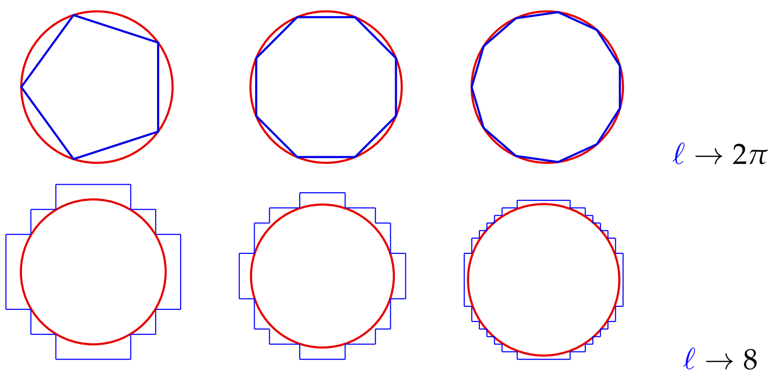


FIGURE 3: Haut: le perimètres de polygones réguliers inscrits dans un cercle concerge vers le perimètre du cercle. Bas: la discrétisation de Gauss d'un cercle de rayon 1 a un périmètre constant de 8 (à une différence de h près selon la discrétisation), et ne converge donc pas vers le résultat recherché.

0.4 Opérateurs de Laplace-Beltrami corrigé par normales

Nous présentons ici trois adaptations de méthodes existantes pour construction d'opérateurs sur des surfaces polygonales à des surfaces digitale, via une correction basée sur les normales. Il existe déjà des méthodes pour obtenir ces opérateurs sur les surfaces digitales. L'une d'entre elle (par Caissard et al.[Cai+19]) a des garanties de convergences fortes, mais présente des inconvénients: elle demande d'estimer la distance géodésique entre toutes les paires de points de la surface (ce qui est couteux), et produit une matrice dense qui occupe donc plus de mémoire et est plus lourde à inverser. Une autre méthode (Coeurjolly et al. [CL22]) basée sur le calcul exterieur discret produit elle des opérateurs creux, mais n'a pas de garanties de convergences.

Nos méthodes se basent sur des notions d'espace tangents corrigés. Grâce à un vecteur normal $\mathbf{u}$ qui ne correspond pas forcément à la normale obtenue naïvement à partir de la surface digitale, on peut par exemple corriger la longueur d'un vecteur $\mathbf{v}$ en utilisant non pas la norme de $\mathbf{v}$ mais $||\mathbf{v} \times \mathbf{u}||$. De la même manière, on peut corriger l'aire d'un parallélogramme défini par les vecteurs $\mathbf{v}$ et $\mathbf{w}$ en prenant $\det(\mathbf{u}, \mathbf{v}, \mathbf{w})$.

Calcul interpolé corrigé

Cette méthode reprend l'esprit de la méthode de Coeurjolly et al. [CL22], en essayant cependant de placer les normales aux arêtes plutôt que sur les faces. Ces normales sont ensuite bilinéairement interpolées lors des calculs faits sur les surfels, ce qui nous permet de considérer le champ de normales comme continu sur toute la surface. Cela ne respecte pas le fait que les normales devraient être des vecteurs unitaires, mais donne des formules plus simples pour les calculs. De plus, des experiences avec une interpolation plus complexe donnant des vecteurs moins loin de vecteurs unitaires ne donne pas de meilleurs résultats.

Ces normales interpolées sont ensuite utilisées afin de réaliser les calculs dans la Grassmanienne, dont on dérive des endomorphismes musicaux, des produits intérieurs et finalement un Laplacien.

Éléments finis corrigés

On adopte ici une approche éléments finis et on cherche à résoudre le problème de Poisson (sous sa forme faible) suivant: pour f donné, trouver g tel que pour tout Φ

$$\int_\Omega \nabla g . \nabla \Phi = - \int_\Omega f\Phi + \int_{\partial\Omega} \Phi \langle \nabla g, \mathbf{n} \rangle \,, \tag{1}$$

Nous faisons le choix de prendre Φ comme les fonctions localements bilinéaires à l'intérieur de chaque élément (les surfels). On se retrouve avec une base de fonctions Φ_i où Φ_i est la fonction bilinéaire à l'intérieur de chaque surfel valant 1 au sommet i et 0 aux autres. On peut alors montrer que cela revient à trouver $\mathbf{g}$ tel que $L\mathbf{g} = M\mathbf{f}$, et où les coefficients des matrices sont obtenus via les formules suivantes: $L_{i,j} = \int_\Omega \langle \nabla\Phi_i, \nabla\Phi_j \rangle_G$ et $M_{i,j} = \int_\Omega \langle \Phi_i, \Phi_j \rangle$.

L'aspect corrigé par les normales intervient lors du calcul de la métrique G utilisée dans les produits scalaires et les formules de gradient. On choisit ici des normales constantes par face (des normales interpolées rendant les calculs d'intégrales trop difficiles).

On obtient ainsi une autre version de la matrice de stiffness L et la matrice de masse M.

Éléments virtuels raffinés corrigés

Cette méthode s'inspire des méthodes par éléments virtuels raffinés comme [Bun+20; Bun+24]. On cherche à suivre le procédé de [Bun+20]: d'abord, rajouter un point à l'intérieur de chaque face afin de minimiser le carré des aires de la triangulation résultante. Chaque sommet ajouté peut s'exprimer comme un barycentre des autres sommets de la face, et les poids de ce barycentre sont utilisés afin de créer une matrice P qui permet de passer des valeurs de la surfaces originales à la surface triangulée. Ensuite, on construit un Laplacien $L_\triangle$ corrigé sur la surface triangulée. Le Laplacien final s'obtient par $L = P^\top L_\triangle P$.

On choisit des normales situées aux arêtes: ainsi, trouver le point qui va minimiser l'aire induite par la triangulation résultante ne consiste pas juste à prendre le centre du surfel. Minimiser cette aire revient alors à optimiser un polynôme de degré 6: on estime son minimum à l'aide de quelques étapes d'une descente de gradient.

Comparaisons et résultats

L'un des premiers objectifs derrière le développement de ces trois méthodes est de montrer que l'on peut adapter des schémas existants en schémas pour les surfaces digitales. On vérifie experimentalement sur la sphère que les résultats que l'on obtient semblent bien converger lorsque $h \to 0$. C'est bien ce que l'on observe lorsque l'on résoud des problèmes de Poisson, mais ce n'est pas le cas quand on cherche à calculer le Laplacien d'une fonction: on peut cependant obtenir un résultat convergent en rajoutant un terme de diffusion dans ce cas. Nous n'avons pas obtenu de garanties théoriques de convergence, mais nous esperons pouvoir reprendre les preuves existantes sur les méthodes d'éléments finis et les adapter à notre méthode d'éléments finis corrigés.

L'un des intérêts du calcul interpolé était de supprimer les discontinuités entre surfels, en esperant voir une meilleur qualité: l'erreur obtenue est effectivement meilleure, mais toutes les méthodes se trouvent dans les mêmes ordres de grandeurs.

Application à la régularisation de surface

Puisque nous avons une estimation du Laplacien Δ, mais également des estimateurs de normales et de courbure, on essaye de retrouver les positions de la surface avant digitalisation en s'appuyant sur la formule $\Delta\mathbf{p} = -2H\mathbf{n}$ où $\mathbf{p}$ sont les positions, $\mathbf{n}$ sont les normales et H la courbure moyenne. On développe ainsi une méthode de régularisation pour surfaces digitales: on cherche à minimiser l'énergie $||\Delta\mathbf{p}' - H\mathbf{n}||$. Cela permet de corriger les erreurs à haute fréquences, les plus importantes visuellement et particulièrement présentes dans les surfaces digitales. Afin d'éviter une tros grosse déviation dans les basses fréquences, un terme d'attache aux données est rajouté, donnant l'énergie suivante:

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

Nous comparons notre méthode à une précédente méthode de régularization de surfaces digitales par Coeurjolly et al. [CLG21]. Notre méthode évite un rétrécissement dû au fait que notre énergie serait minimisée lorsque la surface rétrécit.

On peut élargir cette méthode à d'autres cas: par exemple, une surface dont les normales sont données dans une normal map. On peut également modifier la courbure moyenne au préalable, afin de rajouter des contraintes d'éditions de courbure choisies au préalable. On peut même directement modifier puis intégrer les estimateurs de courbures, notamment via le shape operator S, une matrice 3×3 liée aux normales par la relation $\Delta\mathbf{n} = S$. On obtient ainsi un autre champ de normal qui prend mieux en compte les contraintes de courbure souhaitées.

On observe que l'utilisation d'un Laplacien corrigé donne bien de meilleurs résultats que la version non modifiées. Cependant, cette correction n'est pas forcément très visible, et l'utilisation d'un Laplacien naïf peut également présenter des avantages dans cette situation: les résultats obtenus sans corrections donnent des surfaces où toutes les arêtes ont des longueurs similaires et sont plus régulières.

0.5 UV-mapping grâce à une fonctionnelle d'Ambrosio-Tortorelli

Les UV-maps sont des paramétrisations 2d de surfaces 3d auxquelles on permet des discontinuités. En d'autre terme, on cherche à prendre une surface, à la découper puis à l'aplatir. Cela permet notamment de créer ensuite des fichiers de texture, des images que l'on va vouloir plaquer sur la surface pour lui donner une certaine apparence. Il y a deux problèmes lors de la constructions de ces UV-maps: on veut d'un côté limiter la distorsion de la paramétrisation (une trop grosse distorsion résulte en des besoins de résolutions très différents entre plusieurs triangles, et rend la manipulation des images des textures plus difficile), et d'un autre limiter la longueur des coupes (qui rendent elles aussi la manipulation de l'image plus difficile).

De nombreuses approches existent pour essayer de résoudre ces deux problèmes. Cependant, ce sont généralements des approches qui séparent ces problèmes: les coupes sont calculées avant la paramétrisation. La qualité de la paramétrisation dépend cependant des coupes, et il y a donc un travail de prédiction de la qualité de paramétrisation depuis les coupes. Des travaux récents proposent une optimisation *simultanée* des deux, afin de ne pas avoir des coupes qui prédisent la paramétrisation mais qui s'y adaptent [Por+17; Li+18]. Notre objectif ici est de réaliser une telle approche en s'inspirant de la fonctionnelle d'Ambrosio-Tortorelli [AT90].

Fonctionnelle de Mumford-Shah et fonctionnelle d'Ambrosio-Tortorelli

Regardons d'abord la fonctionnelle de Mumford-Shah [MS89], souvent utilisée en traitement d'image afin de faire du débruitage ou de la ségmentation, et dont l'objectif est de permettre d'obtenir une fonction lisse par morceaux à partir d'une entrée g. Cette fonctionnelle prend une fonction u définie sur un domaine Ω, ainsi qu'un sous-ensembles de Ω noté K. Ce sous-ensemble correspond aux discontinuités qui vont être permises à la fonction u. Cette fonctionnelle pénalise ensuite l'écart entre u et une fonction g (correspondant l'entrée du problème, une image que l'on veut lisser par exemple), elle pénalise également le gradient de u (on cherche à obtenir u lisse) sur Ω à l'exception de K: ainsi, K permet des dicontinuités à u Un troisième terme vient pénaliser la longueur de K. On a également deux coefficients α et λ qui permettent d'équilibrer les différents termes. La formule de la fonctionnelle est donc:

$$\mathscr{M}\mathscr{S}(K,u) := \alpha \int_{\Omega\setminus K} |u-g|^2 + \int_{\Omega\setminus K} |\nabla u|^2 + \lambda \mathscr{H}^1(K\cap\Omega)\,,$$

Cette fonctionnelle est cependant assez difficile à optimiser. En effet, il est assez difficile de d'optimiser directement le domaine d'intégration du deuxième terme, autrement dit de voir comment le deuxième terme varie selon les variations de K. La fonctionnelle d'Ambrosio-Tortorelli [AT90] propose une autre formulation, faisant intervenir une constante additionnelle ϵ et substituant l'ensemble K par une fonction v:

$$\mathscr{A}\mathscr{T}_\epsilon(u,v) := \alpha \int_\Omega |u-g|^2 + \int_\Omega |v\nabla u|^2 + \lambda \int_\Omega \left(\epsilon|\nabla v|^2 + \frac{1}{4\epsilon}(1-v)^2\right)\,,$$

Lorsque ϵ tend vers 0, v prend des valeurs se rapprochant soit de 0 soit de 1. Les endroits où v vaut 0 correspondent aux discontinuités: il est démontré que les solutions de la fonctionnelle d'Ambrosio-Tortorelli Γ-convergent ves les solutions de la fonctionnelle de Mumford-Shah.

Cette fonctionnelle est plus simple à otpimiser: à u (respectivement à v) fixé, la fonctionnelle est quadratique convexe en v (respectivement en u): le minimum s'obtient alors en une étape d'une méthode de Newton. On peut donc alterner les étapes d'optimisations de u et v, et de régulièrement baisser ϵ, afin d'estimer un minimum de la fonctionnelle.

Nos approches

Nous proposons deux approches s'inspirant de la fonctionnelle d'Ambrosio-Tortorelli, mais basées sur des discrétisations différentes des fonctions u et v.

Une première approche consiste à placer u aux coins de chaque triangle et v aux arêtes. Puisque le premier terme de la fonctionnelle ne nous sert pas (nous n'avons pas d'entrée à laquelle la sortie doit correspondre), on le substitue par un terme mesurant la distorsion, par exemple en prenant l'énergie de Dirichlet symmétrique [SS15]. u étant discontinue aux arêtes, on remplace son gradient par une notion de distance entre les deux arêtes des deux triangles:

$$(v\nabla\mathbf{u})^2 = \beta v^2 \int_0^1 ||s(\mathbf{u}(c_{i1}) - \mathbf{u}(c_{j1})) + (1-s)(\mathbf{u}(c_{i2}) - \mathbf{u}(c_{j2}))||^2 ds.$$

On obtient alors la fonctionnelle suivante:

$$AT_{modif}(\mathbf{u}, v) = \underbrace{\alpha \sum_f A_f \Psi_q(\mathbf{u}_f) +}_{\text{distorsion term}} \quad \underbrace{\sum_e A_e((v\nabla\mathbf{u})^2}_{\text{gradient or smoothing term}} \quad + \lambda\epsilon(\nabla v)^2 + \frac{\lambda}{4\epsilon}(1-v)^2).$$

On peut reprendre la méthode d'optimisation d'AT afin de minimiser cette fonctionnelle: on alterne des optimisations à v fixé et à u fixé. Cependant, à v fixé, il faut minimiser une énergie comprenant un terme de distorsion qui n'est pas quadratique convexe: cela prend donc plus de temps qu'avant, car nous la minimisons en plusieurs étapes d'une descente quasi-Newton (voir [SGK19] pour la construction du Hessien).

Bien que cette approche semble assez naturel (v correspond directement aux coupures), elle pose deux inconvénients: les matrices obtenues prennent du temps à inverser, du fait de la taille de u et du grand nombre de valeurs non nulles dans les matrices. Également, il est difficile de faire respecter l'injectivité des UV-maps: on manipule une soupe de triangles, et il est à la fois trop couteux et trop restrictif de les empêcher de se chevaucher complêtement.

Une deuxième approche consiste donc à placer u aux sommets de la surface et v sur les faces. On perd donc la facilité de placer v directement aux endroits des coupes, mais il est cependant plus facile d'évaluer le gradient de u au même endroit que v. On peut également remarquer que le gradient de u au carré donne une première moitié de la formule de l'énergie de Dirichlet symétrique (valant $0.5 * (|\nabla u|^{-2} + |\nabla u|^2)$). Ainsi, plutôt que de transformer le premier terme d'attache aux données en un terme de distorsion et d'essayer de se débrouiller avec le second terme, on supprime simplement le premier et le second devient notre mesure de la distorsion. On peut ainsi voir les triangles comme rigides ou non, et que l'on va permettre à certains triangles de ne pas être rigides afin que les autres puissent mieux se positionner. On obtient la fonctionnelle suivante, que l'on peut optimiser de manière similaire que la précédente:

$$E(\mathbf{u}, v) := \sum_f A_f(v^2 + \gamma) \underbrace{\left(\frac{|\nabla\mathbf{u}_f|^2 + |\nabla\mathbf{u}_f|^{-2}}{2} - 1\right)}_{\Psi_f(\mathbf{u})} + A_f\lambda\left(\epsilon|\nabla v_f|^2 + \frac{1}{4\epsilon}(1-v_f)^2\right)$$

$$= \sum_f A_f\left((v^2+\gamma)\Psi_f(\mathbf{u}) - \lambda\epsilon v_f(\Delta v_f) + \frac{\lambda}{4\epsilon}(1-v_f)^2\right).$$

Il faut cependant un moyen de passer de v à des coupes. Il faut faire en sorte que cette coupe suive les triangles qui ont été étirés: on chosisit donc de considérer chaque composante connexe de triangles où $v < 0.5$, puis, dans chaque composantes connexese, on prend le plus court chemin reliant les deux points les plus espacés. Cela ne permet pas toujours de découper partout où il l'aurait fallu (par exemple lorsque les triangles à découper forment une étoile, on ne découperait ici que deux branches de l'étoile), mais c'est une procédure rapide et généralement suffisante (il suffit au pire de la répéter).

Cette deuxième méthode présente des avantages aux niveau de la vitesse d'execution: on parvient à une méthode similaire en terme de réduction de la distorsion et de longueur de coupure pour une durée de calcul généralement réduite par rapport à la méthode OptCuts (tableau 1).

Model	bunnyhead	camelhead	bimba	hand	cat	armadillo	duck1	planck1	venus	tooth	circular box
OptCut's time	2.5s	89s	13s	0.60s	0.65s	29s	2.08s	**1.19s**	0.61s	25.4s	22s
Our time	**1.6s**	**38s**	**6s**	**0.13s**	**0.11s**	**6.1s**	**0.77s**	1.32s	**0.36s**	**2.1s**	**3.8s**
Initial distortion	9.10	7.45	7.64	10.04	6.67	10.3	5.59	9.03	8.78	4.26	5.28
OptCut's distortion	**4.36**	4.26	4.50	5.2	4.38	5.22	4.49	4.54	4.37	4.24	4.53
Our distortion	4.39	4.26	4.50	5.3	4.50	5.22	4.49	4.56	4.39	4.25	4.54
OptCut's cut length	**2.88**	2.93	2.01	**4.5**	**1.28**	**2.07**	**1.09**	**1.61**	**1.70**	0.24	**1.13**
Our cut length	3.82	**2.56**	**1.90**	5.6	1.76	4.25	1.31	2.33	2.94	**0.10**	1.50

TABLE 1: Comparaison quantitative entre la méthode OptiCuts et la notre en terme de durée de calcul, de distorsion obtenue et de longueur de coupure. Les meshes `bimba`, `hand`, `cat`, `armadillo`, `duck`, `planck`, `venus`, `tooth` and `circular box` proviennent des entrées disponible de le dépôt d'OptCuts [Li+18].

Elle permet aussi assez facilement d'intégrer les contraintes d'injectivité [SS15], mais aussi des contraintes comme vouloir encourager les coupes dans les endroits peu visibles (en faisant varier le coefficient λ selon un terme d'occlusion ambiante par exemple), où bien à des contraintes de packing où l'on va obliger la paramétrisation à tenir dans un certain rectangle.

0.6 Conclusion

Nous avons introduit plusieurs méthodes de construction de l'opérateur de Laplace-Beltrami sur les surfaces digitales, et les experiences suggèrent que ces opérateurs sont bien convergents: nous n'avons cependant pas fourni de garanties théoriques sur la convergence, même si nous esperons qu'il est possible d'en trouver à partir des preuves existantes de convergences en éléments finis. Il est également possible d'examiner des discrétisations avec d'autres éléments, comme des éléments avec valeurs aux arêtes ou bien des éléments d'ordre supérieurs. L'utilisation de ces opérateurs dans le cadre d'autres géométries (comme avec les normal maps) peut également être approfondit.

Nous avons également introduit deux méthodes adaptées de la fonctionnelle d'Ambrosio-Tortorelli afin de construire des UV-maps, mettant en avant l'importance du choix de discrétisation. Ces méthodes bénéficieraient de solver plus rapides pour la distorsion, et il serait intéressant d'étudier l'utilisation de Progressive Parameterizations [Liu+18]. Il serait également intéressant d'étudier d'autres méthodes de coupe, et d'envisager des moyens de revenir sur des coupes précédemment réalisées dans la deuxième méthode.

Introduction

The Laplace-Beltrami operator, used in various PDEs such as the heat equation or the wave equation, has been discretized very early using electrical networks in order to approximate solutions of these PDEs. This thesis focuses on its usages in geometry processing, where we use mathematical tools and concepts in order to manipulate 3D discrete objects such as surface meshes, volume meshes or point clouds. A problem representative of those found in this domain is the reconstruction of scanned points into a triangle mesh. For example, the points can be grouped into triangles with methods such as plane fitting, and the noise of the scanned data can be smoothed out using tools similar to Fourier analysis. In this field, the Laplace-Beltrami operator rests among the most commonly used mathematical objects, with applications in a wide array of methods such as mesh smoothing, mesh compression, geodesics approximation, shape description and shape matching.

Since we work on discrete objects (and in the case of this thesis, on discrete surfaces specifically), we need a discrete version of the Laplace-Beltrami operator. The process of discretizing the operator, in the form of a matrix, shows several challenges such as being symmetric and respecting the maximum principle (usually by having only positive non diagonal coefficients). Among these challenges is the convergence of the operator: ideally, the operator correctly approximates its smooth equivalent: as the surface resolution gets higher, the closer the results obtained using the discrete operator should be to the expected results. In the case of the most commonly found discretization for triangle meshes, the cotan Laplacian, it is indeed true in a weak sense on most meshes excepted for some surfaces such as the Schwarz lantern, where the normals do not converge to the one of the underlying surface.

Our work focus on the case of digital surfaces. These surfaces arise in 3D image analysis, where we work on subsets of $\mathbb{Z}^3$. While in the case of triangle meshes non converging normals represent a pathological case, here on the contrary normals can safely be assumed not to converge. As a result, even for simple estimators such as for the length and the area, achieving convergence is not easy, but not impossible: converging estimators for these quantities do exist, as well as for normals and curvatures.

Our goal is to make use of these estimators in order to develop calculus methods for digital surfaces that behave as close as possible to their smooth counterparts. Such methods have already been developped, and even proven to converge for the Laplace-Beltrami operator: they are however slow to build (requiring the geodesic distance between any two points of the surface), and produce dense operators thus consuming a lot of memory and being slow to invert. We therefore aim at developping sparse operators through local constructions which can, as opposed to the previous dense operators, be as quickly built and easily inverted as the standard method used for triangle surfaces (e.g. cotan Laplacian).

Our method consists in equipping our calculus with a corrected normal field. We show several ways to build a digital calculus. We propose adaptations of three existing methods on geometry where an additional normal field is provided: first, and adaptation of discrete exterior calculus where the corrected normal field is used in order to redefine the product between k-forms. A previous method based on discrete exterior calculus has already been proposed: this one however, uses corrected normal placed at vertices which are then interpolated inside each face, where the previous method used normals placed on faces and considered constant inside each face. A second one is an adaptation of the finite element method where the corrected normal field (here constant per face) is used in order to define a metric used in our calculations. The third one is an adaptation of the virtual refinement method, with corrected normals placed at vertices, where each face is split into triangles using a virtual vertex which minimizes the sum of the squared areas (corrected using the normals) of the created triangles.

We experimentally verify that these methods behave as expected, notably in term of convergence when solving a Poisson problem or when computing the Laplacian of a function. We also use these methods in order to build a Laplace-Beltrami operator which we then use along with normal and curvature estimators to develop a regularization method for digital surfaces. We then extend this regularization method to triangle surfaces with normal maps and to include curvature edition constraints. The corrected schemes described in this chapter were presented at the conference DGMM2024, and the addition of the virtual refinement method and applications to regularization are currently in submission for a JMIV special issue.

We also study adaptations of the Ambrosio-Tortorelli functional in order to produce UV maps. During the discretization of this functional, we will again have to choose different discretizations of differential operators (include the Laplacian).

UV maps are 2d flattening of surfaces, mainly used for applying textures (stored as images) onto the surface, with some discontinuities called "seams" or "cuts". When building UV maps, the goal is to minimize the distorsion of the parameterization while minimizing the seams length. Minimizing the distorsion avoids having triangles mapped to the plane at two different scales and thus requiring different resolutions, and helps modifications made onto the texture be similar to the resulting modification onto the surface. We also want to avoid having too much seam length: in the same way that a map of the Earth can be cut in order to have less distorsion and still be harder to manipulate when it has too much discontinuities, the quality of a UV map is reduced when too much seams are present. Optimizing our adapted Ambrosio-Tortorelli functional results in a simultaneous optimization of the distorsion and of the seams of the parameterization. We observe that different discretizations result in different trade offs, in particular on speed and injectivity. These works around UV mapping were presented at SMI2023.

The following thesis is laid out as follows:

- **Chapter 1: The Laplace-Beltrami operator.** We introduce some notions of exterior calculus, allowing us to define key concepts of differential geometry and to properly define a Laplace-Beltrami operator on surfaces. We also explore a few of the properties of this operator. We then list some of the many uses in geometry processing, such as for mesh smoothing, mesh compression, geodesics approximation, shape description and shape matching.

- **Chapter 2: Discretizing the Laplace-Beltrami operator.** Now that we have introduced the definition of the Laplace-Beltrami operator and its need in geometry processing, we examine the different ways to discretize it: first on triangle meshes, where the various scheme such as the Finite Element Method, the Finite Volume Method or Discrete Exterior Calculus all result in the widely used cotan formula. Then, we look at the more complex case of general polygonal meshes, where these different schemes result in different discretizations of the operator. Finally, we look at some of the shortcomings, in particular in the case of digital surfaces: these surfaces arise in digital geometry (geometry in $\mathbb{Z}^3$) and correspond to the surface defined by voxels. We see that using classical polygonal methods on these surfaces do not yield operators that seem to converge to the expected operator for the corresponding shape.

- **Chapter 3: Corrected Laplacians.** Having seen that standard methods for building discrete Laplace-Beltrami operators do not yield good enough results in digital geometry, we introduce the use of *corrected* operators: we use normal estimators in order to build a field of corrected normals on our surface, and we use these normals as an indicator of the true tangent field of the surface. We then explore various schemes for building Laplace-Beltrami operators which we adapt to include these new tangent fields. We then test the application of these operators on simple known case in order to see if the correction does seem to work: while in some cases the corrected operator is convergent (while the non corrected one is not), in others it requires a diffusion term to converge. We thus conclude that these corrected operators are promising for digital surfaces.

- **Chapter 4: Some applications of corrected Laplacians.** Now that we have developed corrected operators for digital surfaces, we look at how they can be applied. Our application is the reconstruction of the original surface (surface regularization) by use of normal and mean curvature estimators along with our corrected Laplace-Beltrami: we introduce a regularization method for digital surfaces on this base. We then further explore this method: instead of estimated normals on digital surface, we use normal maps on triangle meshes, aiming to recover the high resolution surface that was used to build the normal map. In these applications, the corrected operator results in more accurate results, however the non-corrected ones result in a geometry where almost all edges have the same length, which may be preferrable if a regular geometry is wanted. We also explore possibilities of curvature modification before the regularization.

- **Chapter 5: UV-mapping using an Ambrosio-Tortorelli functional.** We tackle here the problem of UV-mapping. The problem consists in finding a 2d parameterization of a surface mesh, preferrably without injective and reducing some distorsion metric. During a standard UV-mapping procedure, the surface is first cut and then each patch is flattened while reducing some distorsion. Our goal is to provide a method where both of these taks are done at the same task: in order to do that, we rely on the Ambrosio-Tortorelli functional, which allows us to formulate variations of this problem as a variational problem. We explore two different ways of discretizing this functional.

- **Conclusion.** We summarize our contributions listed in this thesis, and discuss a few of the perspectives for future works. Notably, further works around corrected operator could explore theoretical proof of convergence, or higher order methods for use on multi scales surfaces. UV generation could also be improved with faster distorsion optimization methods and smarter strategies for cuts.

CHAPTER

The Laplace-Beltrami operator 1

Résumé

Dans ce premier chapitre, nous rappelons d'abord quelques éléments de définitions du calcul exterieur afin de pouvoir définir l'opérateur de Laplace-Beltrami via la formule $\Delta = \text{div} \circ \text{grad}$, ou l'opérateur de Laplace-de Rahm $\Delta = d\delta + \delta d$.

Cet opérateur intervient dans des équations connues telle que l'équation de la chaleur $-\Delta T = \frac{\partial T}{\partial t}$ (l'opérateur charactérise ainsi la diffusion) et l'équation des ondes $\Delta f = \frac{\partial^2 f}{\partial t^2}$. Le spectre de cet opérateur correspond également aux bases de Fourier dans des espaces Euclidiens, il est donc utile pour généraliser les outils de l'analyse de Fourier à des espaces non Euclidiens comme des surfaces.

Nous nous intéressons également à l'utilisation de cet opérateur en traitement de géométrie, c'est à dire dans des problèmes autour des formes 3d utilisées en informatiques, comme la reconstruction de surface ou le débruitage de surface.

Nous nous concentrerons ici aux applications liées aux surfaces. L'opérateur de Laplace-Beltrami apparaît fréquemment dans ce domaine: cela s'explique notamment par son lien avec l'énergie de Dirichlet. Cette énergie sert à mesurer, pour une fonction u, à quel point cette fonction est lisse et vaut $\int |\nabla u|^2$. Les minimiseurs de cette énergies correspondent, à un terme de bord prêt, aux fonctions harmoniques c'est à dire de Laplacien nul. Nous listons dans ce chapitre des applications de cet opérateur dans les problèmes suivants:

- Lissage de surface
- Compression de surface
- Approximation de géodésiques
- Signature de points, description de surface
- Correspondance de surface

Ces méthodes se basent donc toutes sur la possibilité de construire un opérateur de Laplace-Beltrami sur des surfaces.

First, we provide a few intuitions on the concepts used in exterior calculus, which we then use to define the Laplace-Beltrami operator. We then give our motivation for studying this operator: it has been widely used in geometry processing, and we list some of the tasks it has been used in.

1.1 Definition and properties

1.1.1 Exterior Calculus

Our goal is to present a brief introduction to several concepts used when defining integrals and differential operators on surfaces (and more generally on smooth manifolds). In section 1.1.2, we will introduce the Laplace-Beltrami operator defined using these concepts. Keeping these definitions in mind will be useful later, when we will try to build operators on surface meshes that mimic the behaviour of their smooth equivalent. We do not provide rigorous definition (which would take a few chapters by itself), but only aim at giving an intuition of what the presented objects represent and how they relate to each other. For a more rigorous presentation of these concepts, see Lee J. [Lee12].

Exterior algebra Suppose we have a vector space E of dimension n, with a basis $(\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n)$ and an inner product $\langle ., . \rangle$. While vectors give us a way to represent a small portion of length, we do not directly have a way to represent higher dimensional parts of space such as signed areas or signed volumes. These higher dimensions objects can however be created by combining multiple vectors together, yielding a *k-vector*, thanks to an operation called the wedge product and noted $\wedge$. We then build a linear space called an *exterior algebra* or *Grassmann algebra* using our vectors and our newly defined k-vectors.

The wedge product takes several vector and assign the space "spanned" by these vectors. This span could be defined as tuple of vectors: the span of two vectors $\mathbf{u}$ and $\mathbf{v}$ could be defined as $(\mathbf{u}, \mathbf{v})$. However, in order to make sense as (signed) areas, we add several rules: it has to be linear in each input, and changing the order of the vector results in the same area with opposite sign. Thus, the wedge product is defined as an alternating operation between vectors: $\mathbf{u} \wedge \mathbf{v} = -\mathbf{v} \wedge \mathbf{u}$, and in particular $\mathbf{u} \wedge \mathbf{u} = 0$.

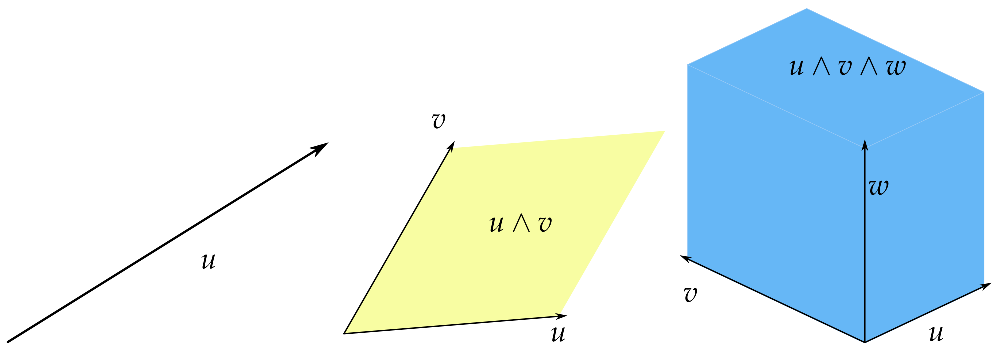


FIGURE 1.1: A vector represent a portion of length, a bivector represent the portion of signed area defined by its two vectors, and a 3-vector represents the volume defined by its three vectors.

Let us see an example on two 2d vectors in cartesian representation: $\mathbf{u} = a\mathbf{e}_1 + b\mathbf{e}_2$ and $\mathbf{v} = c\mathbf{e}_1 + d\mathbf{e}_2$. Using the properties given earlier, we have:

$$\begin{aligned}
\mathbf{u} \wedge \mathbf{v} &= (a\mathbf{e}_1 + b\mathbf{e}_2) \wedge (c\mathbf{e}_1 + d\mathbf{e}_2) \\
&= a\mathbf{e}_1 \wedge (c\mathbf{e}_1 + d\mathbf{e}_2) + b\mathbf{e}_2 \wedge (c\mathbf{e}_1 + d\mathbf{e}_2) \\
&= ac\mathbf{e}_1 \wedge \mathbf{e}_1 + ad\mathbf{e}_1 \wedge \mathbf{e}_2 + bc\mathbf{e}_2 \wedge \mathbf{e}_1 + bd\mathbf{e}_2 \wedge \mathbf{e}_2 \\
&= ad\mathbf{e}_1 \wedge \mathbf{e}_2 + bc\mathbf{e}_2 \wedge \mathbf{e}_1 \\
&= (ad - bc)\mathbf{e}_1 \wedge \mathbf{e}_2
\end{aligned}$$

This object is called called a 2-vector or bivector, and represents the parallelogram defined by these two vectors (figure 1.1). The magnitude of this bivector corresponds to the 2d determinant of $\mathbf{u}$ and $\mathbf{v}$, which gives the signed area of the parallelogram they define.

The wedge product can also be used on k vectors: the result is then called a k-vector. This wedge product can then be calculated between k-vectors by decomposing them: for example, the product between the vector $\mathbf{u}$ and the bivector $\mathbf{v} \wedge \mathbf{w}$ is simply given by $\mathbf{u} \wedge \mathbf{v} \wedge \mathbf{w}$.

The space spanned by vectors and k-vectors is called an *exterior algebra* or *Grassmann algebra*, and is denoted $\bigwedge(E)$. We focus on the linear subspaces $\bigwedge^k(E)$ of $\bigwedge(E)$ spanned by the k-vectors. In particular, we have $\bigwedge^0(E) = \mathbb{R}$ and $\bigwedge^1(E) = E$ We can derive the basis of each of theses space of $k-$vectors (and thus the dimension of each) from the basis of our vector space: the basis of $\bigwedge^k E$ is $\{\mathbf{e}_{i_1} \wedge \mathbf{e}_{i_2} \wedge ... \wedge \mathbf{e}_{i_k} | 1 \leq i_1 < i_2 < ... < i_k \leq n\}$, and its dimension is $\binom{n}{k}$.

We can also notice that $\bigwedge^n E$ is of dimension 1, meaning that every element is of the form $f(x)\mathbf{e}_1 \wedge \mathbf{e}_2 \wedge ... \wedge \mathbf{e}_n$, for $x = (\mathbf{x}_1, ..., \mathbf{x}_n) \in E^n$. From the definition of the wedge product, it follows that f must be an alternate multilinear form, and $f(\mathbf{e}_1, \mathbf{e}_2, ...\mathbf{e}_n) = 1$. These correspond to the definition of the determinant, so $f = \det$. This makes clear the relationship between the volume defined by the tuple of vectors x and the corresponding k-vector: the determinant of x defines the magnitude of $\mathbf{x}_1 \wedge ... \wedge \mathbf{x}_n$.

When $n = 3$, we have 1-vectors (or simply vectors) to describe lengths, 2-vectors for areas and 3-vectors for volumes. These k-vectors will serve in order to define "infinitesimal" pieces of length, area or volume to integrate on: a curve for example is an assembly of infinitesimal pieces of length and an infinitesimal piece of surface is a 2-vector.

Covectors and k-forms Since we have spaces of vectors and k-vectors, we can define covectors and k-form as their dual. The dual of E, noted E^*, is the space of applications $E \to \mathbb{R}$. This space is isomorphic to E: it has the same dimension n, and a basis for E^* can be naturally derived from a basis of E. If we denote $\mathbf{e}^i$ the covector verifying:

$$\mathbf{e}^i(\mathbf{e}_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j, \end{cases}$$

then this basis is $\mathbf{e}^1, \mathbf{e}^2, ..., \mathbf{e}^n$.

We can do the same with k-vectors as well, i.e. studying the dual of $\bigwedge^k E$. We can use the wedge product on the element of E^* in order to define $(\bigwedge^k E)^*$: we have the same properties as before, so if α and β are elements of E^*, $\alpha \wedge \beta = -\beta \wedge \alpha$ and $\alpha \wedge \alpha = 0$. In the same way $\{\mathbf{e}_{i_1} \wedge \mathbf{e}_{i_2} \wedge ... \wedge \mathbf{e}_{i_k} | 1 \leq i_1 < i_2 < ... < i_k \leq n\}$ form a basis of $\bigwedge^k E$, $\{\mathbf{e}^{i_1} \wedge \mathbf{e}^{i_2} \wedge ... \wedge \mathbf{e}^{i_k} | 1 \leq i_1 < i_2 < ... < i_k \leq n\}$ form a basis of $\bigwedge^k(E^*)$. These spaces are naturally isomorphic to $(\bigwedge^k E)^*$, and in practice we consider these two spaces the same.

The relationship we saw between n-vectors and the determinant becomes clearer here. We know that $\bigwedge^n(E^*)$ is a vector space of dimension one, and since $(\mathbf{e}^1 \wedge \mathbf{e}^2 \wedge ... \wedge \mathbf{e}^n)(\mathbf{e}_1 \wedge \mathbf{e}_2 \wedge ... \wedge \mathbf{e}_n) = 1$, all elements of this vector space are proportional to det. det thus form a basis of $\bigwedge^n(E^*)$. While det gives the Euclidean volume, we sometime use other *volume forms* ω instead. A volume form is an element of $\bigwedge^n(E^*)$, and gives the idea of what a unit volume is in this space, and can vary depending on space (if attached to a density for example).

Another example of covector is the differential of a function f at a point a, noted df_a, which verifies:

$$\forall h \in E, f(a+h) = f(x) + df_a(h) + o(h)$$

Thanks to $k-$forms, we have an object that will "measure" $k-$vectors. In the context of integration, we can think of these as infinitesimal elements of integrand, that will be evaluated on inifitesimal elements of space ($k-$vectors).

Musical isomorphisms In finite dimension, these k-vectors can be seen as measuring how well a vector aligns to a certain vector. In fact, we can reformulate this thanks to the dot product and the isomorphism between the dual space and the primal, which gave our basis $\{\mathbf{e}^i\}$ for the dual from the primal basis $\{\mathbf{e}_i\}$. For a covector $\alpha = \alpha_1\mathbf{e}^1 + ... + \alpha_n\mathbf{e}^n$ and a vector $u = u^1\mathbf{e}_1 + ... + u^n\mathbf{e}_n$, n,

$$\begin{aligned}\alpha(u) &= u^1\alpha(\mathbf{e}_1) + ... + u^n\alpha(\mathbf{e}_n)\\ &= u^1\alpha_1\mathbf{e}^1(\mathbf{e}_1) + ... + u^n\alpha_n\mathbf{e}^n(\mathbf{e}_n)\\ &= u^1\alpha_1 + ... + u^n\alpha_n,\end{aligned}$$

meaning applying α to a vector u is the same as taking the dot product of u and the vector $\alpha_1\mathbf{e}_1 + ... + \alpha_n\mathbf{e}_n$. In a more general setting, the dot product is replaced by the inner product $\langle .,. \rangle$, and this relationship between the dual and the primal is given by musical isomorphisms.

There are two musical isomorphisms: the flat operator $\flat : E \to E^*$, defined as $\mathbf{u}^\flat = \langle \mathbf{u}, . \rangle$ and the sharp operator $\sharp : E^* \to E$ defined as the inverse of the flat: $\alpha = \langle \alpha^\sharp, . \rangle$.. These operators can then be extended to k-vectors and k-forms by applying the operator to each component individually.

The sharp operator is used to properly define the gradient of a function. The gradient of f at point a is the vector ∇f_a such that $df_a(x) = \langle \nabla f_a, x \rangle$. This corresponds to the sharp of the differential of f at point a.

Hodge star operator The concept of 2-vectors in 3d may have shown some similarities with the one of cross product between two vectors (noted $\mathbf{u} \times \mathbf{v}$): they both have the same norm but they are orthogonal. Actually, $\det(\mathbf{u}, \mathbf{v}, (\mathbf{u} \times \mathbf{v})) = ||\mathbf{u} \times \mathbf{v}||^2 = ||\mathbf{u} \wedge \mathbf{v}||^2$: the cross product is the orthogonal to the bivector such that their wedge product gives a volume whose magnitude is the squared of their norm. In some sense, it maps to a bivector the vector that complements it in order to have a full volume. We can define an operator that realizes this mapping from 2-vectors to 1-vectors called the Hodge star operator, noted $\star$. Since $\bigwedge^2(\mathbb{R}^3)$ and $\bigwedge^1(\mathbb{R}^3)$ are both of dimension one, and the kernel of this linear operator is of dimension 0, it is an isomorphism: its inverse is also called the Hodge star operator from $\bigwedge^1(\mathbb{R}^3)$ to $\bigwedge^2(\mathbb{R}^3)$. We can extend this as a linear mapping between k-vectors and $(n-k)$-vectors: the Hodge star operator gives the orthogonal $(n-k)$-vector to a k-vector such as the determinant of their wedge product is the squared norm of the k-vector. For $\mathbf{u} \in \bigwedge^k(E)$,

$$\det(\mathbf{u} \wedge \star\mathbf{u}) = \langle \mathbf{u}, \mathbf{u} \rangle.$$

We actually require this definition to hold for any pair of k-vectors (the hodge dual is not uniquely defined otherwise):

$$\det(\mathbf{u} \wedge \star\mathbf{v}) = \langle \mathbf{u}, \mathbf{v} \rangle.$$

The hodge star operator can also be defined for k-forms. First, we need to define the inner product on k-forms (since it was only defined for k-vectors until now). A way to define this inner product is through the musical operator: we can send 1-forms to vector, and take the value of the inner product of the resulting vector as the value of the inner product between the covectors $\langle \alpha, \beta \rangle = \langle \alpha^\flat, \beta^\flat \rangle$. If the inner product is represented by an inner metric G, meaning $\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^\top G\mathbf{v}$, then the inner product between two covectors α, β is $\alpha^\top G^{-1}\beta$. Now that we have an inner product (and a norm) between k-forms, we can also define the Hodge star: $\star$ maps k-forms to $(n-k)$-forms, and for any α, β $k-$forms,

$$\alpha \wedge \star\beta = \langle \alpha, \beta \rangle \omega,$$

where ω is the volume form.

Smooth manifolds, surfaces Until now we have mostly used $\mathbb{R}^n$ in order to illustrate our concepts. However, our goal is to be able to use these notions on *smooth manifolds*, and especially on surfaces. Intuitively, a surface is a subset of $\mathbb{R}^3$ which is locally similar to $\mathbb{R}^2$. This notion is more rigorously defined thanks to the definition of an atlas. We consider a subset $M \in \mathbb{R}^n$ as our domain.

An atlas is a collection of open sets $U_i \subset M$ and a collection of charts (U_i, ϕ_i) that maps these sets to a portion of $\mathbb{R}^m$ (with $m \leq n$), where $\phi_i : U_i \to \mathbb{R}^m$ are homeomorphisms (figure 1.2) They may overlap ($U_i \bigcap U_j \neq \emptyset$), in which case the composition $\phi_j \circ \phi_i^{-1}$ is called a transition map and has to also be an homeomorphism. Thanks to this atlas, we can have local coordinates $(x_1, ..., x_n)$ by taking the components of $\phi(p)$ for any point p contained in U_i. We also want to be able to differentiate functions defined on this surface. Hence we require transitions maps to be differentiable too.

Thanks to our chart, we can build the tangent space at a point p of our manifold. We look at every smooth curve $\gamma : I \to M$, where I is an interval containing 0 and $\gamma(0) = p$. We define a equivalence relation between these curves: $\gamma_1 \sim \gamma_2$ if, for any f smooth function defined on a neihborhood of p in M, $(f \circ \gamma_1)'(0) = (f \circ \gamma_2)'(0)$. The equivalence classes of this relation define a vector space of dimension m called the tangent space of M at point p T_pM. A basis of this space is denoted

$$\left.\frac{\partial}{\partial x_1}\right|_p, ..., \left.\frac{\partial}{\partial x_m}\right|_p$$

The dual of this space is T_p^*M, whose basis is denoted:

$$dx_1, ..., dx_m$$

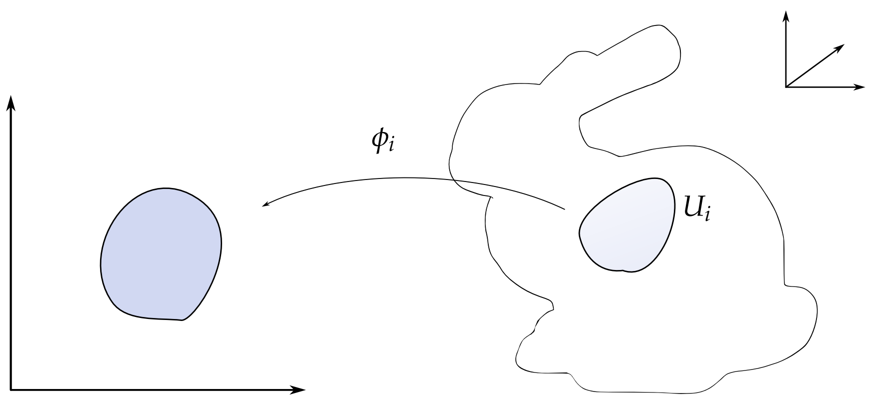


FIGURE 1.2: A chart mapping a part of the surface of a bunny to the 2d plane.

Our goal now is to actually define the integrand for integrals over our manifold. Since k-forms define "infinitesimal" things to integrate, we define differential k-forms as fields of k-forms. This corresponds with the difference between the notion of differential at a point and of differential of the function: for every point of our manifold, we define a 1-form. In particular, differential 0-forms correspond to scalar valued functions on the manifold. These can be seen equivalently as scalar fields over the mesh and scalar functions. The function that attribute this 1-form to each point will then be able to be further differentiated thanks to the exterior derivative that we will define later.

Differential k-forms are sometimes referred to as simply k-forms, while they are not the same as the k-form we previously described. From now on, we do not really have to mention k-forms in the sense of k-covectors, so k-forms will refer to differential k-forms.

Exterior derivative We define the exterior derivative in order to have a notion of differentiation not only from 0-forms to 1-forms, but generalized from k-forms to $(k+1)$-forms.

The exterior derivative $d_k : \bigwedge^k E \to \bigwedge^{k+1} E$ is the linear map defined by the following properties:

- d_0 corresponds to the differential: the differential of f at point p is $df = \sum_i \frac{\partial f}{\partial x_i} dx_i$
- if α is a l-form and β a k-form then $d_{k+l}(\alpha \wedge \beta) = d_l(\alpha) \wedge \beta + (-1)^l \alpha \wedge d_k(\beta)$
- $d_{k+1} \circ d_k = 0$

The second property generalizes the behaviour of the differential on product of functions: for f, g functions, $d(fg) = d(f)g + fd(g)$

The notation we choose for basis of $1-$forms (dx, dy, dz in 3d) is coherent with the fact that those can be obtained by derivation of the $0-$form $\alpha : (x, y, z) \to x$, $d\alpha = dx$. It follows that $d(dx) = 0$.

Using these properties, we can evaluate the differential of a differential $k-$form. As an example, let us evaluate the differential of a differential 2-form $\alpha = (3x +$

$2z)dx \wedge dy + ydy \wedge dz$

$$\begin{aligned}
d(\alpha) &= d((3x+2z)dx \wedge dy + ydy \wedge dz) \\
&= d((3x+2z)dx \wedge dy) + d(ydy \wedge dz) \\
&= d(3x+2z) \wedge dx \wedge dy + (3x+2z)d(dx \wedge dy)) + d(y) \wedge dy \wedge dz + yd(dy \wedge dz) \\
&= (3dx + 2dz) \wedge dx \wedge dy + (3x+2z)(d(dx) \wedge dy - dx \wedge d(dy)) + dy \wedge dy \wedge dz \\
&\qquad + y(d(dy) \wedge dz - dy \wedge d(dz))) \\
&= 2dz \wedge dx \wedge dy + (3x+2z)(0-0) + 0 + y(0 \wedge dz - dy \wedge 0)) \\
&= 2dx \wedge dy \wedge dz.
\end{aligned}$$

The third property is necessary in order to satisfy the generalized form of Stokes's theorem. This theorem states that, for ω some differential k-form defined over a domain Ω,

$$\int_\Omega d\omega = \int_{\partial\Omega} \omega$$

If we instead integrate $d(d\omega)$, we have:

$$\begin{aligned}
\int_\Omega d(d\omega) &= \int_{\partial\partial\Omega} \omega \\
&= 0 \quad \text{since } \partial \circ \partial = 0
\end{aligned}$$

Having $d \circ d$ is necessary in order to be coherent with this theorem. This also highlight that the exterior derivative is related to the boundary operator ∂. Now that we have the exterior derivative, we can redefine a few differential operators on functions f or vector fields $\mathbf{u}$:

- the gradient: $\nabla f = (df)^\sharp$
- the divergence: $\nabla \cdot \mathbf{u} = \star d \star u^\flat$
- the curl: $\nabla \times \mathbf{u} = (\star du^\flat)^\sharp$

Codifferential We can also define the codifferential δ of a differential k-form thanks to the differential and the hodge star operator, as

$$\delta = (-1)^k \star^{-1} d \star .$$

The codifferential satisfies the following property: for α a k-form and β a $k-1$-form, on a domain Ω with no boundary, $\int_\Omega \langle d\alpha, \beta \rangle \omega = \int_\Omega \langle \alpha, \delta\beta \rangle \omega$. This can be proven from the Stokes theorem:

$$\begin{aligned}\int_\Omega d(\alpha \wedge \star\beta) &= 0 \quad \text{(from Stokes theorem)}\\ &= \int_\Omega d\alpha \wedge \star\beta + (-1)^{k-1}\alpha \wedge d \star \beta\\ &= \int_\Omega \langle d\alpha, \beta\rangle\omega + (-1)^{k-1}\alpha \wedge \star\star^{-1}\, d\star\beta\\ &= \int_\Omega \langle d\alpha, \beta\rangle\omega - \langle \alpha, (-1)^k \star^{-1} d\star\beta\rangle\omega\\ &= \int_\Omega \langle d\alpha, \beta\rangle\omega - \int_\Omega \langle \alpha, \delta\beta\rangle\omega.\end{aligned}$$

We say that the codifferential is the adjoint of the differential with respect to the inner product defined by $(\alpha, \beta) = \int_\Omega \langle \alpha, \beta\rangle\omega$:

$$(d\alpha, \beta) = (\alpha, \delta\beta).$$

1.1.2 The Laplace-Beltrami operator

The Laplace operator, in a n-dimensional Euclidean space, is defined as:

$$\Delta = \sum_i \frac{\partial^2}{\partial x_i^2}$$

For example, in the 2d case, we have $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$, in 3d: $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$...

This operator measures how much a function deviates from its average value on a small neighborhood. This is the operator responsible for diffusion, and a famous equation involving this operator is the heat equation:

$$-\Delta T = \frac{\partial T}{\partial t}.$$

It says that the temperature diffuses along time: the more the temperature differs from its neighborhood, the more the temperature will evolve in order to minimize this difference.

Another way to obtain an intuition of the role of the operator is through the construction of a finite difference approximation on a regular grid with spatial step h (figure 1.3):

$$f(x+h, y) \approx f(x, y) + h\frac{\partial f(x, y)}{\partial x} + \frac{h^2}{2}\frac{\partial^2 f(x, y)}{x^2}$$

$$f(x-h, y) \approx f(x, y) - h\frac{\partial f(x, y)}{\partial x} + \frac{h^2}{2}\frac{\partial^2 f(x, y)}{x^2}$$

By summing the two we obtain:

$$\frac{f(x+h, y) + f(x-h, y) - 2f(x, y)}{h^2} \approx \frac{\partial^2 f(x, y)}{x^2}$$

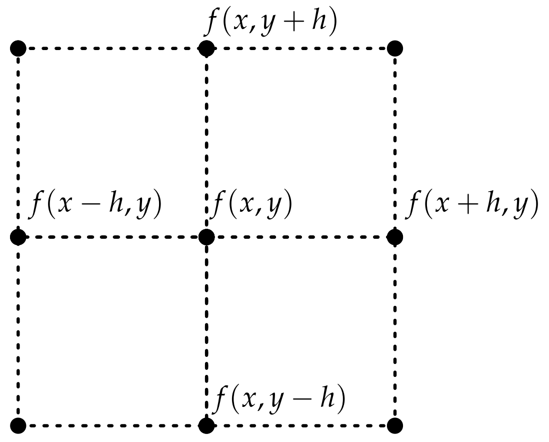


FIGURE 1.3: f discretized over a regular grid

We repeat along the other dimension, and we sum the result with the previous one, obtaining:

$$\begin{aligned}\frac{f(x+h,y)+f(x-h,y)+f(x,y+h)+f(x,y-h)-4f(x,y)}{h^2} &\approx \frac{\partial^2 f(x,y)}{x^2}+\frac{\partial^2 f(x,y)}{y^2}\\ &\approx \Delta f(x,y)\end{aligned}$$

A generalization of the Laplace operator to non-Euclidean space, called the Laplace-Beltrami operator, can be defined as $\Delta = \text{div} \circ \text{grad}$. A further generalization is the Laplace-de Rahm operator and comes from exterior calculus, using the differential d and codifferential δ: $\Delta = d\delta + \delta d$. This formulation has the advantage of generalizing the operator for general k-forms instead of only 0-forms. In the case of 0-forms, it simplifies to $\Delta = \delta d = \star d \star d$.

Properties

We list here a few properties of the Laplace-Beltrami operator that will be used in applications or when descretizing it.

Integration by part The following equality holds:

$$\int_\Omega \langle \nabla f, \nabla g\rangle dA = \int_\Omega f \Delta g dA + \int_{\partial\Omega} f\langle \nabla g, \mathbf{n}\rangle d\ell,$$

where $\mathbf{n}$ is the normal to $\partial\Omega$ pointing outward Ω. It is similar in a way to the integration by part, with ∇ having the role of the first derivative and Δ the role of the second.

A special case of this equation is:

$$\int_\Omega \langle \nabla f, \nabla f\rangle dA = \int_\Omega f \Delta f dA + \int_{\partial\Omega} f\langle \nabla f, \mathbf{n}\rangle d\ell.$$

The left hand side of the equation corresponds to the Dirichlet energy: it follows that with Dirichlet boundary conditions ($f_{|\partial\Omega} = 0$) or Neumann boundary conditions ($\langle \nabla f, \mathbf{n}\rangle = 0$), it can be evaluated using the Laplacian.

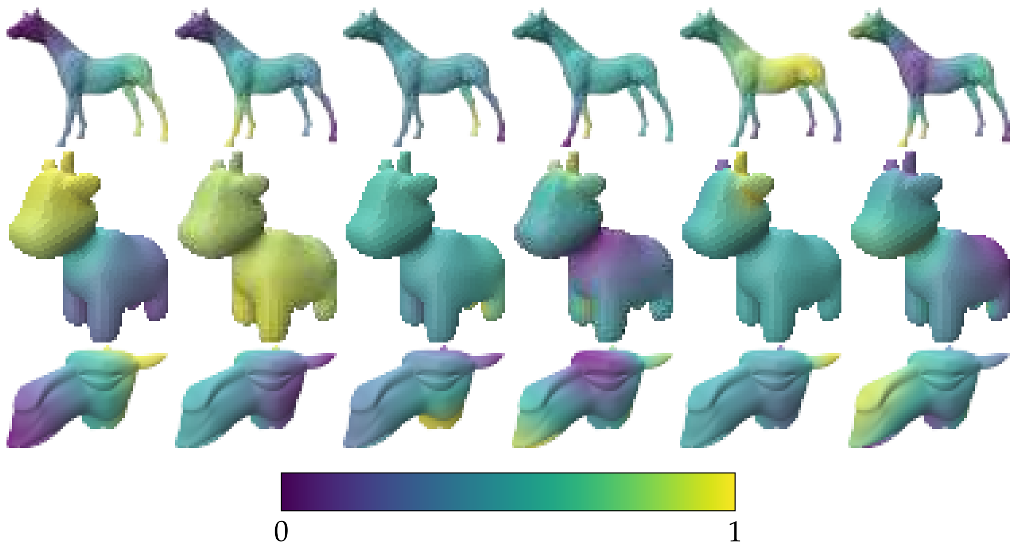


FIGURE 1.4: First non constant eigenvectors on the horse mesh (top), spot (middle) and camelhead (bottom), sorted from left (lower frequencies) to right (highers frequencies).

Curvature The Laplace Beltrami operator is closely related to mean curvature H, through the following equality

$$\Delta \mathbf{p} = 2H\mathbf{n},$$

where $\mathbf{p}$ are the positions of the surface and $\mathbf{n}$ the normals of the surface.

Eigenvalues and eigenvectors Since the operator is symmetric and non negative, it has real positive eigenvalues. In fact, these eigenvalues are known for simple cases: when we want to solve the wave equation $\partial^2 f/\partial t^2 = \Delta f$, we can search for standing wave using separation of variables: we assume $f(x, y, t) = g(x, y)h(t)$. By inserting g into the equation, we find that it must be solution of $\Delta g = kg$ where k is some constant: we are in fact looking for eigenvectors of Δ. On a one dimensional periodic domain, a basis of the solution is given by the sine and cosine functions, which correspond to the Fourier basis in this space. The same observation can be made in the plane and 3d euclidean space, so we can naturally have the idea of using the eigenvectors of the Laplace-Beltrami operator on a surface in order to build a Fourier basis on it. The lower frequencies correspond to the eigenvectors with the smallest eigenvalues (in magnitude) (see figure 1.4). On a sphere for example, this gives the well-known spherical harmonics (figure 1.5). The inverse problem is also studied: from the eigenspectrum of a Laplacian, how much a shape can we recover? This question was formulated in the paper "Can one hear the shape of a drum" by M.Kac [Kac66], and was later answered by Gordon et al. [GW96]: "You can't hear the shape of a drum" who provided different shapes with identical eigenvalues.

1.2 Geometry Processing

1.2.1 Usage in computer graphics and mesh filtering

Geometry Processing is the field grouping computational geometry works around 3d shapes, often used for computer graphics. These shapes come in different forms: for surfaces we have (among others) triangles meshes, polygonal meshes, implicit

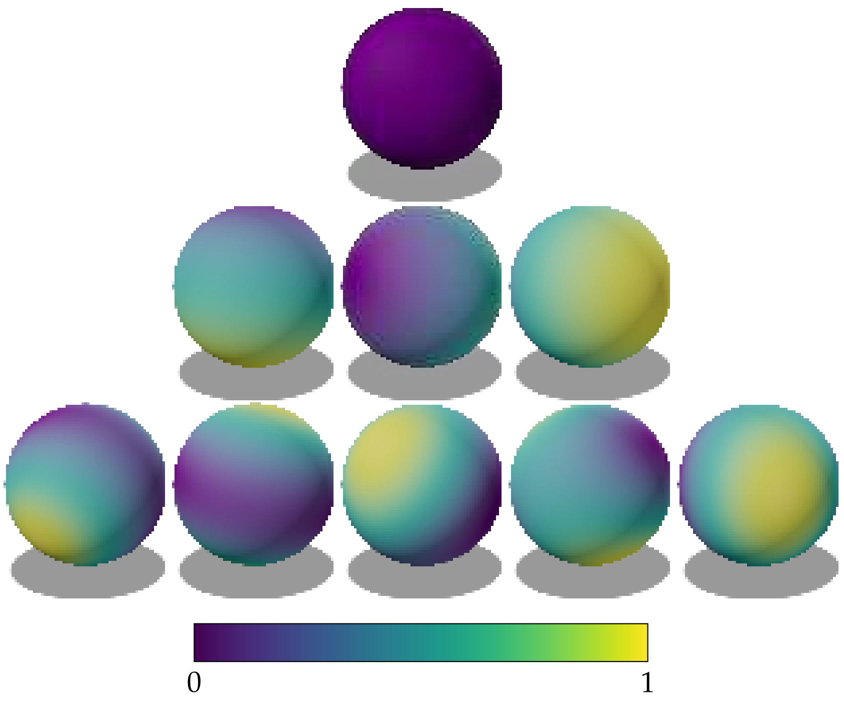


FIGURE 1.5: First 9 spherical harmonics ranked by eigenvalues magnitude: top is lower (lower frequencies) and bottom higher frequencies. These are of the form : $P_l^m(\cos\theta)\cos(m\phi)$, where l and m are two parameters (l corresponding to the row) and P_l^m is the associated Legendre polynomial.

surfaces, for volume tetrahedral meshes, voxels, implicit volumes, we also have point clouds... Problems found in geometry processing include building meshes from point cloud, removing noise, mesh simplification and mesh registration.

The Laplace-Beltrami operator has a long history of usages in geometry processing. The paper [Sor05] is dedicated to it and lists usages for mesh edition, signal processing, surface parameterization... and it has since then still widely been used. In particular, in variational approaches, where a problem is solved by finding an energy to minimize and a way to do this minimization, differential operators and in particular the Laplace-Beltrami operator can often by found. Notably, minimizing the Dirichlet Energy $\int |\nabla u|^2$ is equivalent to finding a harmonic u, up to boundary terms. This energy is often used in order to express smoothness, and among its many appearances (or slightly modified version) we find [MS89] for denoising, inpainting.., [Cha+10] for mesh deformation, for UV maps in the LSCM and Spectral Cofnromal Parameterization [Lév+02; Mul+08] and in [SC18] for computing cuts for surface parameterization.

We enumerate here a few of the the problems where methods revolving around the Laplace-Beltrami have been used, although the following list is all but exhaustive: it would take a lot more to list all of the usages and occurrences of this operator in geometry processing.

Mesh Smoothing Mesh smoothing (or surface fairing) often makes use of the *mean curvature flow* $\Delta \mathbf{p} = -2H\mathbf{n}$, where $\mathbf{p}$ are the positions, H the mean curvature and $\mathbf{n}$ the normals. This flow corresponds to the gradient of the membrane energy over the domain Ω: $E_A(\Omega) := \int_\Omega dA$, i.e. the squared L2 norm of the mean curvature over the mesh. Moving vertices along this flow acts as a gradient descent in order to minimize this energy: regions with higher curvature are to be moved the highest in order to smooth the surface, while regions with no curvature do not have to be moved. A drawback of this method is that it tends to shrink the surface: if we have a sphere, which cannot be smoothed further, the mean curvature flow is not 0 and will shrink the sphere. A first method by Taubin [Tau95] alternate between "α steps" and "μ steps", one corresponding to the flow and the other growing the shape again in order to avoid the effect of shrinkage. Later, Desbrun et al. [Des+99] formulate the method as a diffusion of the position: $\frac{\partial \mathbf{p}}{\partial t} = \lambda \Delta \mathbf{p}$, and that the gradient descent step can be integrated with an implicit Euler step instead of the explicit step previously used (see figure 1.6). It has the benefit of stabilizing the solution, which otherwise required small steps (in h^2 where h is the length of the smallest edge) in order to avoid oscillations on the surface. Here, the shrinkage is avoided by scaling the mesh back after each step. Other methods minimize the Willmore energy $E_W(\Omega) := \int_\Omega H^2 dA$, whose gradient is given by the bilaplacian of the positions $-\frac{1}{2}\Delta^2\mathbf{p}$ Crane et al. [CPS13] provide a method aiming at mimizing this energy while enabling as large as possible timesteps.

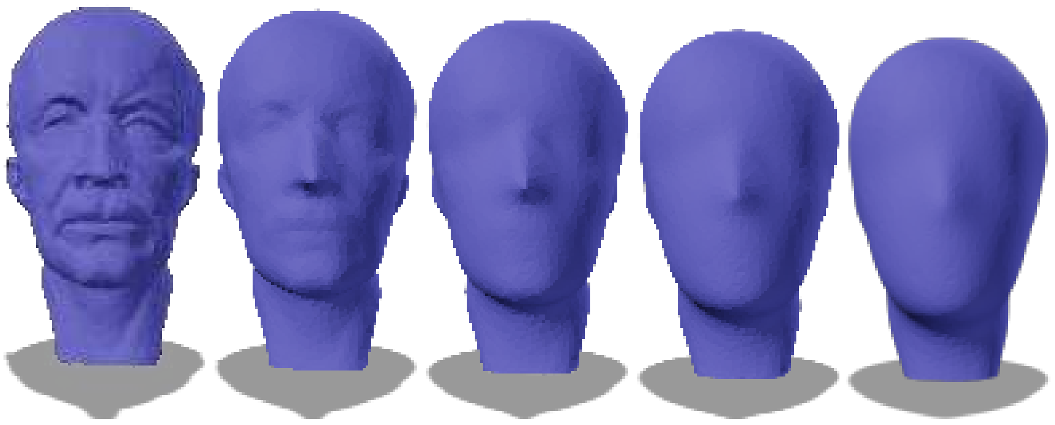


FIGURE 1.6: Left: original mesh, and several iterations of the fairing method using implicit Euler integration of the mean curvature flow.

Mesh Compression It has also been used for mesh compression, where the goal is to represent one or many meshes with the lowest possible memory. These approaches use delta coordinates instead of standard coordinates, that do not encode the absolute position of the vertex but the deviation of the vertex to the barycenter of its neighbors. Using delta coordinates and a Laplacian, it is possible to integrate these delta coordinates in order to recover the original geometry even using low precision quantization for delta coordinates: this is the method proposed by Sorkine et al. [SCT03], and they argue that the resulting visual error is lower than with higher precision global coordinates (figure 1.7). While the coordinates itself take less space than the original mesh, a geometric Laplacian still takes space: instead, simple Laplacian like the graph Laplacian that only uses the connectivity of the mesh can be used. This method is called "High-pass quantization": they show that the Laplacian tends to not preserve lower frequencies very well, and they argue that this is not a visual issue as details make most of the visual part [Sor05]. A later work by Lobaz et al.[LV14] try to remedy this issue by solving this problem on multiple scales, creating a mesh hierarchy. They reformulate the optimization process in order to have a Laplacian intervene for each level of the hierarchy, keeping lower frequencies which are not obtained with a single Laplacian, whose expression only involves its one ring. While using a geometric Laplacian is not efficient for a single mesh, it can be for a sequence of meshes where all the meshes show some similarity: for example, in an animation, as a mesh is deformed, all meshes can still be similar to the starting mesh. Váša et al. [Váš+14] propose a method where several meshes are averaged, and keeping a variation of delta coordinates called delta trajectories.

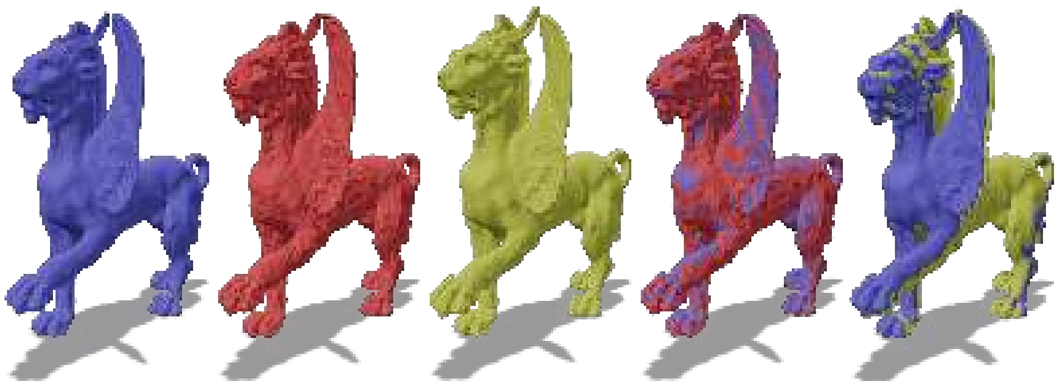


FIGURE 1.7: Blue: original mesh. Red: mesh with 8 bits quantized cartesian coordinates. Yellow: mesh recovered from 8 bits quantized delta coordinates. Visually, the mesh recovered from delta coordinates appears closer to the original than the one with naive quantization, where the noise is clearly visible. However, superposing the compressed meshes with the original show that the one recovered from delta coordinates tends to have more low frequencies errors.

Geodesics approximation Geodesic can be approximated thanks to the heat equation. Indeed, the geodesic distance can be expressed as a limit of the heat kernel $k_{t,x}$ where $k_{t,x}(y)$ expresses the amount of heat that point y has after heat that originates from point x following a t long diffusion. The geodesic distance is then given by [Var10]:

$$d(x,y) = \lim_{t \to 0} \sqrt{-4t \log k_{t,x}(y)}.$$

In [CWW17], Crane et al. argue that while computing a heat flow from a source and trying to evaluate the flow directly with the formula is not a good idea due to numerical imprecision when t is small, the direction of the gradient of the heat flow is still pretty close to the actual gradient of the distance function. Hence, they take this gradient, normalize it, take its opposite and integrate it in order to recover the distance field. A summary of the heat method is:

1. integrate the heat flow $\frac{\partial u}{\partial t} = \Delta u$ for some fixed time t
2. evaluate the vector field $X = -\nabla u / |\nabla u|$
3. solve the Poisson equation $\Delta \Phi = \nabla \cdot X$

The resulting Φ is the approximation of the distance (figure 1.8). The heat flow can start from a point, if we want the distance to this point, but it can also be the diffusion of the heat from multiple points: this is used to estimate the distance from a mesh. The method was later used by Sharp et al. [SSC19] in order to diffuse vector fields. One of the changes made in order to achieve that is the use of the connection Laplacian, operating on vector, whose discretization is described by Sharp et al. The method can be used to interpolate between vectors, or to compute logarithmic maps.

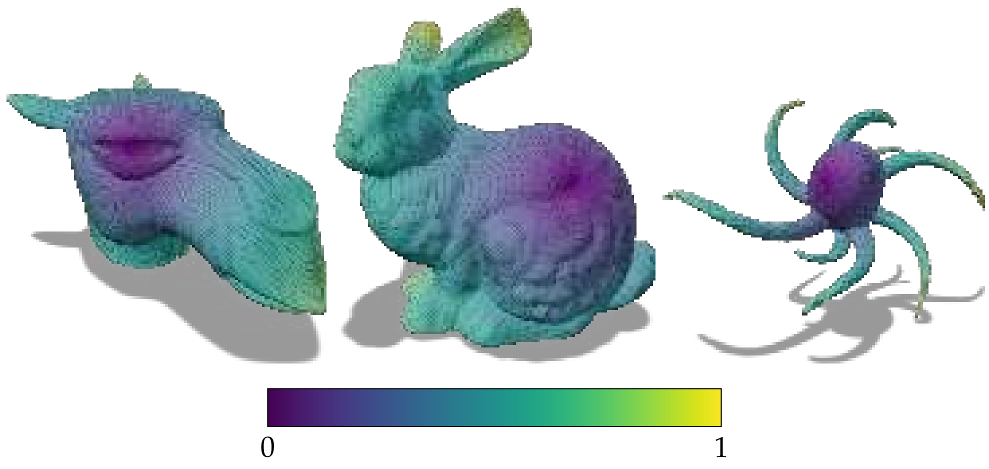


FIGURE 1.8: Geodesic distances approximated with the method from [CWW17] on the camelhead, bunny and octopus mesh.

Recently, the method has also been used in order to obtain signed distance functions to a set (such as curves on a surface) by Feng et al. [FC24]. The source of the heat flow is replaced by the normals assigned to edges of the face, which are then diffused using an edge based connection Laplacian. Once these vectors have been diffused, they are averaged to faces and normalized, and they can be integrated as in the base method to obtain a scalar field corresponding to the distance. This method can be used to estimate distance from points or sets, and signed distances from sets such as closed curve while being robust to missing parts in the sets.

1.2.2 Spectral mesh analysis

Many usages revolve around the eigenvalues of the Laplace-Beltrami operator, which enable Fourier analysis on meshes. We can for example use this basis to decompose any signal on our surface mesh, and do some filtering by removing the lower frequencies or the higher, depending on the desired effect. This basis has the advantage of depending only on the underlying surface, so it is robust to differences in triangulations: it could be used in order to identify two different triangulations of the same shape. Since the operator is also invariant to isometries, these eigenvectors are also quite robust to variations of pose. Several works have used these properties in order to do Fourier Analysis on meshes. [LZ10] describes a few of these: parameterization and remeshing, clustering and segmentation, shape correspondance... We explore a few of those: the usage in shape descriptors and in shape matching. Other applications include quad remeshing [Don+05], where the eigenvalues can be used to place the singularities, and deep learning where a diffusion layer is applied in the neural network [Sha+20], or more general geometric deep learning (see [Bro+21; Cao+20] for summaries). Diffusion can also be applied to labels in semi-supervised learning [Bud+20].

Point signatures, shape descriptors Various shape descriptors or point signatures have been developped using the eigenvalues of the Laplace-Beltrami operator or from operator involving the Laplace-Beltrami, such as the heat equation or the wave equation. The goal of these descriptors is to characterize shapes or point in order to be able to find other similar shapes or points: for example, in order to create a "shape Google", we need to define the geometrical words used to define the shapes that we want to search and retrieve [Lia+11; Bro+11]. These descriptors aim at transformation and pose invariance, in order to be then used to recognize a shape that may have been deformed or remeshed. Point descriptors have also been used for segmentation [ASC11]. The shape descriptor shapeDNA by Reuter et al. [RWP06] (and similarly the works by Jain et al. [JZ07]) is based on the first eigenvalues of the Laplace-Beltrami operator: similar shape have the same eigenvalues, while different can have different eigenvalues (not always, as shown by Gordon et al. [GW96]). ShapeDNA only uses the eigenvalues, and does not rely on the eigenvectors: on the contrary, the Global Point Signature (GPS) from Rustamov R. [Rus07] samples the values of the eigenvectors on some points of the mesh. It is an example of point signature: in order to obtain a shape descriptor, the values sampled at a few points can be taken. The Heat Kernel Signature (HKS) [SOG09; ZRH11] and the Auto Diffusion Function are based on the eigendecomposition of the keat kernel, with a time parameter t that controls wether the descriptor will tend to be local (small t, small diffusion) or global (large t, global diffusion) (see figure 1.9) The Wave Kernel Signature (WKS) [ASC11; LW15] is similar in idea to the HKS, but it is based on Schrödinger's wave equation instead of the heat equation. A comparison of these descriptors and their combinations for the task of shape retrieval can be found at [LH14].

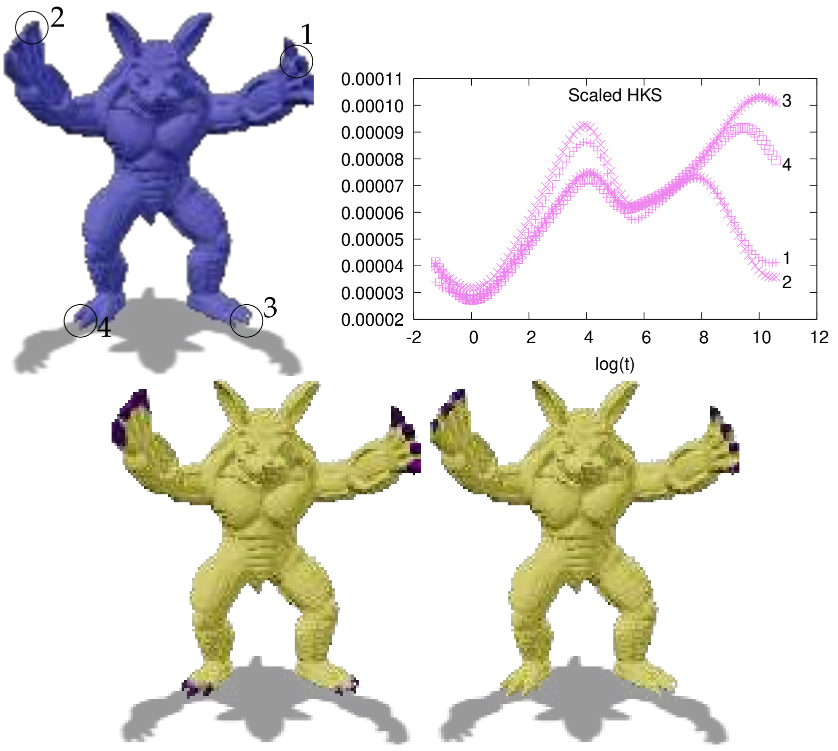


FIGURE 1.9: Heat Kernel Signature for 4 points on the armadillo mesh. On the lower row, we see the points matched to point 1 by comparing the distance of their HKS, with a lower threshold on the right row than the left.

Shape matching There are multiple approaches using the lower frequencies eigenvectors in order to do shape matching: the objective is, for two shape corresponding shape (who may have been remeshed, or deformed), to find a vertex to vertex mapping between these two shapes. The method by Lombaert et al. [Lom+13] achieves this using the lower frequencies eigenvectors: instead of performing the matching in space (which does not account for rigid transformation or deformation), it is done by selecting closest points in the spectral domain. Once the closest points have been found and a first map is obtained, the correspondance map is obtained by diffusion of this map. Another framework for shape matching is given by functional maps [Ovs+12]. The idea behind functional map is that, in order to map a function f defined on surface $\mathscr{M}$ to function g defined on surface $\mathscr{N}$, one can simply compute the correspondance between a basis of functions on $\mathscr{M}$ on a basis on $\mathscr{N}$. A common choice for this basis of function is the use of the eigenvectors of the Laplacian. Computing this correspondance (often represented as a matrix C) is the most important part of computing functional maps: this can be done by minimizing the energy measuring how much this mapping commutes with the Laplacian of each domain for example. A near-isometric mapping is then a sparse, diagonally dominant matrix, while a denser matrix usually indicate that the mapping is not isometric (the shapes may not correspond). For a more thorough introduction to functional maps, see Ovsjanikov et al. [Ovs+17].

1.3 Conclusion

We have seen that the Laplacian operator can be generalized to surfaces using the Laplace-Beltrami operator, and that it has been used in geometry processing for a wide array of tasks. In the next chapter, we will see the available methods in order to discretize it on surfaces.

CHAPTER

Discretizing the Laplace-Beltrami operator 2

Résumé

Puisque l'on veut employer l'opérateur de Laplace-Beltrami sur des surfaces discrètes, il faut pouvoir construire une discrétisation de cette opérateur.

Lorque l'on manipule une fonction définie sur une surface, on suppose qu'elle est échantillonnée sur plusieurs points de la surface (souvent correspondant aux sommets de la surface). On place toutes les valeurs prises par la fonction dans un vecteur **u**, et la construction du Laplacien revient ensuite à déterminer une matrice $\mathbb{L}$ telle que $\mathbb{L}\mathbf{u}$ représente le Laplacien de la fonction **u**. En général, on cherche $\mathbb{L}$ de la forme $\mathbb{L} = M^{-1}L$, où M est appelée *mass matrix* et L *stiffness matrix*.

Nous rappelons trois schémas permettant de discrétiser l'opérateur, et qui résultent en la formule dite cotangente sur les maillages triangulaires: les méthodes des éléments finis [Red19; DE13; EG04], les méthodes des volumes finis [DUS55; EGH00] et le calcul exterieur discret [Hir03; Des+05]. Nous nous intéressons ensuite aux maillages polygonaux, sur lesquels ces schémas donnent des résultats différents, avec des avantages et inconvénients respectifs.

Nous listons ensuite quelques propriétés que l'on souhaite obtenir sur notre opérateur, et listées par Wardetzky et al. [War+07].

Notre objectif ici est de parvenir à construire des opérateurs de Laplace-Beltrami pour surfaces digitales. Ce sont les surfaces que l'on obtient en géométrie digitale, c'est à dire la géométrie construite par des objets qui sont des sous ensembles de $\mathbb{Z}^n$. On s'intéresse dans notre cas à $n = 3$, et à des surfaces: cela correspond au surfaces des objets définis par des voxels. La construction d'opérateurs de Laplace-Beltrami pour ces surfaces n'est pas évidente. Une étude de la convergence d'estimateurs simples d'aire suggère ainsi que l'on ne peut pas simplement adapter les méthodes existantes pour les surfaces triangulaires ou polygonales usuelles. Des estimateurs convergents de normale et de courbure pour ces objets existent cepedant, comme la méthode Integral Invariant [LCL17]: on peut s'appuyer dessus pour obtenir des estimateurs d'aires convergents. On observe bien que les méthodes pour surfaces polygonales standards ne convergent pas sur ces surfaces, et qu'il est donc nécessaire de construire des méthodes adaptée.

Since we want to use the Laplace-Beltrami operator for geometry processing tasks, we need a way to discretize it on discrete surfaces such as meshes. We list here some of the available methods for building Laplace-Beltrami operators on surface meshes like triangular and polygonal surfaces, and then we briefly present the field of digital geometry and how the presented methods fail in the case of digital surfaces.

There is a wide variety of Laplacian operators (for volume meshes, for graphs) that do not necessarily try to approximate the continuous setting. We mostly focus here on first order schemes for scalar functions on surfaces. We do not, however, assume that the mesh is a manifold. It can have a boundary too. In other words, an edge can belong to more than two faces, or to only one.

Functions here are represented by their values, assumed to be sampled on the mesh (and, unless mentioned, sampled at vertices). These values are all held in a vector $\mathbf{u}$. By building a Laplace-Beltrami operator, we usually mean the following: find matrices L and M such that $\mathbb{L} = M^{-1}L$, where $\mathbb{L}$ is a matrix that takes $\mathbf{u}$ as an input and outputs a vector holding the approximations of the values of the Laplacian of $\mathbf{u}$ at respective vertices. The matrix $\mathbb{L}$ is called the strong (or pointwise) version of the Laplacian, and is rarely explicitely built: instead we build the two matrices M and L, where M is called the *mass matrix* and L the *stiffness matrix*. The matrix L also represents the weak (or integrated) version of the Laplacian.

Since the difference between $\mathbb{L}$ and L is that one is pointwise and the other integrated, M can be interpreted as the matrix that takes a function as input and outputs its locally integrated version (by multiplying each value by a choosen associated area), and while this does not always correspond to the way it is built, it is quite common to find a *lumped* version of this matrix that is diagonal (by summing each row and assigning the value to the diagonal) following this interpretation and allowing easier matrix inversion. Several versions of the mass matrix exist (based on the barycenter dual, the Voronoi dual...): in this section, we mostly focus on the stiffness matrix.

We will make use of several existing schemes to solve PDEs and to build differential operators on various type of discrete surfaces, which can be roughly categorized as follows:

- **Finite element methods** Finite Element Methods (FEM, [Red19; DE13; EG04]) are amongst the most common methods for PDE solving on meshes. These approaches often rely on using a (sometimes implicit) basis function over the surface, and use weak PDE formulations of problems in order to formulate the operator construction as an evaluation of integrals. These approaches have the advantage of being well studied, and have the most extensive litterature on convergence guarantees [BEL89].

- **Finite volumes** Finite Volume (FV, [DUS55; EGH00]) methods rely on discretizations of the dual in order to associate cells to vertices, and convert integration on those cells into boundary evaluations using Ostrogradsky's theorem.

- **Mimetic finite differences** Mimetic Finite Differences (MFD, [LMS14]) and Discrete Exterior Calculus (DEC, [Hir03; Des+05]) based approaches focus more on conserving algebraic properties (such as $\mathrm{div} \circ \mathrm{curl} = 0$ and $\mathrm{curl} \circ \mathrm{grad} = 0$). They usually find a way to derive a an operator (such as the gradient, the divergence...) and derive the other in order to follow these properties. These approaches tend to be easier to generalize to more complexe geometries, but require the addition of stabilization terms. Virtual Element Methods (VEM, [Vei+12]) have shown that is possible to use FEM approaches while avoiding explicit definition of the basis, resulting in an approach that also resembles MFD based ones.

2.1 Desirable Properties

We elaborate on the properties of the operator we want to build, more particularly on the stiffness matrix L. We generally want L to be sparse (uses less memory, easier to invert, to compute eigenvalues...), and to be of the following form:

$$(Lx)_i = \sum_{j \in V(i)} \omega_{i,j}(x_i - x_j),$$

where $V(i)$ is the neighborhood of vertex i, and $\omega_{i,j}$ are the sought coefficients of L. In other words, the stiffness matrix measure the average difference between the values at a vertex and its neighbors.

Note that this formula also encompasses the wide variety of "Laplacians" that can be encountered that do not aim to be geometric Laplacians and to approximate the Laplace-Beltrami operator, such as the graph Laplacian, the Tutte Laplacian...

Wardetzky et al. [War+07] provides a few additional desirable properties, as well as the "no free lunch" result: it is not possible to provide an operator upholding all these properties for any mesh. These properties are the following:

- symmetry: $\omega_{i,j} = \omega_{j,i}$. This guarantees real eigenvalues and an orthogonal basis from eigenvectors.
- locality: $\omega_{i,j} \neq 0 \implies v_i$ and v_j are neighbors
- linear precision: if the domain is embedded into the plane, and the function u on this domain is linear, then $Lu = 0$ at interior vertices
- positive weights: $i \neq j \implies \omega_{i,j} \geq 0$, this ensures the maximum principle. The maximum principle stipulates that harmonic function have no local extremas at interior points: if we want to find the equilibrium of temperature on a surface with Dirichlet boundary conditions, then the hottest point and the coldest point are on the boundary.
- positive semi-definiteness: L should be PSD. Indeed, the discrete Dirichlet energy, given by $u^\top Lu$, should be positive
- convergence: when a smooth surface S is approximated by a sequence of meshes M_i that converge toward S (under an appropriate norm), the discrete solutions of Dirichlet problems given by each $\mathbb{L}_i$ converge toward the solution in the smooth setting. While this may seem to be the most important property, this is also the hardest to prove, and except for the cotan Laplacian (which has its convergence proven by Wardetzky [War08]) we will not discuss it.

Since all of these properties cannot be held at the same time, we sometimes choose one to discard. For example, some formulations such as the cotan laplacian (that we will see in section 2.2) do not follow the maximum principle while some alternative uphold it at the cost of locality, and argues that locality in this formulation is not necessary ([BS07]).

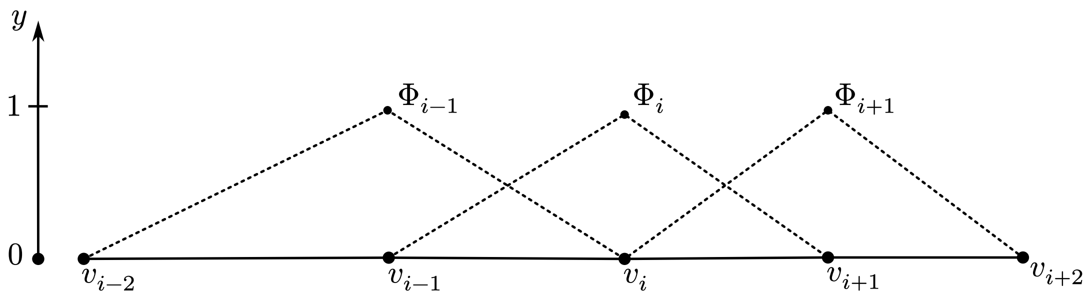


FIGURE 2.1: Hat functions over a linear mesh.

2.2 Triangle meshes

While there is a wide variety of schemes for Laplace-Beltrami operator construction (Finite Element based methods, Finite Volume based ones, Discrete Exterior Calculus...), almost all of them result in the cotan formula on triangle meshes (see equation 2.7). We'll go through a few of these methods and see how this formula can be derived from these different schemes, as well as how it can be built on intrinsic Delaunay triangulations in order to avoid negative weights.

2.2.1 Finite Element Method

We show here a summary of the finite element method for Poisson equations in order to build the Laplace-Beltrami operator. For a more complete approach to the finite element method on surfaces for similar problems, see the article from Dziuk et Elliott [DE13], or the book by Ern et Guermond [EG04] for a more general introduction to classical finite elements.

The first step for a finite element approach is to define basis functions. In the case of scalar functions defined at vertices, and for a first order approximation, the natural basis is the basis made of hat functions. In two dimensions (so on a linear mesh), hat functions ϕ_i are the functions that are piecewise linear on each segment of the mesh, with each function having a value of 1 at one vertex (which is the vertex that will be associated to this function) and 0 at all others (see figure 2.1). These basis provide a way to define a function on the whole mesh from values sampled at vertices: for $\mathbf{f} = [\mathbf{f_0}, \mathbf{f_1}, ..., \mathbf{f_n}]$ we have the associated function $f = \sum_i \mathbf{f_i} \Phi_i$ which linearly interpolates the sampled values within f. These functions have a few properties: they are continuous over the whole domain, they are harmonic inside each face and linear on the edges.

In the 3d case, and especially on triangle meshes, this can also be achieved quite naturally through the use of barycentric coordinates. These barycentric coordinates are defined for a point v inside the triangle as, for each vertex v_i, the ratio of the area of the triangle made with v and the edge opposite to the vertex over the area of the whole triangle A (see figure 2.2).

These coordinates can be used to build a parameterization s, t of the triangle using $s = \frac{A_i}{A}$ and $t = \frac{A_j}{A}$ (and it follows that $1 - s - t = \frac{A_k}{A}$. From this parameterization of the triangle, we can now easily build 3 linear functions $f_i(s,t) = s$, $f_j(s,t) = t$ and $f_k(s,t) = 1 - s - t$. Then, in order to go back to the scale of the whole mesh, we stitch together all functions whose values are 1 on a choosen vertex. These functions now define a linear basis for functions over the mesh: they form a piecewise linear basis $\Phi_1, \Phi_2, ...\Phi_n$ for functions over the whole mesh, where Φ_i has value 1 at vertex i and 0 at all the others (see figure 2.3). Using this basis, and applying a finite element

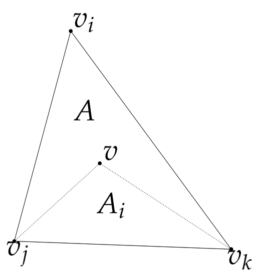


FIGURE 2.2: Barycentric coordinates for a point inside a triangle. For each vertex v_i, the associated value is the area A_i of the triangle defined by the point with the opposite edge to the vertex over the area of the whole triangle A. These coordinates are indeed positive, equal 1 when a point is on the corresponding vertex (and 0 on the opposite edge), and since $A_i + A_j + A_k = A$ their sum is indeed one.

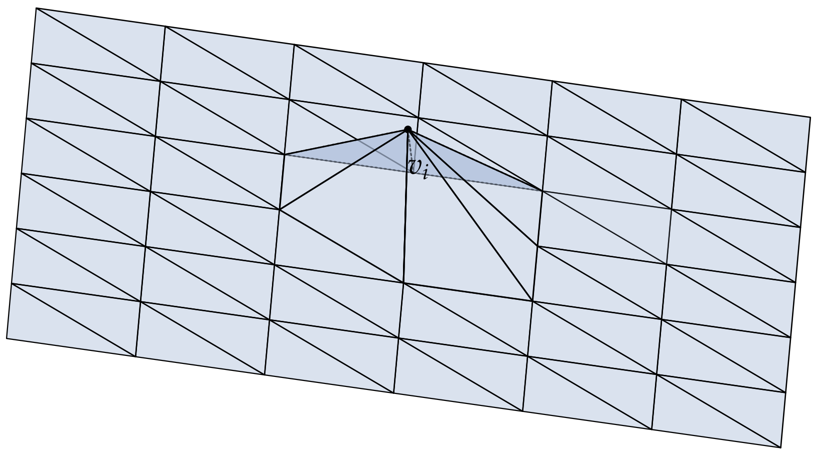


FIGURE 2.3: Hat functions over a piece of triangulation of the plane.

schemes, we can derive the entries for the stiffness matrix L and the mass matrix M.

We want to solve the weak Poisson equation, meaning the weak form of $\Delta f = g$. Weak solutions satisfy the following property:

$$\int_\Omega \Phi \Delta f dA = \int_\Omega \Phi g dA, \quad \forall \Phi.$$

Here we take Φ, f and g as functions in the linear space derived from the basis we just defined, so it becomes:

$$\forall i, \int_\Omega \Phi_i \Delta f dA = \int_\Omega \Phi_i g dA. \tag{2.1}$$

Now, we use the "integration by part" (and we assume no boundary) property of the Laplace-Beltrami operator in order to transform the first part:

$$\begin{aligned} \int_\Omega \Phi_i \Delta f dA &= -\int_\Omega \nabla \Phi_i \cdot \nabla f dA \\ &= -\int_\Omega \nabla \Phi_i \cdot \nabla \sum_j \mathbf{f_j} \Phi_j dA \\ &= -\sum_j \mathbf{f_j} \int_\Omega \nabla \Phi_i \cdot \nabla \Phi_j dA. \end{aligned}$$

We can also decompose the second part:

$$\begin{aligned} \int_\Omega \Phi_i g dA &= \int_\Omega \Phi_i \sum_j \mathbf{g_j} \Phi_j dA \\ &= \sum_j \mathbf{g_j} \int_\Omega \Phi_i \Phi_j dA. \end{aligned}$$

We now define L and M as:

$$L_{i,j} = \int_\Omega \nabla \Phi_i \cdot \nabla \Phi_j dA \tag{2.2}$$

$$M_{i,j} = \int_\Omega \Phi_i \Phi_j dA, \tag{2.3}$$

Equation 2.1 becomes:

$$\begin{aligned} \forall i, -\sum_j \mathbf{f_j} \int_\Omega \nabla \Phi_i \cdot \nabla \Phi_j dA &= \sum_j \mathbf{g_j} \int_\Omega \Phi_i \Phi_j dA \\ \Leftrightarrow L\mathbf{f} &= M\mathbf{g}. \end{aligned} \tag{2.4}$$

Solving the Poisson problem now means inverting L. All that is left to do is to evaluate the coefficients $L_{i,j}$ (equation 2.2) and $M_{i,j}$ (equation 2.1) to build the matrices. We look at these integrals inside a single triangle, and the matrices are then obtained by summing each triangle's contribution. We denote as $L_\triangle$ and $M_\triangle$ the matrices obtained from a single triangle.

For $M_\triangle$, the evaluation is quite straightfoward, and results in:

$$M_\triangle = \frac{A}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}. \tag{2.5}$$

For $L_\triangle$, we have to go through a few steps first. Our main observation is that since our Φ basis are linear inside each triangle, their gradients are constant inside each triangle. It follows that to evaluate $\int_\triangle \nabla\Phi_i \cdot \nabla\Phi_j dA$, we just have to evaluate the dot product between the two vectors on a single point and multiply by the area A of the triangle. We now provide an expression for these gradients. Let us look at

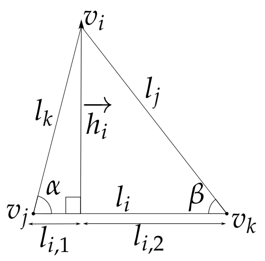


FIGURE 2.4

Φ_i on figure 2.4. We know that $\Phi_i(v_i) = 1$, and $\Phi_i(v_j) = \Phi_i(v_k) = 0$, thus that $\nabla\Phi$ is perpendicular to the edge opposite to v_i (so colinear to $\overrightarrow{h_i}$). Moreover, Φ_i equals 0 at the start of $\overrightarrow{h_i}$, and 1 at the end, meaning $\nabla\Phi_i \cdot \overrightarrow{h_i} = 1$. We can now assert:

$$\nabla\Phi_i = \frac{\overrightarrow{h_i}}{||\overrightarrow{h_i}||^2}. \tag{2.6}$$

In order to evaluate $\int_\triangle \nabla\Phi_i \cdot \nabla\Phi_j dA$, we have to distinguish two cases: $i = j$ and $i \neq j$.

- **i = j**,

  From 2.6, we have:
  $$\nabla\Phi_i \cdot \nabla\Phi_i = \frac{1}{||\overrightarrow{h}||^2}.$$

  We have the following equations tying edge lengths and angles, $\cot\alpha = l_{i,1}/h_i$ and $\cot\beta = l_{i,2}/h_i$

  The area of the triangle is thus given by

  $$\begin{aligned} A &= \frac{1}{2}(l_i||h_i||) \\ &= \frac{1}{2}(l_{i,1}||h_i|| + l_{i,2}||h_i||) \\ &= \frac{1}{2}(\cot\alpha||h_i||^2 + \cot\beta||h_i||^2) \end{aligned}.$$

  We obtain the following result:

  $$\int_\triangle \nabla\Phi_i \cdot \nabla\Phi_i = \frac{1}{2}(\cot\alpha + \cot\beta).$$

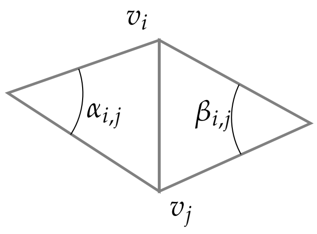


FIGURE 2.5: Angle used in the cotan formula

- $\mathbf{i \neq j}$

  Let us first evaluate the dot product between the unit vectors $\frac{\overrightarrow{h_i}}{||h_i||}$ and $\frac{\overrightarrow{h_j}}{||h_j||}$

  Both $\frac{\overrightarrow{h_i}}{||h_i||}$ and $\frac{\overrightarrow{h_j}}{||h_j||}$, rotated clockwise, are equal to the vectors $\frac{\overrightarrow{v_j v_k}}{||\overrightarrow{v_j v_k}||}$ and $\frac{\overrightarrow{v_k v_i}}{||\overrightarrow{v_k v_i}||}$ respectively. So

$$\begin{aligned}\frac{\overrightarrow{h_i}}{||h_i||} \cdot \frac{\overrightarrow{h_j}}{||h_j||} &= \frac{\overrightarrow{v_j v_k}}{||\overrightarrow{v_j v_k}||} \cdot \frac{\overrightarrow{v_k v_i}}{||\overrightarrow{v_k v_i}||} \\ &= -\frac{\overrightarrow{v_k v_j}}{||\overrightarrow{v_k v_j}||} \cdot \frac{\overrightarrow{v_k v_i}}{||\overrightarrow{v_k v_i}||} \\ &= -\cos\beta,\end{aligned}$$

  we also have the following equality tying edges and lengths: $h_j = l_i \sin\beta$. The area of the triangle is given by:

$$\begin{aligned}A &= \frac{1}{2}(l_i||h_i||) \\ &= \frac{||h_j||||h_i||)}{2\,sin\beta}.\end{aligned}$$

  We can now conclude:

$$\begin{aligned}\int_{\triangle} \nabla\Phi_i \cdot \nabla\Phi_j &= A\frac{\overrightarrow{h_i} \cdot \overrightarrow{h_j}}{||h_i||^2||h_j||^2} \\ &= \frac{A}{||h_i||||h_j||}\frac{\overrightarrow{h_i} \cdot \overrightarrow{h_j}}{||h_i||||h_j||} \\ &= -\frac{1}{2\sin\beta}\cos\beta \\ &= -\frac{1}{2}\cot\beta.\end{aligned}$$

We obtained the coefficients for L by adding them triangle by triangle. On a manifold surface mesh, we can directly give the value for non diagonal coefficients. For v_i, v_j sharing an edge, following notations from figure 2.5:

$$L_{i,j} = -\frac{\cot\alpha_{i,j} + \cot\beta_{j,i}}{2}. \tag{2.7}$$

Diagonal coefficients are then given as the opposite of the sum of the other coefficients on the row. This is the cotan formula, whose first apparition dates from the work of MacNeal [Mac49] (section 3.2) (and perhaps even earlier in the works of Courant). Note that the coefficients can be computed without actually computing the angles, just from the lengths l_i, l_j and l_k (respectively opposite to v_i, v_j, v_k) of the triangle. The area can be computed from Heron's formula $A = \sqrt{s(s-l_i)(s-l_j)(s-l_k)}$ where $s = \frac{l_i+l_j+l_k}{2}$. Then, $\cot\theta_i = \frac{l_j^2+l_k^2-l_i^2}{4A}$. Since we only use lengths, this can be used in intrinsic triangulations (which we will see later), and it is easier to adapt when using a correction metric.

2.2.2 Finite Volume Method

Another way to obtain the cotan formula for the Laplacian is through the finite volume method. We use the formulation $\Delta = \mathrm{div} \circ \mathrm{grad}$: first we discretize the gradient, then the divergence, and we compose them. The gradient can be quite naturally defined per face, and we use a discretization of the dual mesh of our surface in order to evaluate the divergence. In general, finite volume method focus on the discretization of the divergence in order to ensure that conservative quantities are indeed conserved. For a more complete presentation of these methods, see the book by Eymard et al. [EGH00].

The first step is to discretize the gradient. We reuse the same linear interpolation inside each triangle that we used in the previous section: as we have seen, this results in a constant gradient inside each triangle. This means we can see our gradient operator as an operator that assign a vector field defined on triangle from a scalar field defined on vertices. We could build a matrix representing this operator, but we will only make use of the formula from equation 2.6. We will denote the gradient of our function f as the vector $\nabla f_\triangle$ in the calculations that will follow:

$$\nabla f_\triangle = \mathbf{f_i}\frac{\overrightarrow{h_i}}{||h_i||^2} + \mathbf{f_j}\frac{\overrightarrow{h_j}}{||h_j||^2} + \mathbf{f_k}\frac{\overrightarrow{h_j}}{||h_j||^2}.$$

Since we want our Laplacian to be defined on vertices, the divergence should be an operator that brings the gradient back to vertices: it should take a vector field on triangles as input and outputs a scalar field on vertices. What we will obtain is actually the integrated version of the divergence, which would then have to be divided by the area if we wanted the pointwise version, but we use this version since we want the integrated Laplacian. In order to evaluate this divergence, we make use of two elements: a dual to our mesh in order to associate a cell to each vertex, and the Green-Ostrogradski theorem in order to facilitate the evaluation of the integral of the divergence.

The Green-Ostrogradski theorem states that the integral of the divergence of a vector field on a domain M is equal to the integral of the flux of this vector field along the boundary of the domain ∂M:

$$\int_M \mathrm{div}\overrightarrow{f}\,dA = \int_{\partial M} \overrightarrow{f}\cdot\overrightarrow{N}\,d\ell,$$

where $\overrightarrow{N}$ is the normal along the border of the domain M. This gives us a way to evaluate the integrated divergence: if we define a domain we want to integrate the divergence of a gradient field on, in the discrete setting we only have to evaluate a sum of dot product between the border of this domain and our gradient field.

In order to define the domain of this integral, we use the dual Voronoi mesh. More precisely, we care about the boundary of those cells, on which we will base our evaluation. This dual Voronoi mesh, illustrated on 2.6, is built by taking the circumcenter of each triangle, and adding edges crossing the edges of the primal (these edges are orthogonal to the primal). In some cases, the circumcenter may fall outside the triangle, which is something that we will go back to later when discussing intrinsic triangulations.

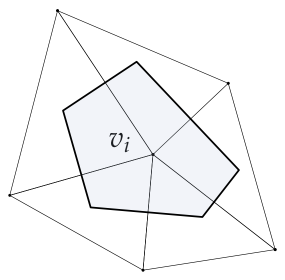


FIGURE 2.6: Dual Voronoi cell for vertex v_i defined for the Finite Volume Method.

We focus on one triangle, and on one vertex inside this triangle, with the notations from figure 2.7. The integrated divergence is given by

$$\nabla f_\triangle \cdot (l_{ij}\overrightarrow{n_{ij}} + l_{ik}\overrightarrow{n_{ik}}).$$

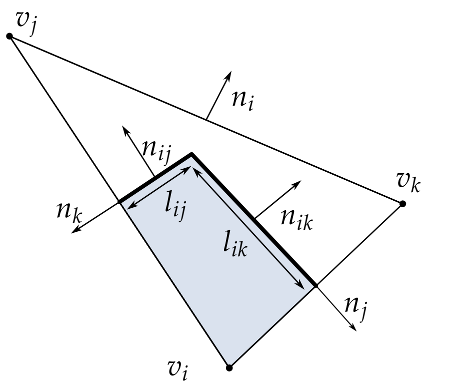


FIGURE 2.7: Dual Voronoi cell for vertex v_i defined for the Finite Volume Method.

Using the fact that the integral of the normal on a closed curve is 0, we have:

$$l_{ij}\overrightarrow{n_{ij}} + l_{ik}\overrightarrow{n_{ik}} + \frac{l_j}{2}\overrightarrow{n_j} + \frac{l_k}{2}\overrightarrow{n_k} = 0$$

and

$$l_i\overrightarrow{n_i} + l_j\overrightarrow{n_j} + l_k\overrightarrow{n_k} = 0,$$

Therefore:

$$\begin{aligned} \nabla f_\triangle \cdot (l_{ij}\overrightarrow{n_{ij}} + l_{ik}\overrightarrow{n_{ik}}) &= -\nabla f_\triangle \cdot (\frac{l_j}{2}\overrightarrow{n_j} + \frac{l_k}{2}\overrightarrow{n_k}) \\ &= \nabla f_\triangle \cdot \frac{l_i}{2}\overrightarrow{n_i}. \end{aligned}$$

Reusing the notations from 2.4, we have $A = \frac{1}{2}l_i||h_i||$, meaning $l_i = \frac{2A}{||h_i||}$. We also have $\overrightarrow{n_i} = -\frac{\overrightarrow{h_i}}{||h_i||}$. Therefore:

$$\nabla f_\triangle \cdot \frac{l_i}{2}\overrightarrow{n_i} = -A\nabla f_\triangle \cdot \frac{\overrightarrow{h_i}}{||h_i||^2}.$$

We are now back to the calculations done from equation 2.6 and after. The obtained results still hold:

$$\begin{aligned} -A\nabla f_\triangle \cdot \frac{\overrightarrow{h_i}}{||h_i||^2} &= (\mathbf{f_i}\frac{\overrightarrow{h_i}}{||h_i||^2} + \mathbf{f_j}\frac{\overrightarrow{h_j}}{||h_j||^2} + \mathbf{f_k}\frac{\overrightarrow{h_j}}{||h_j||^2}) \cdot \frac{\overrightarrow{h_i}}{||h_i||^2} \\ &= \mathbf{f_i}\frac{\cot\alpha + \cot\beta}{2} - \mathbf{f_j}\frac{\cot\beta}{2} - \mathbf{f_k}\frac{\cot\alpha}{2}. \end{aligned}$$

We repeat the process for each triangle, and again we end up with the cotan formula for the stiffness matrix. For the mass matrix, we build a diagonal matrix where each value holds the area of the dual cell associated to the vertex (sometimes called Voronoi mass matrix).

2.2.3 Discrete Exterior Calculus

Another way to discretize the operator (and end up with the cotan formula again) is through the Discrete Exterior Calculus scheme. In this method, we discretize some "basic" operators first such as the differential or the hodge star operator, which are then put together in order to build more complex ones, such as the Laplace-Beltrami operator. For an extensive description of the method, see [Hir03; Des+05]. In this section we concentrate on building a Laplace-Beltrami operator on a triangle mesh. We use the following definition of the Laplace-Beltrami operator: $\Delta = \delta d$, where d is the differential and δ the codifferential.

We work on k-forms; since we are studying a surface mesh, we have 0, 1 and 2-forms. We need to define where we discretize each: in order to naturally integrate them, we discretize them on elements of the same dimension: 0-forms are put on vertices, 1-forms on (oriented) edges and 2-forms on (oriented) faces. These orientations are usually given for faces, and chosen arbitrarly for edges. We also define dual forms on the elements of the dual mesh. For the dual mesh, we use the same Voronoi dual as on figure 2.6.

The differential is pretty easy to define: for each edge, we associate the value of one of its edge minus the other, choosen respecting the orientation of each edges (we add the vertex it points to and substract the one it points from).

In order to build the codifferential, recall its definition: it is the self adjoint operator of the differential. In other words, for any 1-forms α and 0-form β, $\langle d\beta, \alpha\rangle = \langle \alpha, \delta\beta\rangle$. Suppose that we have the hodge star operators $\star_0$ and $\star_1$ (respectively for 0- and 1-forms), then, in matrix form, this translate to : $\alpha^\top d^\top \star_1 \beta = \alpha \star_0 \delta\beta$. Since this holds true for any α and β:

$$d^\top \star_1 = \star_0 \delta$$
$$\delta = \star_0^{-1} d^\top \star_1 .$$

Since we have already built the differential operator, all that is left to do is compute the matrices $\star_0$ and $\star_1$. Since these matrices define inner products on 0-forms and 1-forms, we must define these inner products. $\star_0$ is the inner product on 0-forms, corresponding to scalar functions, defined on vertices in our case: this corresponds to our mass matrix. It can be built by associating the area of the dual cells of the vertices. Since this is the mass matrix, we can remove its inverse fom the Laplace-Beltrami to obtain the expression for the stiffness matrix: $d^\top \star_1 d$.

In the continous setting, the Hodge star is defined as following the identity, for any k-forms α, β:

$$\langle \alpha, \beta \rangle \omega = \alpha \wedge \star_k \beta.$$

In some sense, the Hodge star takes a k-form and gives the $(n-k)$-form that "complements" the volume. Thus, in its discrete form, it makes sense to define it as an operator that takes a form tied to a primal element and to assign it a form tied to a dual element. In the case of 1-form on edges (inside a surface), we assign the 1-form of the same "value" on an orthogonal edge. Then, it makes sense to have it as a diagonal matrix. If we want the complement of a 1-form valued 1 on an edge to a unit volume, we must "compensate" the lengths of the edges, thus the mapping operator is the ratio of the lengths of the dual edge and the primal edge (for a more rigorous definition of the Discrete Hodge Star operator, see Hiranis's thesis [Hir03] section 4.1 or the work of Desbrun et al. on DEC [Des+05], section 6).

We can develop the expression involving our differential and the hodge star operator to obtain the following expression for the coefficients of the stiffness matrix:

$$L_{i,j} = \frac{*l_{i,j}}{l_{i,j}},$$

where $*l_{i,j}$ is the length of the dual edge to $l_{i,j}$. This ratio can be evaluated (figure 2.8) and we end up finding the cotan formula again.

2.2.4 Intrinsic Laplacian, non-manifold Laplacian

An improvement on the standard cotan Laplacian is the instrinsic Laplacian proposed by Bobenko and Springborn [BS07]. The standard cotan formula does not guarantee positive weights, and indeed experimentally we can observe that the maximum principle is not always respected. However, the cotan formula does guarantee positive weights if the triangulation is Delaunay: for any pair of adjacent triangles, the sum of opposite angles α and β must be lower than π. Indeed, since $0 < \alpha \leq \pi$ and $0 < \beta \leq \pi$:

$$\begin{aligned} \cot \alpha + \cot \beta \geq 0 &\Leftrightarrow \frac{\cos \alpha}{\sin \alpha} + \frac{\cos \beta}{\sin \beta} \geq 0 \\ &\Leftrightarrow \frac{\cos \alpha \sin \beta + \cos \beta \sin \alpha}{\sin \alpha \sin \beta} \geq 0 \\ &\Leftrightarrow \frac{\sin(\alpha + \beta)}{\sin \alpha \sin \beta} \geq 0 \\ &\Leftrightarrow \alpha + \beta \leq \pi. \end{aligned}$$

A geometric interpretation as to why negative cotan weights can cause problems can be found while deriving the cotan formula through the Finite Volume Method, when building the dual Voronoi mesh: when a pair of triangle is non-Delaunay, the dual edge crossing the edge shared by the two triangles is flipped (see figure 2.9), and there is an overlap between two cells.

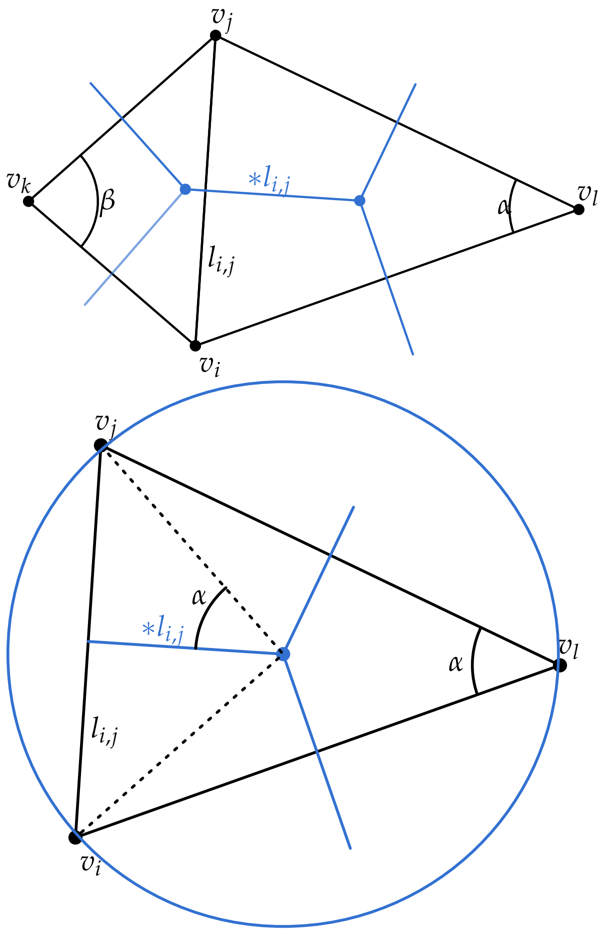


FIGURE 2.8: We want to evaluate the ratio $\frac{*l_{i,j}}{l_{i,j}}$. We focus on one triangle (bottom): using the fact that the vertices of the dual are the circumcenter and the inscribed angle theorem, we know that the two angles labeled α are indeed the same. The cotan is then obtained since we have built a right triangle with angle α whose opposite edge length is $l_{i,j}/2$ and adjacent edge length is $*l^1_{i,j}$

When a pair of triangles is non-Delaunay, it is possible two replace them with a Delaunay pair by applying a triangle flip. While this is easy to define in the plane (see figure 2.10), it becomes trickier on surface meshes. Indeed, just changing the connectivity would change the geometry of the surface we are working on, which is something we want avoid: we only want to change the connectivity of the surface, but to keep the same underlying surface. The solution to this is to use intrinsic Delaunay triangulations, which also corrects the length of the newly created edge so that it corresponds to the length of the edge traced on the underlying surface (figure 2.11). By repeating flips for non-Delaunay pair, we eventually achieve a Delaunay triangulation of the mesh (Bobenko and Springborn [BS07] also provide a proof that the edge flip algorithm finishes).

The method then consists in building the cotan Laplacian not on the given triangulation but on the intrinsic Delaunay triangulation. This results in changed connectivity, meaning we break the locality property, but we avoid unwanted negative weights and respect the maximum principle.

A further improvement is the non-manifold Laplacian. While the cotan method can be formulated as face based (and thus be used on non-manifold surfaces), the

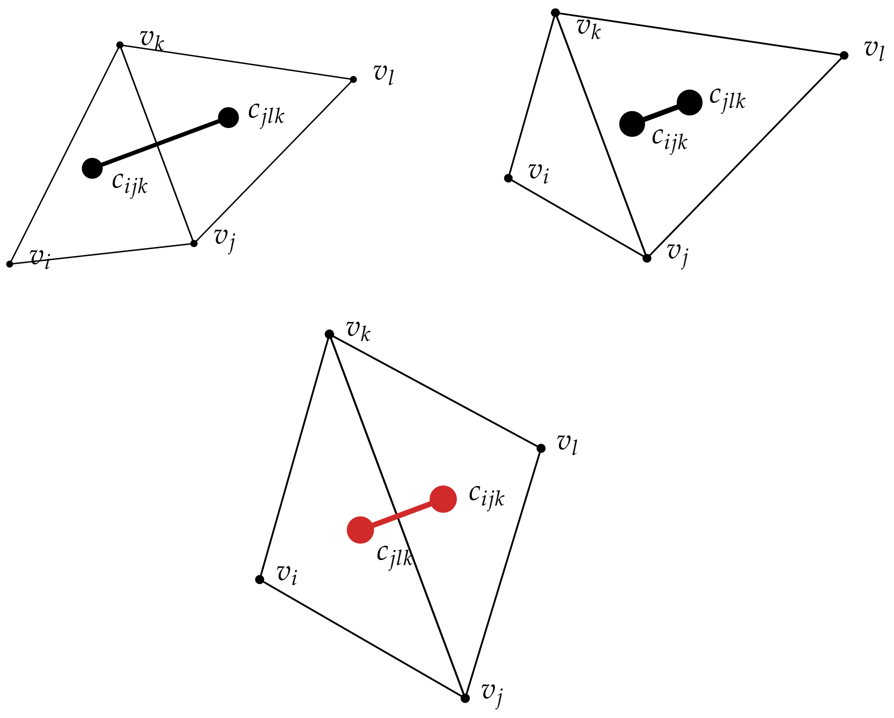


FIGURE 2.9: When all angles are less than $\frac{\pi}{2}$, the centers of the circumcircles stays inside their respective triangles. When an angles reaches $\frac{\pi}{2}$, the center of the circumcircle of the triangles enter another triangle: while this means we integrate things that should happen inside the triangle outside it, the dual mesh is still correctly defined. When the sum of the opposite angles reaches π, the edge flip, and this bad dual configuration is reflected in the form of negative weights in the stiffness matrix.

method to build Delaunay surfaces is usually based on flipping non Delaunay triangle pairs. This notion of Delaunay flips involve two triangles sharing an edge, and is not well defined when three triangles share the same edge (see fig 2.12). The method by Sharp and Crane [SC20] solves this problem by building a "tufted cover" above the original triangulation, on which it is possible to compute a Delaunay triangulation, and then average the weights obtained on the cover in order to bring back the construction to the original surface. In order to build this tufted cover, each face is duplicated into a "back" and "front" face. Then, faces on the same "side" are stitched together (figure 2.13). The resulting surfaces (there can be multiple ones) are manifold and without boundary: we can compute the intrinsic triangulation on each one of them and build an intrinsic cotan Laplacian with positive weights. Since each face is duplicated into 2 others, half the contribution of each Laplacian built is used. The article also provides a data structure to represent the tufted cover and the flips efficiently.

2.2.5 Strong consistency

There exist methods such as those of Belkin et al. [BSW08] and later Hildebrandt and Poltier [HP11] in order to build an operator with a strong consistency guarantee. Indeed, Wardetzky [War08] shows that the cotan operator converges in a weak sense, but also shows that it does not always converge in L^2 norm, and thus there

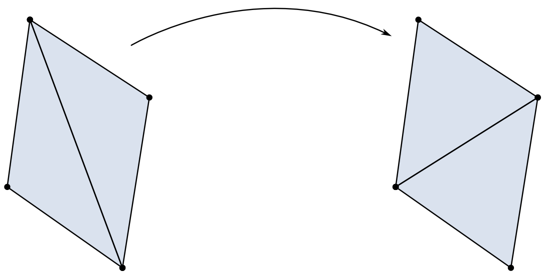


FIGURE 2.10: In the plane, applying a flip to correct a non Delaunay triangle pair is easy: we just replace the shared edge by the edge connecting the previously opposed vertices of the triangle pair.

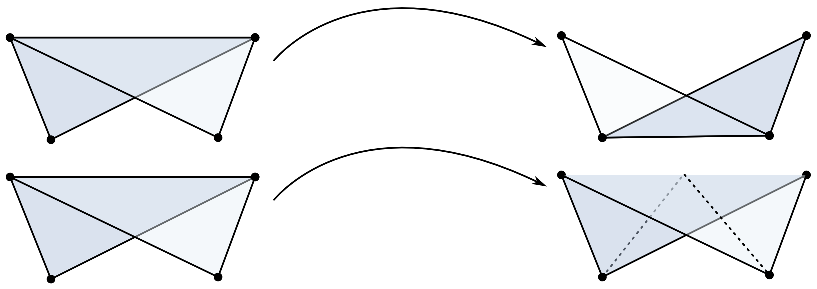


FIGURE 2.11: On a surface, if we just connect the opposite edge, we change the geometry (up). What we want is to flip the edge but to keep it on the same geometry (down), so we define the new edge to be on the original surface.

is no guarantee of convergence for the L^∞ norm. These other methods converge in a pointwise sense, and thus also for the L^2 norm, under an assumption of bounded aspect ratio of the meshes when the precision gets lower (meaning no triangles get stretched).

A first method, by Belkin et al. [BSW08], later generalized to point clouds in arbitray dimensions by Belkin et al. [BSW09], is based on heat kernels. It builds a dense matrix, by associating for every vertex pair (not restricted to adjacent pairs) v_i and v_j:

$$\omega_{i,j} = \frac{A_{ijk} + A_{jkl}}{3} \frac{1}{4\pi h^2} e^{-\frac{||v_i - v_j||^4}{4h}}.$$

which corresponds to an approximation of the heat kernel, and where h is a positive

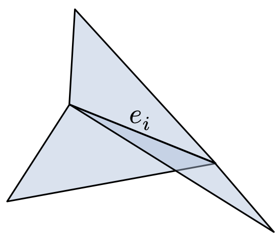


FIGURE 2.12: Here, the edge e_i is shared by three triangles: it is a non manifold edge. We cannot really define Delaunay property and Delaunay flips here: should the sum of the three angles be taken into account (then the flip should be done on three triangles at once), should flips be computed two by two (how do you two triangles without flipping the remaining one?)

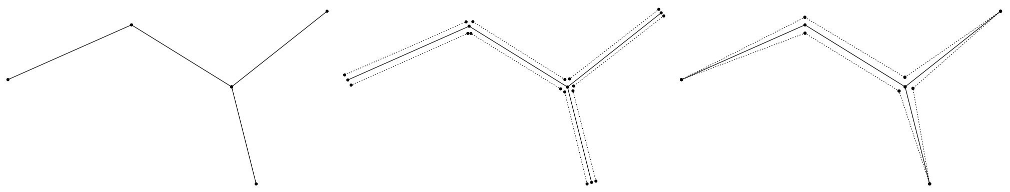


FIGURE 2.13: Illustration of the process of building a tufted cover on a curve in 2d (which could correspond to the cross section of a non manifold edge). Here a vertex is shared by three edges: it is non manifold. The first step of the process consists in duplicating each face into a back and a front face. Note that the vertex positions are not actually moved, they only are here for ease of visualization. The second step consists in stitching these faces together. On the border, faces are stitched to themselves. The result is a closed, manifold curve.

constant which act as the parameter of heat diffusion (quantifying how much time we diffuse) and $||v_i - v_j||$ is the geodesic distance between v_i and v_j. While the paper provide strong convergence guarantees (and experimental results in accord), this method shows several downsides: it can be slow to compute due to the estimation of the geodesic distance, and it requires a dense matrix which takes a lot of memory and can be costly to invert.

A second method by Hildebrandt and Poltier [HP11] is based on *r-local* functions: functions whose supports gets smaller as the mesh precision gets higher, while having constant L^1 norm and bounded $W^{1,1}$ norm. Then, a matrix built from these functions is built and serve as a replacement for the inverse mass matrix (we still use a cotan stiffness matrix). The evaluation of these hat functions is faster (since they have local support), and the geodesic distance can even be approximated by the Euclidean distance for ease of computations. The resulting matrix is sparse, and even if it is not symmetric, the authors argue that the non symmetric part becomes small and can be removed. This method is thus better at providing an operator that can be inverted, but is still more complicated than inverting a diagonal mass matrix.

2.3 Polygonal Meshes

Polygonal meshes provide more freedom over the geometry, but are harder to work with. This difficulty can be seen when trying to find a generalization for barycentric coordinates inside a face. The problem is the following: inside a polygon defined by vertices $v_1, v_2, ...v_k$, for any point v inside the face, find the weights $\lambda_1, \lambda_2, ...\lambda_k \geq 0$ such that $v = \sum_i \lambda_i v_i$ and $\sum_i \lambda_i = 1$. Several methods have proposed coordinate systems for convex polygons (such as Wachspress coordinates [Wac75]), and some have also provided coordinates for non-convex polygons, where it is more difficult to uphold positivity of all the weights: mean value coordinates [Flo03; HF06] (figure 2.14), maximum entropy coordinates [HS08], moving least squares coordinates [MS10] (see the survey by Chen et Gotsman [CG16] for a more detailed list and comparisons). These weights can be seen as basis functions over the polygon: $p \rightarrow \lambda_i(p)$, which partition the polygon well, but are not necessarily harmonic inside the polygon and do not always have a closed form.

The approach of Martin et al. [Mar+08] is to build numerical approximation at the scale of each face for a basis of harmonic functions. This approximation is built as a combination of harmonic ambient kernels (and thus also harmonic inside a flat domain such as the inside of the polygon). In the case of surfaces in 3d spaces, these

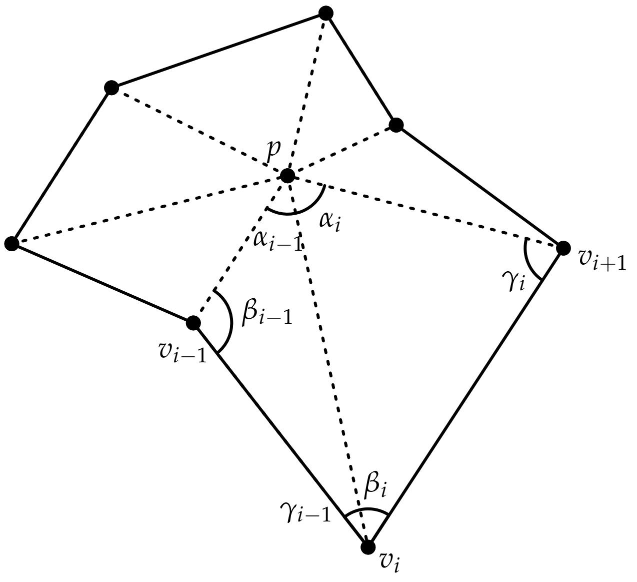


FIGURE 2.14: Star shaped polygon. Here, the Wachpress weight ω_i [Wac75] for v_i at point p is given by $(\cot\gamma_{i-1} + \cot\beta_i)/||p - v_i||^2$, while the mean value weight is given by $(\tan(\alpha_{i-1}/2) + \tan(\alpha_i)/2)/||p - v_i||$. Coordinates λ are then given by normalization of the weights: $\lambda_i = \omega_i / \sum_j \omega_j$.

kernels are choosen of the form $k_c(p) = log(\frac{1}{||p-c||})$. The harmonic basis is then taken as a weighted sum of these kernels centered near the edges of the polygon, with an additional linear term. In order to determine these weights, an optimization process tries to make each function of the basis replicate the behavior of hat functions as much as possible: they have to be 1 valued at one vertex and 0 at the others, and to be linear on the edges. This optimization process samples the expected values on the edges, and fits the weights accordingly. Since each of the kernels composing the basis are harmonic, their sum of these kernels is harmonic too. We thus have a basis of functions (illustrated on figure 2.15) that are harmonics, that approach linear behavior on edges as well and are 1 valued on one vertex and 0 on the others.

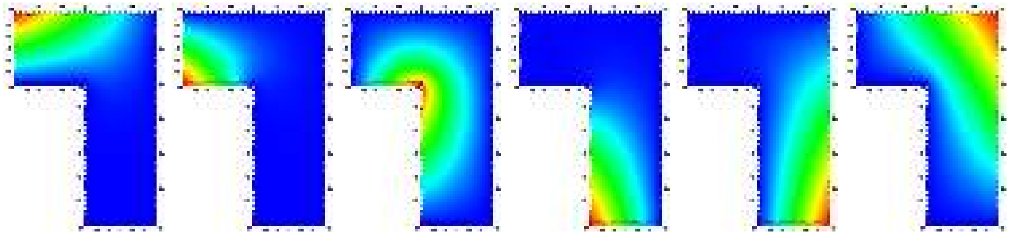


FIGURE 2.15: Harmonic basis functions approximating the behaviour of a hat function on a non convex polygonal face. The spheres on the edges correspond to where the constraints are placed for the minimization process, and the ones slightly outside correspond to the center of the kernels. Figure is taken from [Mar+08]

A downside to these approaches is that once we have defined our basis functions, there are no general way to derive an analytic formula for the integral needed when using a finite element scheme: in our case, $\int_\Omega \nabla\Phi_i \nabla\Phi_j$ where Ω is the polygon and Φ_i, Φ_j any two basis functions. Indeed, while these methods are based on functions that can be differentiated, integration over Ω usually is not easy, and instead numerical integration is required which can take quite some time: on triangle meshes, we previously only had to compute edge lengths and cotangents in order to build our stiffness matrix.

Note that we have elluded here a difficulty that can in fact invalidate those previous methods: how can we define which points are inside a face or not? While this is easy to tell for planar faces, in the general case of polygonal meshes it does not hold true that all faces are planar. This is a problem for barycentric coordinates (since we no longer know which points to define them on), and for harmonic basis: while they are harmonic on the ambient space and on a flat surface, they are not on a curved surface, and the question remain of how to integrate them over a domain we have not actually defined.

An in depth survey of the different methods available for general polygonal surfaces, as well as a comparison can be found in a survey by Bunge and Botsch [BB23]. We examine here three of them. For the specific case of quad meshes, a method averaging the results given by possible triangulations of the faces was also proposed by Xiong et al. [XLH11].

2.3.1 Virtual Triangles

Some methods rely on adding a virtual vertex "inside" the face and then triangulating the face [Bun+20; Bun+24]. The process is repeated for each polygon, forming a triangle mesh on which it is easier to build the operator, using the cotan formula (or a variation such as the intrinsic Delaunay cotan). Once the operator is built, there is the matter of going back to the original geometry (i.e. removing the added vertices). This is done by distributing the weights associated to each added vertex to its neighbors. Each approach provides different ways to find the position of the new vertices to redistribute the weights of added vertices to their neighbors.

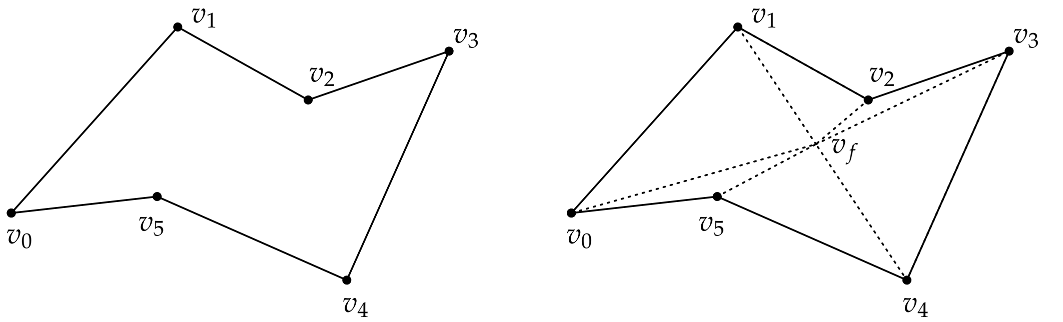


FIGURE 2.16: Virtual vertices are added inside a non planar face in order to then triangulate it.

The newly added vertices are created as a barycenter of the vertices of the face:

$$v_{new} = \sum_i w_i v_i,$$

where $\sum_i w_i = 1$. Using these weights, a prolongation matrix P can be built, to take values from the vertices of the triangulated face to the original face. For each vertex of the boundary, we directly give the value to the corresponding vertex on the original surface. For values on the newly created vertices, it is distributed according to the weights w_i. Therefore, for a single face f, the shape of the prolongation operator on the face $P_f \in \mathbb{R}^{(n+1)\times n}$ is:

$$P_f = \begin{bmatrix} w_1 & w_2 & ... & w_n \\ 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & ... & 1 \end{bmatrix}$$

We can build a larg prolongation operator $P \in \mathbb{R}^{n_v+n_f\ timesn_v}$, where the n_f first rows corresponds to the virtual vertices and have non zero values for the vertices belonging to its face, and the following n_v rows correspond to the identity. Assuming we build a cotan stiffness matrix $L_\triangle$ and mass matrix M_δ on the triangulation, the resulting matrices for the polygonal surface are $L = P^\top L_\triangle P$ and $M = P^\top M_\triangle P$.

The question remains of how to determine the weights. A first method was proposed by Bunge et al. [Bun+20]: here, the weights are chosen so that the sum of the squared areas of the created triangles is minimized. This makes sense in order to give a point that will be close to the interior of the polygon, as for planar face this will yield a point inside the face. However, this is not enough to uniquely define the weights, as multiple choices can give the same position: as an additional constraint, they choose the weights minimizing the L^2 norm of the vector $w = (w_1, w_2, ..., w_n)$, meaning the weights should be as uniform as possible.

Another was presented in [Bun+24]. The goal here is to make the Laplace-Beltrami operator robust by minimizing the trace of the stiffness matrix. In the case of triangle surfaces, the trace of the stiffness matrix is the sum of the cotangent weights. They show that this also the case when the projection is added, so in order to stabilize the operator they minimize the energy defined by the sum of the cotangents inside the newly created triangles. The energy to minimize is differentiated, and the minimization is done through a projected Newton method. A line search is used at each step of the process in order to avoid triangle flips, and the virtual point of Bunge et al. [Bun+20] is used as a starting point (it is guaranteed that there is no flip for star shaped polygons). This can however only be done on planar polygons: when a polygon is non planar, the weights are computed on the polygon projected to the plane maximizing its area. Once the weights have been choosen, the same process as before is used in order to build the projection and the global operator.

2.3.2 Finite Volume Methods

Another method is the diamond Laplacian [BBA21], which is more in line with finite volume methods, in particular with Discrete Duality Finite Volume (DDFV) methods [Her00; DO05].

This method relies on a dual mesh. In order to place vertices, it uses the same method as Bunge et al. [Bun+20]: find the vertex such that the triangulation of the primal induced by it minimizes the sum of squared areas. It also relies on the same prolongation operator P. Using these vertices, we can define "diamond cells" around each edge (figure 2.17). We now want to define the integrated gradient over this cell.

Using Ostrogradski's theorem, we can transform the evaluation of the integral of the gradient into a boundary evaluation:

$$\int_\Omega \nabla f dA = \int_{\partial\Omega} f\mathbf{n} d\ell.$$

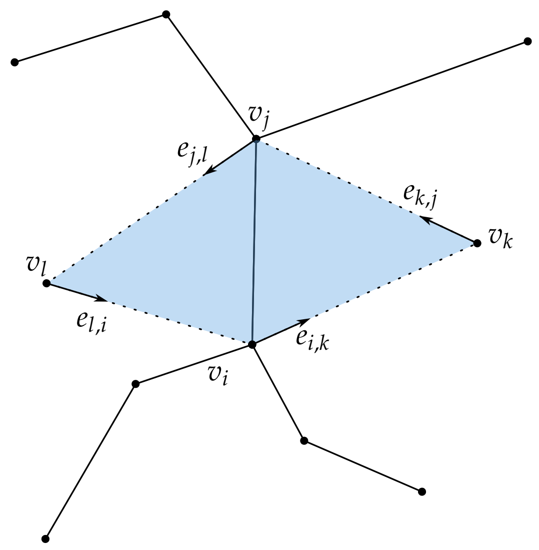


FIGURE 2.17: Diamond cell around primal edge v_iv_j

Assuming f is linear on the boundary, the second term can be written as the sum:

$$\frac{u_i + u_k}{2}e^{\perp}_{i,k} + \frac{u_k + u_j}{2}e^{\perp}_{k,j} + \frac{u_j + u_l}{2}e^{\perp}_{j,l} + \frac{u_l + u_i}{2}e^{\perp}_{l,i}. \tag{2.8}$$

where the $e_{i,k}, \ldots$ are the edge vectors (i.e. unit vectors in the same direction as the edge), and $\perp$ denotes the $\pi/2$ clockwise rotation (giving the normals from the edge vectors). The pointwise version is then deduced by dividing by the area of the cell $|D|$, which can be computed from lengths of the triangles. Since the diamond cell only makes two triangles shared by an edge intervene, the two triangles can be rotated around the edge to end up in the same plane. Then, all of the normals are in the same plan and so is the vector we obtained as approximation of our gradient. Bunge et al. [BBA21] call this the "intrinsic gradient", and it is the one we use: the computation is done to compute a vector not in 3d but in 2d (since contained in the plane).

Since equation 2.8 gives a local expression for the gradient linear in u, the pointwise gradient can be computed globally as a matrix $G_\diamond$, that attributes gradient from vertices values to vectors on primal edges. We index it with a symbol $\diamond$ to emphasize the fact that it is defined on the "diamond" mesh with added vertices, not on the original one: we could define the gradient on the surface as $G = G_\diamond P$ (note that the primal edges are still the same, we only need to apply P at the start of the operator). Also note that the gradient used here are intrinsic (and only have two coordinates), if we wanted the complete expression of the gradient we would need to add the operator taking intrinsic gradients to ambient space (we do not need it here for the Laplace-Beltrami operator).

We also need a inner product for vector on edges $M_{1,\diamond}$. We simply take the diagonal matrix where the value for each coordinate of the vector on the edge is the area of the diamond cell. Given these matrices, the stiffness matrix is given by

$$L = -P^\top G_\diamond^\top M_{1,\diamond} G_\diamond P.$$

The mass matrix can be built in a similar way $M = P^\top M_{0,\diamond} P$ where $M_{0,\diamond}$ defines the inner product for vertices on the diamond mesh. To build this inner product, we just attribute to each vertex one quarter of the areas of its adjacent diamond cells.

2.3.3 Mimetic Finite Differences and Discrete Exterior Calculus based approaches

A first MDF based approach for general polygonal meshes is the one proposed by Alexa et al. [AW11], which solves the problem of non planar faces by first projecting each face onto the plane that maximize the vector area for the computation. Once the face is projected, the gradient can be discretized through a boundary integration, and other operators can be discretized after.

Another approach by De Goes et al. [DBD20] is also in the line of MFD and DEC, and seeks to avoid the need of projecting the face first. The boundary integral of the gradient requires an expression of the normal to the face on the boundary, in other word telling what direction points away from an interior that we have not defined. In order to avoid this problem, the method starts by using a discretization of the cogradient operator $\nabla f^{\perp}$, which is equal to the gradient operator rotated $\frac{\pi}{2}$ around the surface normal $\mathbf{n}(x)$. The stokes theorem giving:

$$\int_{\Omega} \nabla f(x) dA = \int_{\partial\Omega} f(x)\mathbf{t}(x) \times \mathbf{n}(x) d\ell,$$

we can derive

$$\int_{\Omega} \nabla f(x)^{\perp} dA = \int_{\partial\Omega} f(x)\mathbf{t}(x) d\ell.$$

If we have a matrix A sending vertices values to their average value on edges and E a matrix holding the edge vectors, this integral can be computed as $E^{\top} A\mathbf{f}$. For a linear function $\mathbf{f}(x) = s^{\top} x + b$, its gradient is s and its cogradient can be expressed as $[\mathbf{n}]f$ where $[\mathbf{n}]$ is the matrix such that $[\mathbf{n}]u = \mathbf{n} \times u$.

Noting X the matrix holding the coordinates of the vertices, and $B = AX$,

$$\left(\int_{\Omega} [\mathbf{n}] dA \right) s = E^{\top} A(Xs + \mathbf{1}b) = E^{\top} Bs$$

since the integral of 1 along the edge vectors is 0 (they form a closed path). Since this holds true for any s, we have

$$\int_{\Omega} [\mathbf{n}] dA = E^{\top} B.$$

They show that this matrix is equal to the matrix $[\mathbf{a}]$ where $\mathbf{a}$ is the vector area of the polygon, corresponding to the maximum area of the polygon projected onto a plane held by the normal of this plane (and is the same as the one used for projection by Alexa and Wardetzky [AW11]). Therefore, this area can be used to obtain a face normal $\mathbf{n}$ and an face area a, and the gradient operator can be derived from the integrated cogradient operator as:

$$G = -\frac{1}{a}[\mathbf{n}]E^{\top} A.$$

Following a DEC fashion, they then discretize the musical isomorphisms $\flat$, that assign a 1-form to a vector, and $\sharp$, that assign a vector to a 1-form. The $\sharp$ operator is choosen to mimic the relation $\nabla f = \sharp df$.

The flat operator takes a vector, projects it to the tangent plane and then distribute it to the edges of the face. It has the expression

$$V = E(Id - \mathbf{n}\mathbf{n}^{\top}).$$

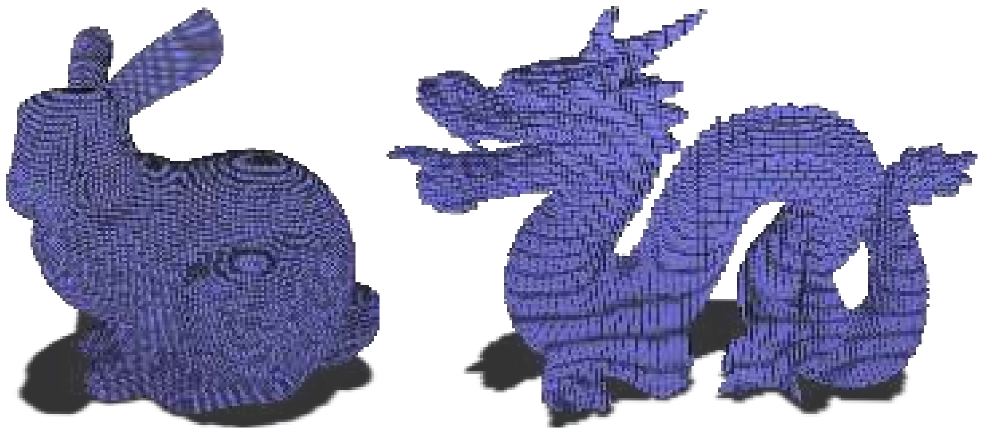


FIGURE 2.18: Digital surface from the bunny and the dragon meshes

The sharp operator does the inverse:

$$U = \frac{1}{a}[\mathbf{n}](B - c\mathbf{1}^\top),$$

where c is the center of the face and $\mathbf{1}$ a vector filled with 1. It verifies $G = UD$, where D is the differential along the edges.

The inner product for 1-form would then be taken as the following: for two 1-forms, we compute the dual vector of each (using the $\sharp$ operator) and use the dot product on the two resulting vectors, divided by the area of the face on which the vector was built. However, this would not be a positive definite product: the $\sharp$ operator sends 1-forms to vectors orthogonal to $\mathbf{n}$.

This inner product can be stabilized using a projection term. This projection term is taken as

$$P = Id - VU.$$

It removes the tangent part of a 1-form, and leaves only the part in the kernel of the $\sharp$ operator. This projection is added to the inner product as a stabilization term, with a λ factor. Hence the inner product for 1-forme is defined as

$$M_1 = \frac{1}{a}U^\top U - \lambda P.$$

We can now give the stiffness part of the Laplace-Beltrami operator on the face as

$$L = D^\top M_1 D.$$

By summing per face operators into a global matrix, we can build our global Laplace-Beltrami operator.

2.4 Shortcomings: Digital Surfaces

2.4.1 Digital surfaces

One of the main focus of our work is the construction of this operator on "digital surfaces". These surface are studied in the general field of Digital Geometry [KR04a], which could be described broadly as geometry in $\mathbb{Z}^d$, corresponding to pixels for $d = 2$ and voxels for $d = 3$ (see figure 2.18).

While the 2d case (geometry in $\mathbf{Z}^2$) is probably the most encountered (as it corresponds to images), the 3d case can arise when encoutering data captured with a volumetric sensor. For example, Flin et al. [Fli+05] study 3d digital objects obtained as the result of X-ray tomography. Such objects are also produced as the results of some medical imaging tools, such as MRI scanners.

We will focus on surfaces in this set, called digital surfaces. Digital surfaces are defined as the boundary of a subset of $\mathbb{Z}^3$. Since we call voxels our volume elements, our surface elements (squares) will be called surfels. We can define the Gauss digitization $D_h(X)$ of a volume $X \subset \mathbf{R}^3$ as the intersection of this volume with $(h\mathbb{Z})^3$, where h is the digitization step (figure 2.19): $D_h(X) = X \cap (h\mathbf{Z}^3)$ The sets of points will then be the base of each voxel of our digital volume. This gives us a way to transform a continuous shape into a digital one: from now on, when we talk about the "original shape" that gave us a digital surface, we assume that the digital surface is the result of the Gauss digitization of this shape.

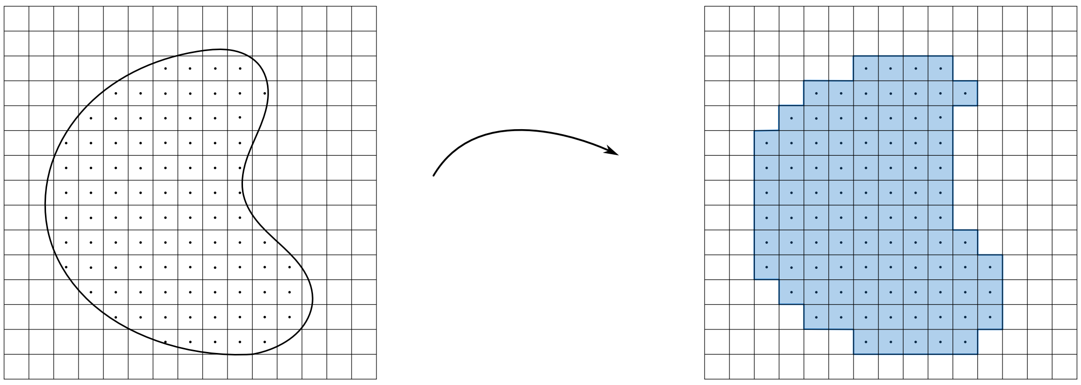


FIGURE 2.19: Gauss digitization of a 2D shape

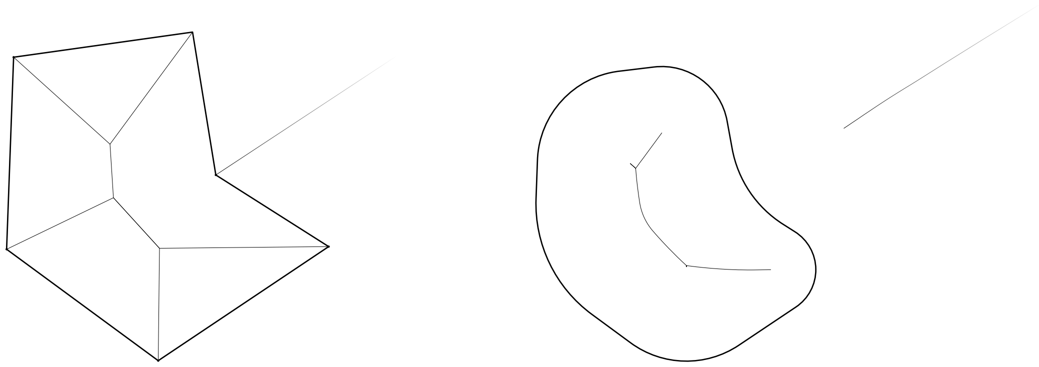


FIGURE 2.20: Shapes and their medial axis. On the left shape, the medial axis touches the shape: the reach is 0. This reflects the fact that around sharp corners, if we sample a point around the shape we cannot always assign it uniquely to a closest point on the shape. On the right shape, the reach is stricly positive.

Assuming such a surface is built as the boundary of the Gauss digitization of a volume, we can derive desirable quantities and properties [LT16]. Concerning the estimators, the notion we are interested in is the multigrid convergence, meaning that the estimator converges when the gridstep h tends to 0 (as the grid gets more precise). Properties from the continuous setting do not always carry on. We list here the first few definitions and properties of [KR04b; LT16]. We need to define the reach, which intervene in most of these properties.

In order to define the reach, we'll first define the notion of medial axis. The medial axis $MA(\Omega)$ of a shape Ω is the set of points such as the number of points of $\partial\Omega$ closest to it is more than one. The reach of a surface is the minimal distance from it to its medial axis. In other words, it gives the largest distance at which any point away from the surface has exactly on closest point on this surface.

$$\rho(\Omega) = d(MA(\Omega), \partial\Omega). \tag{2.9}$$

In a sense, having a strictly positive reach allows the assignation of each point of a discretization to the original surface without ambiguity (figure 2.20), provided the discretization step h is small enough. Note that several points on the discretization can end up with the same projection on the shape. However theorem 3 of Lachaud and Thibert [LT16] shows that the measure of the set of these points tends to 0 if the reach of the shape is strictly positive.

We can derive several other properties, still assuming that the reach ρ of the original surface is strictly positive. We have the convergence of the discretization of the shape to the shape in Hausdorff distance (theorem 1 of [LT16]). However, it is not necessarily manifold (some edges can be shared by most by two faces), but if h is small enough, for a surfel where the normal of the original surface is n, the surface is manifold as long as the angle between n and each axis is greater than $1.260h/\rho$ (theorem 2 of [LT16]).

An example of property that is not carried on from the continuous setting to digital surfaces is the convergence of the area of the estimated shape. On a triangle mesh, as the mesh becomes finer the area of the mesh tends to the area of the original surface (granted that the surface is nice enough: that the normals of the triangle meshes converge to the original normals, a counterexample is the Schwarz lantern). In the case of a digital surfaces, even for simple shapes (such as the discretization of the sphere), it does not converge (figure 2.21). Since we need to evaluate integrals such as $\int_{\partial\Omega} fg dA$ in a finite element based approach, we need to to be able to evaluate them for simple functions ($f = g = 1$ at least, which gives the area), or else the results will most likely not converge.

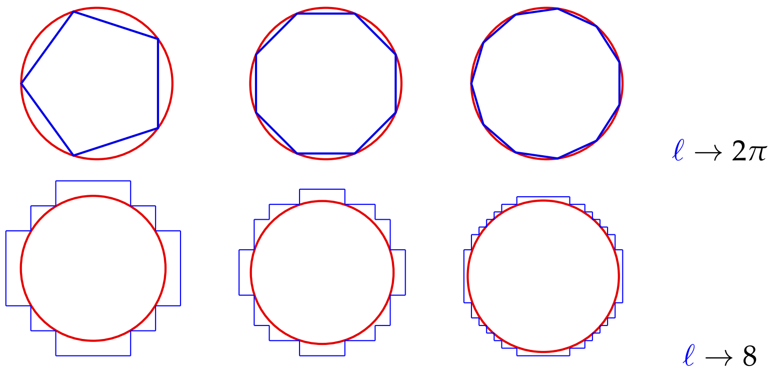


FIGURE 2.21: Top: regular polygons inscribed in a unit circle have their perimeter converge to 2π, the shape they're "sampled" on. Bottom: the Gauss digitization of a circle has a constant perimeter of 8 (up to a small difference of h depending on the discretization) and does not converge to the value we want to obtain. Coeurjolly et al. [CLR12] list many of the convergent estimators for length that can be used instead.

2.4.2 Differential geometry on digital surfaces

Digital surfaces make a great example for meshes whose positions converge in Hausdorff distance but whose normals do not converge to the one of the underlying surface (normals being axis aligned, they only have 6 possible values). Such examples can already be found in the realm of triangular surfaces with cases such as the Schwarz lantern, but those tend to be dismissed as edge cases and not be discussed. In our case, convergent normals are the edge case, so we have to take this non convergence into consideration. Notably, works on the convergence of the cotan formula stipulate that, under the assumption that the positions converge, the Laplace operator weakly converges if and only if the normal vectors converge [War08]. While we will not use the cotan formula (we could triangulate our digital surface, but keeping quads gives us more freedom over the scheme used), we can safely assume that an operator built using a naive approach will not converge either. Thus, the following work leverage the fact that we can achieve convergent normal estimators to add "normal correction" to the schemes available to build a Laplace-Beltrami operator.

Several of such estimators exists, with convergence guarantees. A naive approach that would use an average of values around a face does not suffice, but more sophisticated approaches such as Generalized Voronoi Covariance Measure of Cuel et al. [Cue+15] and Integral Invariant of Coeurjolly et al. [CLL14; LCL17] do. These approaches also yield estimators for curvature and areas.

While digital surfaces are made of squares, the methods we will explore also include general polygonal methods. One of the reason for this interest is that future works may include digital surfaces with varying grid resolution, where faces at the interface between two different resolutions could have an additional vertex inside an edge (figure 2.22).

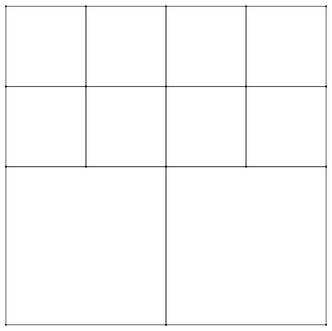


FIGURE 2.22: 2d grid with varying resolution. At the frontier, some quads actually have 5 vertices: these cannot be treated as quads when building a basis function.

2.4.3 Examples

We present here examples of how to solve simple problems using the Laplace-Beltrami operator. We also show how standard operator construction fail when used on digital surfaces. The method used for naive discretization here is the one from de Goes et al. [DBD20], and the corrected ones

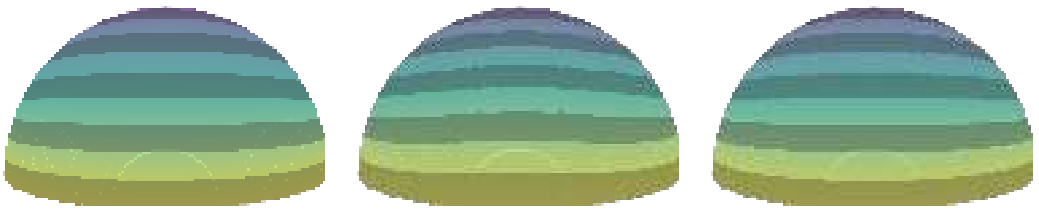


FIGURE 2.23: Laplace problem on a half sphere, discretized with gridstep $h = 0.01$ (95445 surfels). We start from a linear function: left picture is both the integrated function and the expected result, as this is a spherical harmonic. Middle is the result obtain with a non corrected Laplacian: there is small visible distorsion, as well as global distorsion (isolines are not parallel to the border of the half sphere). On the right, results are obtained with a corrected operator: while there is still some small distorsion, isolines are parallel to the border.

Poisson Problem

We want to solve a Poisson problem $\Delta f = g$. We study here the following problem: on a half sphere of equation $x^2 + y^2 + z^2 = 1, y \geq 0$, find f such that $\Delta f = y$ and $f(x, 0, z) = 0$ (boundary condition). This problem has the advantage of having a known solution: linear functions on the sphere are, in fact, the first non constant harmonics, so what we described here is a part of a spherical harmonic. Thus f should be the same as g up to a global multiplicative constant. Figure 2.23 shows that the result are visibly improved by the usage of a corrected operator.

We repeat the experiment with $\Delta f = \frac{1}{2}(3y^2 - 1)$, which is again a spherical harmonic, but of higher frequency. We see on figure 2.24 that again, using a corrected operator visibly improves the quality of the result. This is more visible than with the previous function, suggesting that the correction has an effect in high frequencies.

Boundary conditions We assume here that our domain has a boundary. If our domain does not have one, a value can be chosen at one or several points, which can "become" the boundary. In any case, we need some kind of boundary condition if we want our system to be well defined and to have a unique solution: in general, $\Delta u = g$ does not have a unique solution, since Δ is a linear operator and is not injective. In our matrix form, the fact that we need boundary conditions is seen as the fact that L is PSD instead of positive definite. Thus, if we cannot invert it properly, we require additional constraints. We do not necessarily require full boundary constraints: we could try to only set values for a few points. Stein et al. [Ste+20] explain why this leads to natural Neumann boundary conditions, and thus why just using the Dirichlet Energy as a smoothness energy is not always the best choice.

Heat Geodesics

We use the heat method for geodesic approximation of [CWW17]. Recall the method:

- 1: integrate the heat flow $\frac{\partial u}{\partial t} = \Delta u$ for some fixed time t
- 2: evaluate the vector field $X = -\nabla u / |\nabla u|$
- 3: solve the Poisson equation $\Delta \Phi = \nabla . X$

We thus need a Laplace-Beltrami operator, as well as a gradient and a divergence. Figure 2.25 shows the difference between using a naive construction and a corrected one.

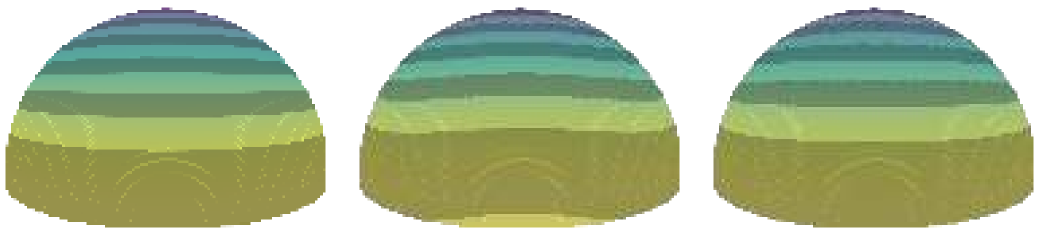


FIGURE 2.24: Laplace problem on a half sphere, discretized with gridstep $h = 0.01$ (95445 surfels). We use a higher spherical harmonic than figure 2.23. Again, left is both the integrated function and the expected result, middle is obtained without correction and right is with. The distorsion is visible in both case, however the shape of the corrected function is visibly better than the non corrected one.

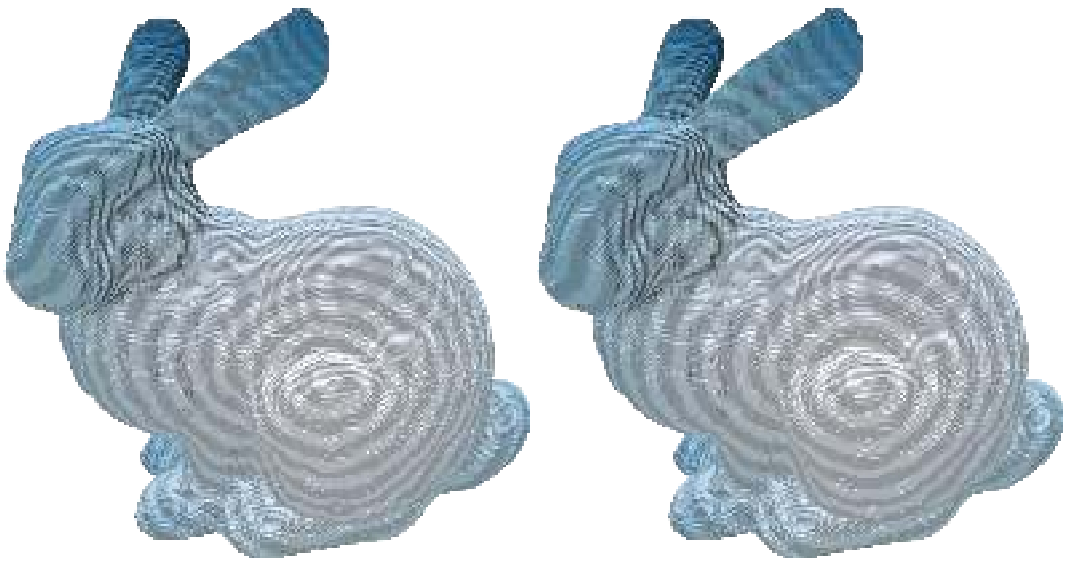


FIGURE 2.25: Left: distances obtained using a naive method, and right using a corrected one. The naive method has an elliptic shape around the center, suggesting that the nature of the surface induces some anisotropy in the distances computations. The corrected method removes this effect.

2.5 Conclusion

We have seen that there are numerous ways to discretize the Laplace-Beltrami operator on triangle meshes and general polygonal surfaces. However, they fail when used on digital surfaces, whose normals often do not converge to those of the original surface. In the next chapter, we see how these discretization schemes can be adapted for digital surfaces using a corrected normal field.

CHAPTER

Corrected Laplacians 3

Résumé

Nous présentons ici trois adaptations de méthodes existantes pour construction d'opérateurs sur des surfaces polygonales à des surfaces digitale, via une correction basée sur les normales. Il existe déjà des méthodes pour obtenir ces opérateurs sur les surfaces digitales, mais résultent soit en des estimateurs denses (et donc plus lourd à manipuler), soit ne présentent pas de garanties théoriques de convergences.

Nos méthodes se basent sur des notions d'espace tangents corrigés. Grâce à un vecteur normal $\mathbf{u}$ qui ne correspond pas forcément à la normale obtenue naïvement à partir de la surface digitale, on peut par exemple corriger la longueur d'un vecteur $\mathbf{v}$ en utilisant non pas la norme de $\mathbf{v}$ mais $||\mathbf{v} \times \mathbf{u}||$. De la même manière, on peut corriger l'aire d'un parallélogramme défini par les vecteurs $\mathbf{v}$ et $\mathbf{w}$ en prenant $\det(\mathbf{u}, \mathbf{v}, \mathbf{w})$. Nous présentons ainsi trois méthodes corrigées:

- **Calcul interpolé corrigé** Méthode inspirée de [CL22], mais basée sur un champ de normales interpolées aux sommets plutôt qu'un champs de normales constant par face (et donc discontinu).
- **Éléments finis corrigés** Méthode inspirée des éléments finis. C'est une méthode simple et étudiée, et nous esperons ainsi que dans le futur des preuves de convergence de cette méthode puissent être adaptée pour obtenir des garanties de convergence sur une méthode corrigée.
- **Éléments virtuels raffinés corrigés** Méthode inspirée des méthodes par éléments virtuels raffinés [Bun+24; Bun+20].

Nous montrons ainsi que l'on peut adapter des schémas existants en schémas pour les surfaces digitales. On vérifie ensuite experimentalement sur la sphère que les résultats que l'on obtient semblent bien converger lorsque $h \to 0$. Nous n'avons pas obtenu de garanties théoriques de convergence, mais nous esperons pouvoir reprendre les preuves existantes sur les méthodes d'éléments finis et les adapter à notre méthode d'éléments finis corrigés.

In the previous section, we have seen that naive discretizations of the Laplace-Beltrami operator did not yield good results on digital surfaces. We introduce here the idea of *corrected* methods: supposing that we have additional informations on the surface such as *corrected* normals, we build differential operators taking into account these additional informations. We briefly recall previous methods using this correction, and we propose three novel normal corrected approaches: one DEC based using interpolation from normals on vertices, a FEM based one using a constant corrected normal per face, and a virtual refinement based one again with normal at vertices. We also provide experiments to test the convergence of these corrected operators, and these experiments suggest that these operators are indeed convergent when solving Poisson problems, and that adding some form of diffusion can make them convergent when computing the Laplacian of a function.

The DEC and FEM based method were presented at DGMM2024 (Discrete Geometry and Mathematical Morphology) [WCL24]. The virtual refinement was added in a submission for a special issue in JMIV (Journal of Mathematical Imaging and Vision).

3.1 Existing methods using corrected geometry

We briefly recall recent methods that used corrected normals to build differential operators on digital surfaces.

3.1.1 Heat Kernel Laplace-Beltrami

The method by [Cai+19] proposes a construction of the Laplace operator based on heat kernels. This requires evaluation of geodesics distance, wich is done by using corrected distances on the corrected tangent planes (using provided normals). This operator is proven to be strongly convergent, however it comes with several downside : it does not produce two sparse matrices L and M but directly a dense matrix $\mathbb{L}$. The evaluation of geodesics distances takes quite some time, and storing the dense matrix takes a lot of memory : in fact, in the experiment section, we do not evaluate results below a certain gridstep h since the matrix does not fit into memory (64 Gb) anymore. We have not used it for solving Poisson problems either, due to the complexity of inverting dense matrices.

3.1.2 Projected PolyDEC

The method [CL22] is built on [DBD20]: corrected normals are used when the construction requires the normal of each face, and when lengths are needed when building metric dependant operators (the hodge star, the sharp and the flat). Instead of taking the naive length of the edge, the length of the edge projected into the corrected tangent space is taken. The paper also evaluates the operator on various problems, such as geodesics using heat diffusion and spherical harmonics, where it performs better than non corrected methods. One of the drawback of this approach is that we assume corrected normals to be constant per face, which suggests that the corrected normal field is not continuous. However, in our work we test a method which uses vertex interpolated normals instead of a constant per face method, and only slightly outperforms the method from [CL22]. This suggests that normal discontinuity does not pose strong issue for that kind of problem. Note that since this method follows the same DEC approach as [DBD20], it provides not only the Laplace-Beltrami operator but other differential operators as well.

3.2 Digital calculus with corrected tangent space

We demonstrate here that the same approach of corrected lengths and areas used in [CL22] can be used to build differential operators with other schemes. 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 denote by $\mathbf{n}$ the natural or naive normal of a surfel σ, that can be computed with a cross product of two consequent edges. The corrected normal field will be denoted $\mathbf{u}$. Inside a surfel, we can decompose this surfel in the natural base of the surfel into $\mathbf{u} = (u^x, u^y, u^z)$. 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 [Vei+12] requires solely the behavior of functions on edges). The methods we present, similarly to [CL22], use a per face construction of sparse operators.

3.2.1 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 [DBD20], building inner products, sharp and flat operators on a per face basis. However the correction of the geometry does not follow [CL22], 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 [Lac+20; LRT22]): $\omega_0^{\mathbf{u}}(\mathbf{x})(\mathbf{v}, \mathbf{w}) := \det(\mathbf{u}(\mathbf{x}), \mathbf{v}, \mathbf{w})$, 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. $\square := [0,1]^2$A function g in a surfel σ is assumed to be bilinearly interpolated. We denote then by $[f_{\boxdot}(\sigma)] := [f_{00}(\sigma), f_{10}(\sigma), f_{11}(\sigma), f_{01}\sigma)]^\intercal$ the degrees of freedom of f, corresponding to its values at each vertex when circulating around σ. We will often write simply $[f_{\boxdot}]$ when the surfel is obvious from the context. We sometimes use averages of these values, whose notations are illustrated in Figure 3.1. In the case of a surfel σ, corresponding to the domain $\square := [0,1]^2$ parameterized by s,t, 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

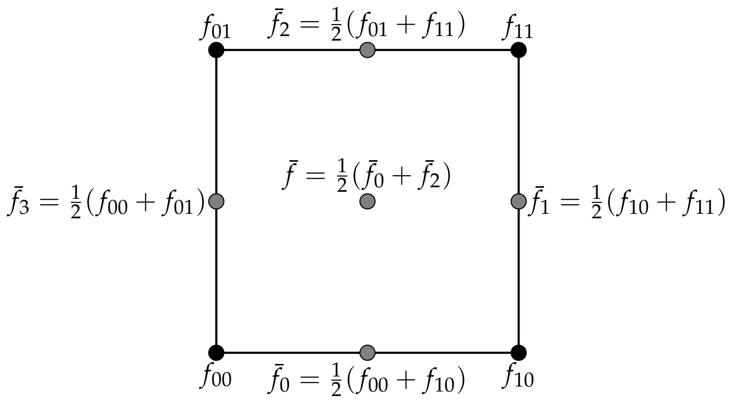

f_{01}
$\bar{f}_2 = \frac{1}{2}(f_{01} + f_{11})$
f_{11}
$\bar{f} = \frac{1}{2}(\bar{f}_0 + \bar{f}_2)$
$\bar{f}_3 = \frac{1}{2}(f_{00} + f_{01})$
$\bar{f}_1 = \frac{1}{2}(f_{10} + f_{11})$
f_{00}
$\bar{f}_0 = \frac{1}{2}(f_{00} + f_{10})$
f_{10}

FIGURE 3.1: Notations for the interpolations of function f values on a surfel.

form reduced on the surfel to $\omega_0^{\mathbf{u}}(s,t) = \langle \mathbf{n} \mid \mathbf{u}(s,t) \rangle = u^z(s,t)$. We can now compute the integral of g inside a surfel:

$$\iint_{\square} g\omega_0^{\mathbf{u}} := \iint_{\square} g(s,t)\mathbf{u}^z(s,t)dsdt = \left[\mathbf{u}^z_{\boxtimes}\right]^{\intercal} \frac{1}{36} \begin{bmatrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{bmatrix} \left[g_{\boxtimes}\right] .$$

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 Φ will generally be defined as the bilinear interpolation of a scalar field ϕ defined over the domain. Thus $\phi(s,t) = \Phi(\mathbf{x}(s,t))$. We relate the gradient of Φ with the partial derivatives of ϕ by writing the standard chain rule with Jacobian matrices:

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

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

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

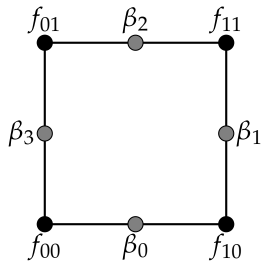


FIGURE 3.2: Notations for the values of a 1-form β on a surfel.

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

$$\begin{bmatrix} (1-t)(\phi_{10}-\phi_{00}) + t(\phi_{11}-\phi_{01}) \\ (1-s)(\phi_{01}-\phi_{00}) + s(\phi_{11}-\phi_{10}) \\ 0 \end{bmatrix} = \underbrace{\begin{bmatrix} 1-t & 0 & -t & 0 \\ 0 & s & 0 & s-1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix}}_{D_0} \left[\phi_{\boxdot}\right].$$

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

$$\iint_\square \nabla\Phi\omega_0^{\mathbf{u}} = \underbrace{\iint_\square CBu^z dsdt}_{\mathscr{G}_\sigma} D_0 \left[\phi_{\boxdot}\right].$$

Let us introduce some weighted averages between vectors $\mathbf{u}$ at the 4 vertices.

$$\bar{\mathbf{u}} := \frac{1}{4}(\mathbf{u}_{00} + \mathbf{u}_{10} + \mathbf{u}_{11} + \mathbf{u}_{01})$$

$$\bar{\mathbf{u}}_0 := \frac{1}{2}(\mathbf{u}_{00} + \mathbf{u}_{10}) \qquad \bar{\mathbf{u}}_2 := \frac{1}{2}(\mathbf{u}_{11} + \mathbf{u}_{01})$$

$$\bar{\mathbf{u}}_1 := \frac{1}{2}(\mathbf{u}_{10} + \mathbf{u}_{11}) \qquad \bar{\mathbf{u}}_3 := \frac{1}{2}(\mathbf{u}_{01} + \mathbf{u}_{00})$$

In this case, the integrated gradient matrix $\mathscr{G}_\sigma$ matrix is a 3×4 matrix such that:

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

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

$$a_\sigma := \iint_\square \omega_0^{\mathbf{u}} = \iint_\square u^z dsdt = \bar{u}^z, \qquad \mathbf{G}_\sigma := \frac{1}{a_\sigma}\mathscr{G}_\sigma D_0.$$

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 β be 1-form, and $\left[\beta_{\diamond}(\sigma)\right] := \begin{bmatrix} \beta_0 & \beta_1 & \beta_2 & \beta_3 \end{bmatrix}^\mathsf{T}$ its values on the 4 edges of σ (figure 3.2). Omitting the differential operator D_0 in the pointwise gradient gives the representative 3D vector of β on surfel σ:

$$\beta^\sharp(\sigma) := \frac{1}{a_\sigma}\mathscr{G}_\sigma\left[\beta_{\diamond}(\sigma)\right].$$

The discrete sharp operator on σ is thus the 3×4 matrix:

$$U_\sigma := \frac{1}{a_\sigma}\mathscr{G}_\sigma \tag{3.1}$$

The flat operator projects a vector field onto the tangent plane and computes its circulation along each edge. The 1-form $\mathbf{v}^\flat$ associated with vector $\mathbf{v}$ is thus:

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

where $\mathbf{t}$ represents the tangent vector to the face at the border, and $I - \mathbf{u}\mathbf{u}^\top$ the projection onto the tangent plane of the face. By linearity, the flat operator V_σ is a 4×3 matrix:

$$V_\sigma := \frac{1}{6}\begin{bmatrix} 6 - 2((u^x_{00})^2 + u^x_{00}u^x_{10} + (u^x_{10})^2) & -(2u^x_{00} + u^x_{10})u^y_{00} - (u^x_{00} + 2u^x_{10})u^y_{10} & -(2u^x_{00} + u^x_{10})u^z_{00} - (u^x_{00} + 2u^x_{10})u^z_{10}) \\ -(2u^x_{10} + u^x_{11})u^y_{10} - (u^x_{10} + 2u^x_{11})u^y_{11} & 6 - 2((u^y_{10})^2 + u^y_{10}u^y_{11} + (u^y_{11})^2) & (2u^y_{10} + u^y_{11})u^z_{10} - (u^y_{10} + 2u^y_{11})u^z_{11} \\ 2((u^x_{01})^2 + u^x_{01}u^x_{11} + (u^x_{11})^2) - 6 & (2u^x_{01} + u^x_{11})u^y_{01} + (u^x_{01} + 2u^x_{11})u^y_{11} & (2u^x_{01} + u^x_{11})u^z_{01} + (u^x_{01} + 2u^x_{11})u^z_{11} \\ (2u^x_{00} + u^x_{01})u^y_{00} + (u^x_{00} + 2u^x_{01})u^y_{01} & 2((u^y_{00})^2 + u^y_{00}u^y_{01} + (u^y_{01})^2) - 6 & (2u^y_{00} + u^y_{01})u^z_{00} + (u^y_{00} + u^y_{01})u^z_{01} \end{bmatrix}$$

Proposition 1. *If* $\mathbf{u}$ *is a unit constant vector over the surfel* f*, then both products* $V_\sigma U_\sigma$ *and* $U_\sigma V_\sigma$ *have rank 2 and satisfy respectively*

$$V_\sigma U_\sigma = \frac{1}{2}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 - 1 & \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}, \tag{3.2}$$

$$U_\sigma V_\sigma = (I - \mathbf{u}\mathbf{u}^\top). \tag{3.3}$$

Proof. For $\mathbf{u}$ constant on the face, both U_σ and V_σ take a simpler form. Using $U_\sigma := \frac{1}{a_f}\mathscr{G}_f$, we get

$$U_\sigma = \frac{1}{2\mathbf{u}^z}\begin{bmatrix} u^z & 0 & -u^z & 0 \\ 0 & u^z & 0 & -u^z \\ -u^x & -u^y & u^x & u^y \end{bmatrix} \quad V_\sigma = \begin{bmatrix} 1 - (u^x)^2 & -u^x u^y & -u^x u^z \\ -u^x u^y & 1 - (u^y)^2 & -u^y u^z \\ (u^x)^2 - 1 & u^x u^y & u^x u^z \\ u^x u^y & (u^y)^2 - 1 & u^y u^z \end{bmatrix}.$$

Product $V_\sigma U_\sigma$ is immediate through computation. For $U_\sigma V_\sigma$, the simplification for the last row requires $\mathbf{u}$ to be unitary, which entails the relation $1 - (\mathbf{u}^x)^2 - (\mathbf{u}^y)^2 = (\mathbf{u}^z)^2$. The rank follows immediately from the matrix forms. $\square$

Corollary 1. *If* $\mathbf{v}$ *is a vector,* $(\mathbf{v}^\flat)^\sharp = \mathbf{v} - \langle \mathbf{v} \mid \mathbf{u} \rangle \, \mathbf{u}$*. In particular, if* $\mathbf{v}$ *is a vector orthogonal to* $\mathbf{u}$*,* $(\mathbf{v}^\flat)^\sharp = \mathbf{v}$*.*

It follows that flat and sharp operators are consistent with the *corrected normal* $\mathbf{u}$ of the face, which defines its "true" tangent plane.

Inner products for discrete forms (i.e metrics). The inner product between 0-forms is simply the integration of their product on the surfel σ. For any bilinearly interpolated functions ϕ, ψ, we obtain on the surfel σ the scalar:

$$\langle \phi \mid \psi \rangle_0 (\sigma) := \iint_\square \phi \psi \omega_0^{\mathbf{u}} = \left[\phi_{\boxdot}(\sigma)\right]^\mathsf{T} M_{0,\sigma} \left[\psi_{\boxdot}(\sigma)\right].$$

The associated metric matrix is a 4×4 symmetric matrix, called *mass matrix*. 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 σ 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

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

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

$$M_{0,\sigma} = \frac{1}{144} \begin{bmatrix} \bar{u}^z_{00} & \bar{u}^z_{00,10} & 4\bar{u}^z & \bar{u}^z_{01,00} \\ \bar{u}^z_{00,10} & \bar{u}^z_{10} & \bar{u}^z_{10,11} & 4\bar{u}^z \\ 4\bar{u}^z & \bar{u}^z_{10,11} & \bar{u}^z_{11} & \bar{u}^z_{11,01} \\ \bar{u}^z_{11,01} & 4\bar{u}^z & \bar{u}^z_{11,01} & \bar{u}^z_{01} \end{bmatrix}.$$

If the corrected normal vector $\mathbf{u}$ is consistent with the naive surfel normal $\mathbf{n}$ (i.e. $\langle \mathbf{u}(s,t) \mid \mathbf{n} \rangle > 0$), then $M_{0,\sigma}$ is positive definite.

We would like the inner product between 1-forms β and γ to be defined by emulating the continuous case. We integrate the scalar product between the vectors associated with the 1-forms on the surfel σ:

$$\langle \beta \mid \gamma \rangle_1 (\sigma) := \iint_\square \left\langle \beta^\sharp \mid \gamma^\sharp \right\rangle \omega_0^{\mathbf{u}} = \left[\beta_{\boxplus}(\sigma)\right]^\mathsf{T} M^{\text{naive}}_{1,\sigma} \left[\gamma_{\boxplus}(\sigma)\right]. \tag{3.4}$$

Using above relations (equations 3.4 and 3.1) we have:

$$\langle \beta \mid \gamma \rangle_1 (\sigma) = a_\sigma (U_\sigma \left[\beta_{\boxplus}(\sigma)\right])^\mathsf{T} (U_\sigma \left[\gamma_{\boxplus}(\sigma)\right] \gamma) = \left[\beta_{\boxplus}(\sigma)\right]^\mathsf{T} \left(\frac{1}{a_\sigma} \mathscr{G}_\sigma^\mathsf{T} \mathscr{G}_\sigma \right) \left[\gamma_{\boxplus}(\sigma)\right].$$

Hence $M^{\text{naive}}_{1,\sigma} = \frac{1}{a_\sigma} \mathscr{G}_\sigma^\mathsf{T} \mathscr{G}_\sigma$; it is a symmetric matrix. If $\mathbf{u}$ is constant over the surfel f with naive normal $\mathbf{n} = (0,0,1)$, it takes the form:

$$M_1 = \frac{1}{9\mathbf{u}^z} \begin{bmatrix} (u^x)^2 + (u^z)^2 & u^x u^y & -(u^x)^2 - (u^z)^2 & -u^x u^y \\ u^x u^y & (u^y)^2 + (u^z)^2 & -u^x u^y & -(u^y)^2 - (u^z)^2 \\ -(u^x)^2 - (u^z)^2 & -u^x u^y & (u^x)^2 + (u^z)^2 & u^x u^y \\ -u^x u^y & -(u^y)^2 - (u^z)^2 & u^x u^y & (u^y)^2 + (u^z)^2 \end{bmatrix}$$

Proposition 2. *If* $\mathbf{u}$ *is a unit constant vector over the surfel* f *and consistent with its naive normal* $\mathbf{n}$ *(i.e.* $\langle \mathbf{u} \mid \mathbf{n} \rangle > 0$*), then the matrix* M_1 *is symmetric positive.*

Proof. Let us assume surfel f has naive normal $\mathbf{n} = (0,0,1)$. Let $\beta_f = \begin{bmatrix} x & y & z & t \end{bmatrix}^\top$ be an arbitrary 1-form defined on f. Then we compute

$$\begin{aligned}\beta_f^\top M_1 \beta_f =& ((u^x)^2 + (u^z)^2)(x^2 - 2xz + z^2) + ((u^y)^2 + (u^z)^2)(y^2 - 2yt + t^2) \\ & + 2u^x u^y (xy - xt - yz + zt) \\ =& (1 - (u^y)^2)(x - z)^2 + (1 - (u^x)^2)(y - t)^2 + 2u^x u^y (x - z)(y - t),\end{aligned}$$

using $\|\mathbf{u}\| = 1$. Denoting $r = x - z$, $s = y - t$, we get:

$$\begin{aligned}\beta_f^\top M_1 \beta_f =& (1 - (u^y)^2) r^2 + (1 - (u^x)^2) s^2 + 2u^x u^y rs \\ \geq& (1 - (u^y)^2) r^2 + (1 - (u^x)^2) s^2 - 2|u^x||r||u^y||s|\end{aligned}$$

We have $|u^x|^2 + |u^y|^2 < 1$. Setting $\epsilon = |u^x|$ leads to $|u^y| < \sqrt{1 - \epsilon^2}$. Hence

$$\begin{aligned}\beta_f^\top M_1 \beta_f \geq& \epsilon^2 r^2 + (1 - \epsilon^2) s^2 - 2\epsilon |r| \sqrt{1 - \epsilon^2} |s| \\ \geq& (\epsilon |r| - \sqrt{1 - \epsilon^2} |s|)^2 \geq 0.\end{aligned}$$

It might be null only if both $r = s = 0$, hence $x = z$ and $y = t$. □

However, it is not definite. To remedy this, we follow [DBD20] and complement the definition to get the *stiffness matrix* as

$$M_{1,\sigma} := \frac{1}{a_\sigma} \mathcal{G}_\sigma^\intercal \mathcal{G}_\sigma + \lambda (I - U_\sigma V_\sigma). \tag{3.5}$$

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* $\delta_1 : \mathcal{E} \to \mathcal{V}$ by adjointness in our inner products.

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

Being true for all pairs (f, α), it follows that $\delta_1 := -{M_0}^{-1} D_0^\intercal M_1$. The *Laplacian operator* $\mathbf{\Delta}_0$ is the composition of the codifferential and the differential, i.e.

$$\mathbf{\Delta}_0 := \delta_1 D_0 = -{M_0}^{-1} D_0^\intercal M_1 D_0.$$

Since it is very costly to build matrix ${M_0}^{-1}$, we will generally not use the two operators δ_1 and $\mathbf{\Delta}_0$ as is when solving numerical problems, but we will rather work with their "integrated" version ($M_0 \delta_1$ and $M_0 \mathbf{\Delta}_0$). We can now see another approach to compute a Laplace-Beltrami operator coming from the Finite Elements framework.

3.2.2 Generalization to Finite Element Method

We show here how to adapt the standard Finite Element Method (FEM) in order to solve a Poisson problem. Our adaptation consist in correcting the metric, 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.

We follow the same steps as in section 2.2.1. The Poisson problem is formulated as solving for g in $\Delta g = f$, with a given border 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 g

$$\int_\Omega \nabla g . \nabla \Phi = - \int_\Omega f\Phi + \int_{\partial\Omega} \Phi \langle \nabla g, \mathbf{n} \rangle \,, \tag{3.6}$$

for any Φ. In our case, we will evaluate against Φ the functions locally bilinear inside each element. The third term is dependent on the boundary condition, and we will make it vanish here for now (we can assume that our surface has no boundary for example).

The FEM approach consists in discretizating 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 equation 3.6. 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 Laplace-Beltrami operator $\mathbb{L} := M^{-1}L$.

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:

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

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 $\mathrm{d}f(\mathbf{w}) = \langle \nabla f, \mathbf{w} \rangle_G$ for any vector $\mathbf{w}$, with $\langle \cdot, \cdot \rangle_G$ the inner product. It follows that $\nabla f = G^{-1} \begin{bmatrix} \frac{\partial f}{\partial s} & \frac{\partial f}{\partial t} \end{bmatrix}^\top$. For the Laplacian, since the metric is constant, we use $\Delta f = \nabla \cdot \nabla f$. We write them more explicitly as:

$$\nabla f = \frac{1}{(u^z)^2} \begin{bmatrix} (1-u_y^2)\frac{\partial f}{\partial s} + u^x u^y \frac{\partial f}{\partial t} \\ u^x u^y \frac{\partial f}{\partial s} + (1-(u^x)^2)\frac{\partial f}{\partial t} \end{bmatrix} \tag{3.7}$$

$$\Delta f = \frac{1}{(u^z)^2} \left((1-(u^y)^2)\frac{\partial^2 f}{\partial s^2} + 2u^x u^y \frac{\partial^2 f}{\partial s \partial t} + (1-(u^x)^2)\frac{\partial^2 f}{\partial t^2} \right) \tag{3.8}$$

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 (3.8), 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 (so much that finite element simulation is a motivation for quad meshing [Bom+13]) 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 f and p bilinear. The local stiffness matrix is then:

$$L_M = \frac{1}{6u^z} \begin{bmatrix} 3u^x u^y + 2 + 2(u^z)^2 & 2(u^y)^2 - 1 - (u^x)^2 & 1 - 3u^x u^y - (u^z)^2 & 2(u^x)^2 - 1 - (u^y)^2 \\ 2(u^y)^2 - 1 - (u^x)^2 & 2 - 3u^x u^y + 2(u^z)^2 & 2(u^x)^2 - 1) - (u^y)^2 & 3u^x u^y + 1 - (u^z)^2 \\ 2 - 3u^x u^y - (u^z)^2 & 2(u^x)^2 - 1 - (u^y)^2 & 3u^x u^y + 2 + 2(u^z)^2 & 2(u^y)^2 - 1 - (u^x)^2 \\ 2(u^x)^2 - 1 - (u^y)^2 & 3u^x u^y + 1 + (u^z)^2 & 2(u^y)^2 - 1 - (u^x)^2 & 2 - 3u^x u^y + 2(u^z)^2 \end{bmatrix} . \tag{3.9}$$

The global stiffness matrix is then obtained by summing over all the local stiffness matrices. The mass matrix is computed from $\int_\Omega fp$, with f and p bilinear:

$$M_M = \frac{u^z}{36} \begin{bmatrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{bmatrix} . \tag{3.10}$$

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 $\mathbf{u}$.

3.2.3 Generalization to Virtual Refinement Method

Here we show how we can adapt the method from Bunge et al. [Bun+20] 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 ssquared areas of the triangles $A_0^2 + A_1^2 + A_2^2 + A_3^2$ (figure 3.3). 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 $u^z(s,t)$ on a surfel:

$$\begin{aligned} A_0 = & \int_0^{s_0} \int_0^{\frac{s}{s_0} t_0} u^z(s,t) t dt ds \\ & + \int_{s_0}^{1} \int_0^{\frac{1-s}{1-s_0} t_0} u^z(s,t) t dt ds. \end{aligned} \tag{3.11}$$

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.

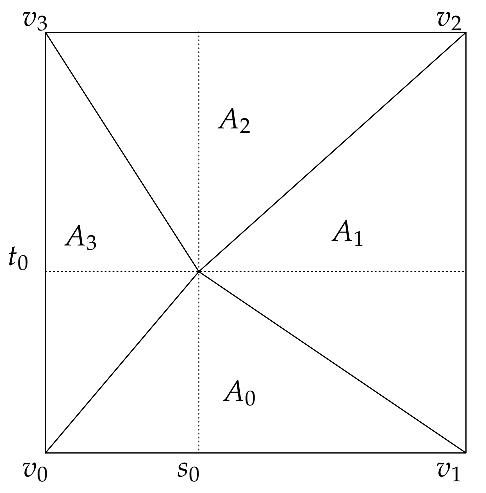


FIGURE 3.3: We find s_0 and t_0 such that the sum of the squared areas $A_0^2 + A_1^2 + A_2^2 + A_3^2$ is minimized.

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 prolongation operator then needs to assign values to vertices and the virtual vertices from the vertices: for a single face, it would be a 5×4 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 the the row. The following n_v rows correspond to the identity matrix.

For the mass matrix, we simply apply the area formula of (3.11). 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 $M_\triangle$ and $L_\triangle$ on the triangulation, using our prolongation operator we compute $M = P^\top M_\triangle P$ and $L = P^\top L_\triangle P$. We now have two ways of building a sparse Laplace-Beltrami operator. We recapitulate the available methods available to build the Laplace-Beltrami operator on digital surfaces (figure 3.4). We see now how they compare against operators from previous works.

Method	scheme	basis functions	matrix shape	corrected normals **u** placement
Heat Kernel [Cai+19]	-	-	dense	faces
Projected PolyDEC [CL22]	DEC	(implicit)	sparse	faces
Corrected FEM	FEM	bilinear	sparse	faces
Corrected Calculus	DEC	bilinear	sparse	vertices
Corrected VRM	virtual FEM	linear	sparse	vertices

FIGURE 3.4: Available methods to build the Laplace-Beltrami operator on digital surfaces.

3.3 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 to get 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.

3.3.1 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 Δf both analytically and with a discrete Laplacian, then compare the two results. In other words, if our stiffness matrix 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 3.5). This is disappointing since we expect the Laplacian operator to have linear convergence when evaluated in forward manner, as observed in [Cai+19] and proven for the mesh Laplacian 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 with a characteristic length of order $h^{\frac{1}{2}}$, which is coherent with the charactristic length used in [HP11]. Using this method, we achieve what seems to be linear convergence on different functions (Figure 3.5), with results comparable or even higher quality than in [Cai+19]. We run the same experiment as figure 3.5, with evaluated normals (using Integral Invariant [LCL17]) instead of ground truth. Results are shown in figure 3.6, and also approach linear convergence.

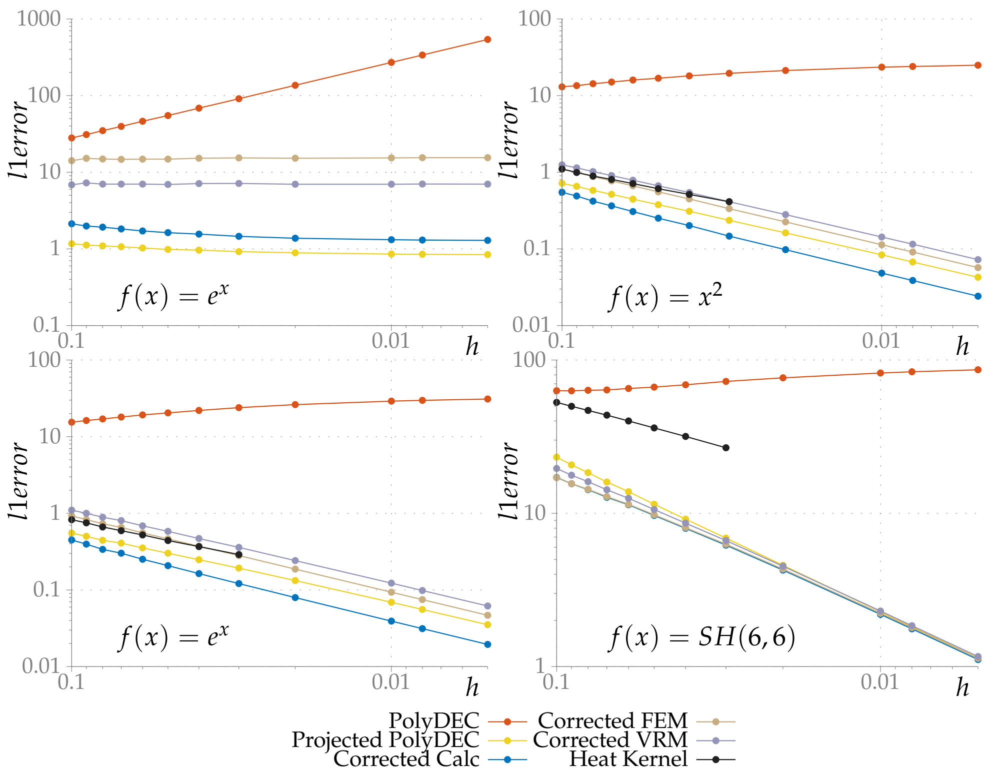


FIGURE 3.5: Forward evaluation of the mesh Laplacian on quadratic, exponential functions and the sixth spherical harmonic. PolyDEC corresponds to [DBD20], Projected PolyDEC to [CL22] and Heat Kernel to [Cai+19]. Top left doesn't have diffusion, other have: adding diffusion significantly improves the results, achieving linear convergence on the sphere. We achieve similar rates of convergence as Heat Kernel, with a better quality on less smooth functions. Note that the Heat Kernel method is limited to a gridstep of $h \geq 0.03$, due to its enormous memory usage.

3.3.2 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 [BB23]), which has not yet been done for digital Laplacians. It also makes more sense 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 3.7). 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 for **u** and approach a similar rate of convergence. (figure 3.8)

3.3.3 Eigenvalues

We follow the evaluation of eigenvalues on the spherical harmonics used in [BB23]. 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 λ in the following generalized

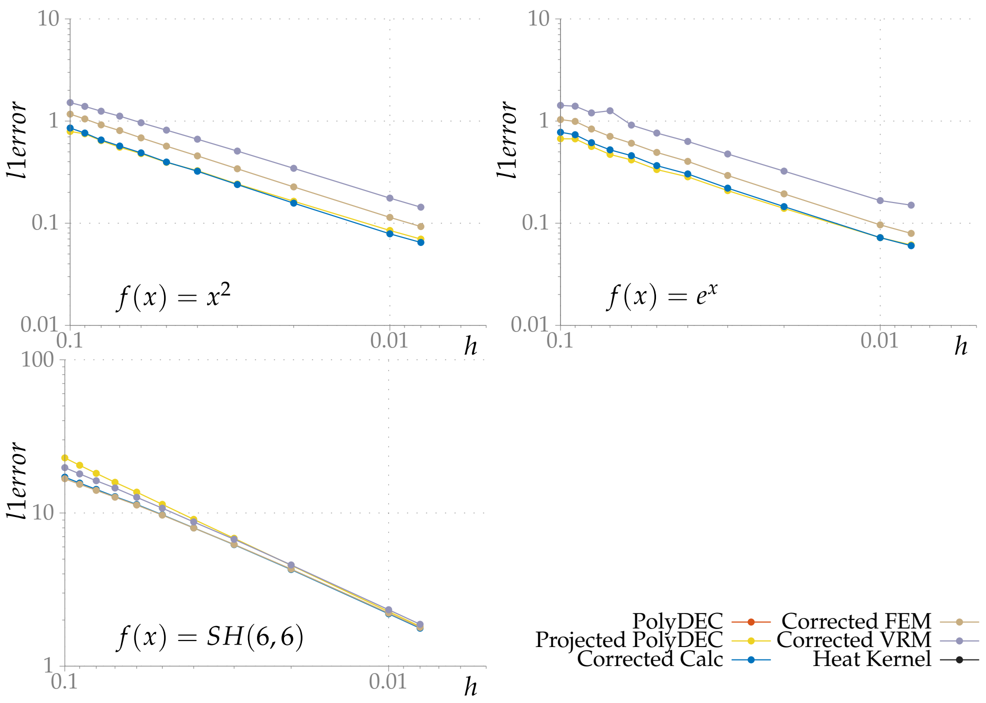


FIGURE 3.6: Evaluation of the Laplacian using integral invariant estimators for the corrected normal vector field **u**. Estimated normals give slightly worse results than with true normals, yet the Laplacian seem to converge too. Computations are performed for grid steps $h \geq 0.008$ because integral invariant normal estimator is very slow on finer grids.

eigenvalue problem $L\mathbf{u} = \lambda M\mathbf{u}$. Figure 3.9 shows the first eigenvalues of our Laplacians on the unit sphere with discretization steps $h = 0.1$ and $h = 0.01$. The PolyDEC method [DBD20] is not accurate, but corrected methods are, with accuracy increased at finer resolutions.

3.3.4 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 build 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 order to use these normals as **u** in the calculus schemes. We then compute a function on the triangle mesh as well as its Laplacian using the cotan Laplacian on the triangle mesh [LZ10], and then their projection on the digital surface, which we use as "ground truth". Forward evaluation results are shown on figure 3.10. 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.

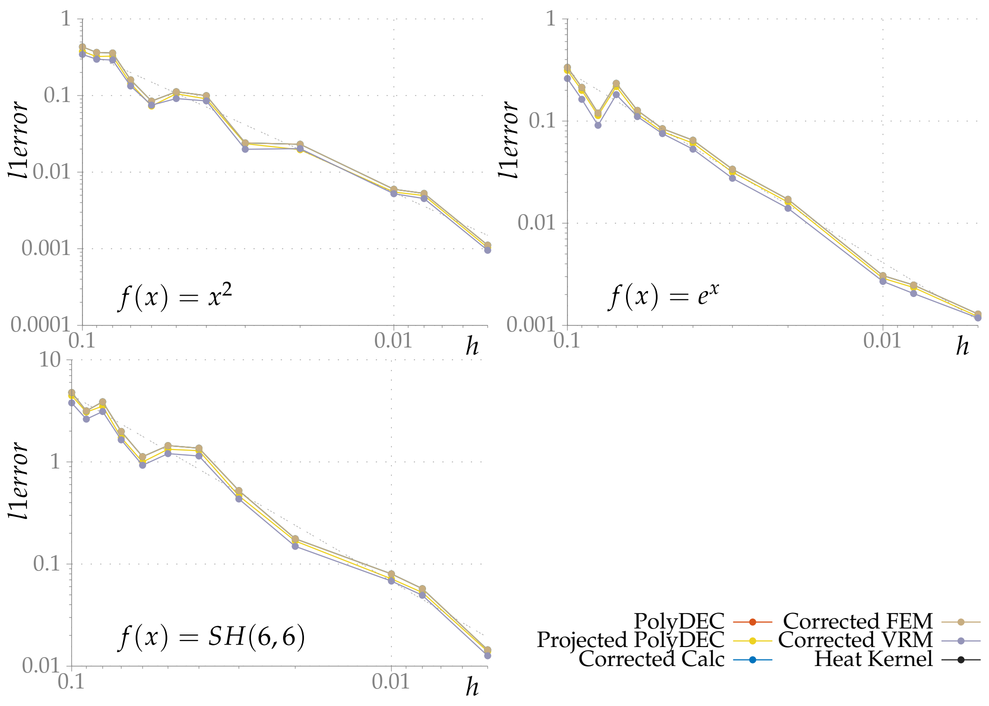


FIGURE 3.7: Error when solving a Poisson problem with different Laplacians. We approach a quadratic convergence rate.

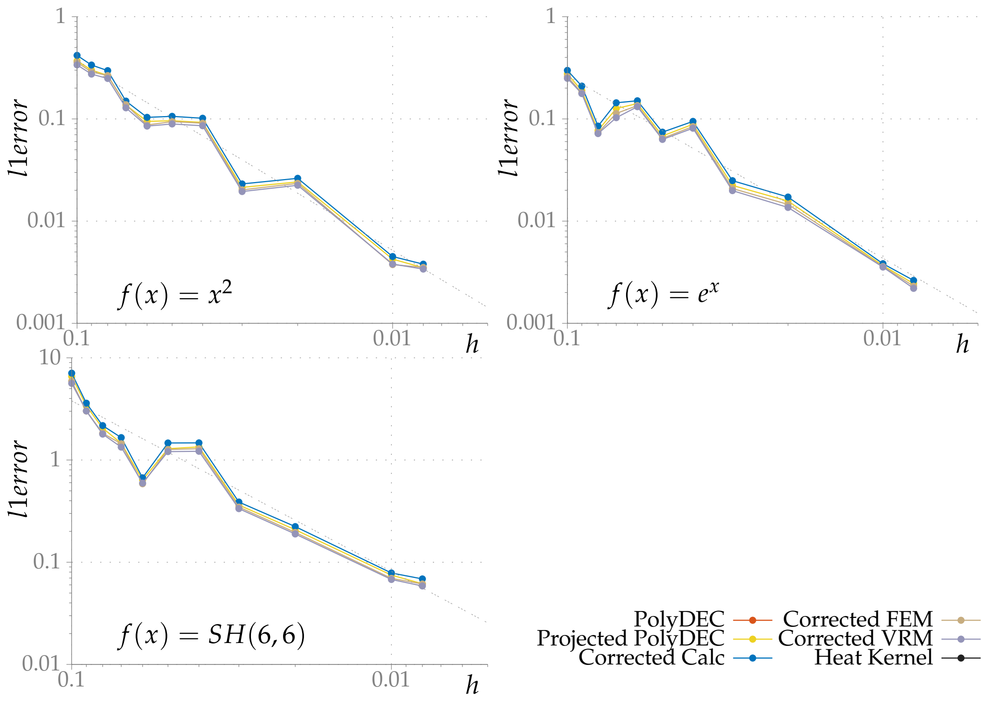


FIGURE 3.8: Comparison of results using integral invariant estimated normals for solving a Poisson problem.

3.4 Conclusion

We show that, similarly to the corrected PolyDEC method [CL22], a corrected normal field can be inserted within discrete calculus schemes yielding different discretizations of the Laplace-Beltrami operator. All these operators seem to converge when solving Poisson problems, and when used in a forward evaluation, the addition of

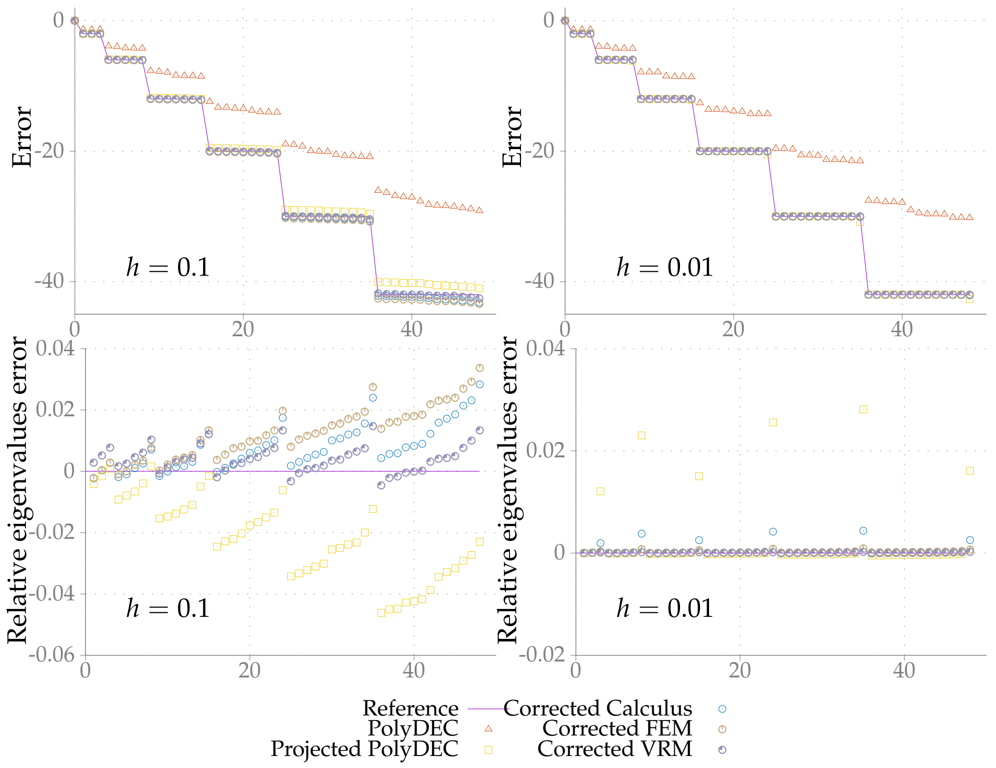


FIGURE 3.9: Smallest 49 eigenvalues of the Laplacian on unit sphere with discretization step 0.1 (left) and 0.01 (right). Relative error is given by $(\hat{\lambda} - \lambda)/\lambda$, where $\hat{\lambda}$ is the approximated eigenvalue and λ the correct value.

a small 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 [Cai+19] is yet proven to converge on digital surfaces (strong consistency). However, since results on a common scheme (the finite element method) seem promising, it may be interesting to see if the proof of convergence from this scheme 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 [Lac+20]). The idea of adding diffusion and modifying the mass matrix can be seen as similar to the approach of [HP11]. However Caissard *et al.* [Cai+19] 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 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 digitizations in general (such as a sphere).

We have integrated our methods into DGtal (pending Pull Request #1735: https://github.com/DGtal-team/DGtal/pull/1735) along with the other corrected operators. The different corrected and non corrected calculi have a common interface. It will make easier comparative evaluations, benchmarking as well as developing new applications.

In the next chapter, we see how our corrected Laplacian can be used to develop a regularization method for digital surfaces.

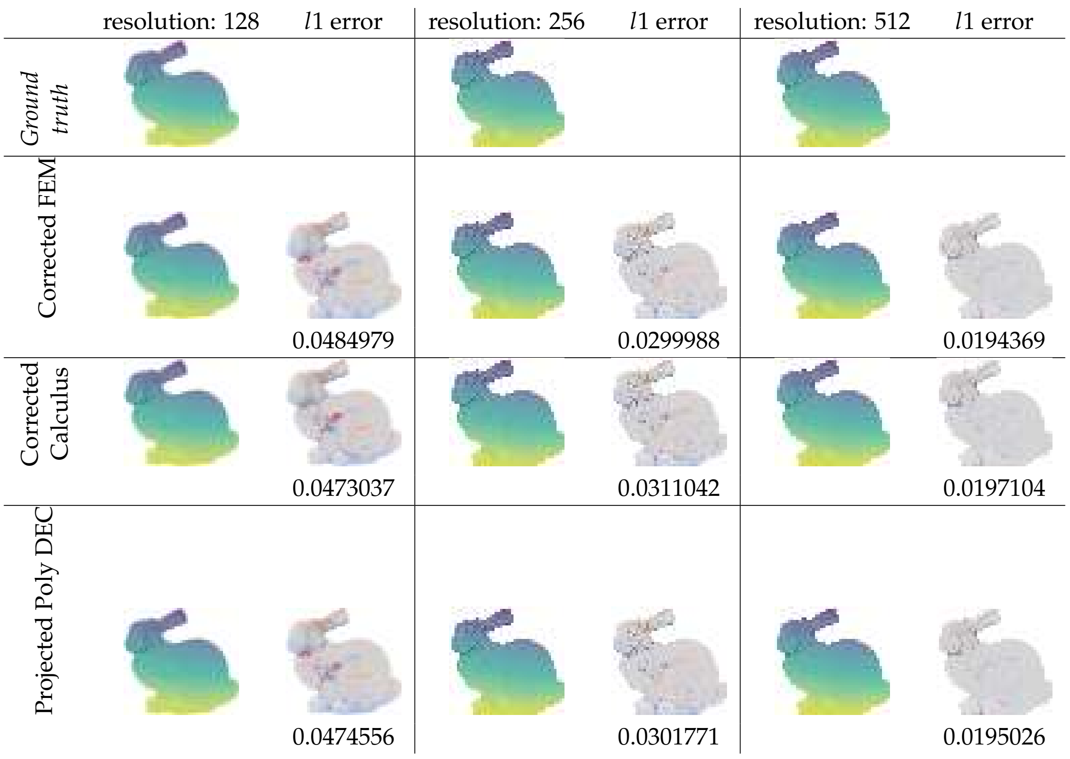


FIGURE 3.10: Comparison between classical cotan Laplacian of a function defined on a triangle mesh and the digital Laplacian operators. All three operators have comparable performances. The error decreases as the resolution increases, suggesting that all three operators are convergent.

CHAPTER

Some applications of corrected Laplacians 4

Résumé

Puisque nous avons une estimation du Laplacien Δ, mais également des estimateurs de normales et de courbure, on essaye de retrouver les positions de la surface avant digitalisation en s'appuyant sur la formule $\Delta\mathbf{p} = -2H\mathbf{n}$ où $\mathbf{p}$ sont les positions, $\mathbf{n}$ sont les normales et H la courbure moyenne. On développe ainsi une méthode de régularisation pour surfaces digitales. On peut élargir cette méthode à d'autres cas: par exemple, une surface dont les normales sont données dans une normal map. On peut également modifier la courbure moyenne au préalable, ou même intégrer les estimateurs de courbure afin de trouver un autre champ de normal et ainsi incorporer des contraintes de courbures dans la méthode.

On observe que l'utilisation d'un Laplacien corrigé donne bien de meilleurs résultats que la version non modifiées. Cependant, cette correction n'est pas forcément très visible, et l'utilisation d'un Laplacien naïf peut également présenter des avantages dans cette situation: les résultats obtenus sans corrections donnent des surfaces où toutes les arêtes ont des longueurs similaires et sont plus régulières.

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.

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 [SCT03], we need less precision in order to recover the result since encoding delta coordinates requires less precision that encoding the absolute position. Our approach can also include curvature modifications steps if needed.

The works presented in this chapter were submited for a special issue in JMIV (Journal of Mathematical Imaging and Vision).

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 [LCL17] (with $\alpha = 1/3$) for normals $\mathbf{n}$ on the digital surface, and then use corrected curvature measures [Lac+20] (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 4.1). 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 α is a weight between the two energies. The α 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 beging 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. [CLG21] (figure 4.2). 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 by [CLG21]. We compare the use of non corrected and corrected operators, by using non corrected ones built using the method from de Goes et al. [DBD20] and the normal corrected variants from Coeurjolly and Lachaud [CL22] (figure 4.3). 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

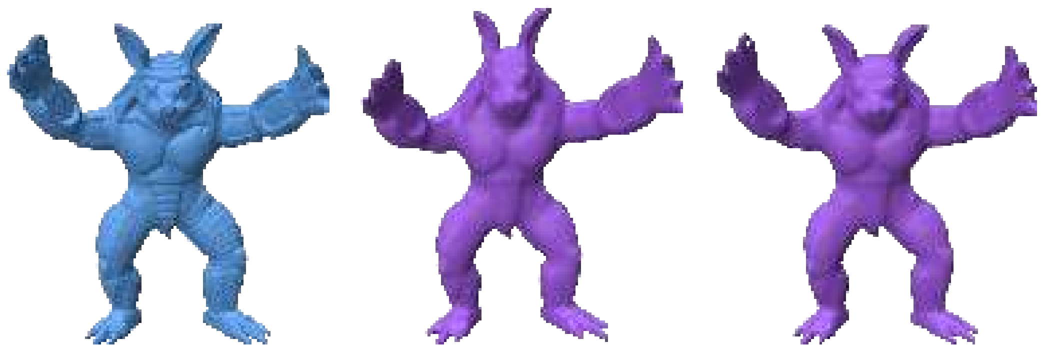


FIGURE 4.1: Regularization of a $256 \times 256 \times 256$ voxelization of the armadillo mesh with and without a position attachement term. While most of the features of the shape are obtained, the members highly deviate from the original position when $\alpha = 0$

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, α 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 4.4), 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 normals original surface: both errors are reduced as the grid step diminishes (figure 4.1).

4.1.1 Choice of α_0

We've chosen α_0 in order to demonstrate the results obtained with a low influence of the data attachment term. We can however experiment with different values (figure 4.5). We observe that the minimum is actually attained with a value of α_0 around 600 for the position error, and 1000 for the normal error. We also observe that the corrected operators improve the results in two ways: first, the error is constantly lower than the non corrected ones, and secondly the error does not vary much for a wide range of α_0. The use of corrected operators here increases the quality of the method and also its robustness.

We chose to stay at a low α_0 in the rests of our works. The error is only slightly worse, and this allows us to more clearly see the effects of a change in our choice of operator. In applications where the result should be numerically as close as possible to the expected one, high values for α_0 should be preferred: however, if we want to minimize the visual error (akin to [SCT03]), then lower values should be preferred (figure 4.6).

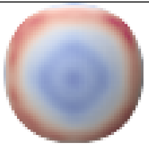
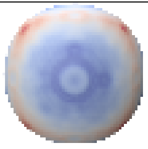
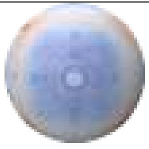
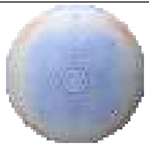
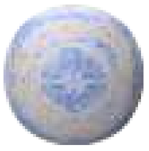
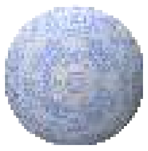
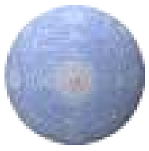
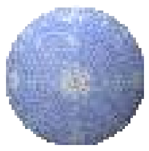
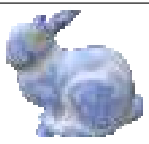
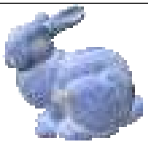
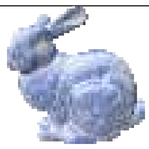
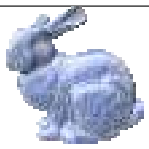
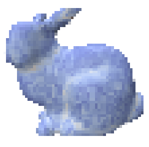
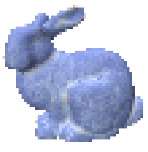
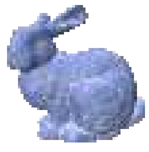
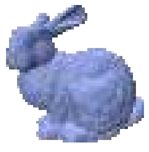
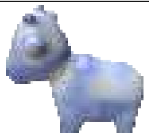
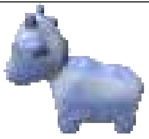
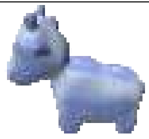
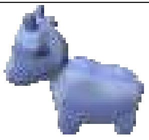
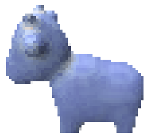
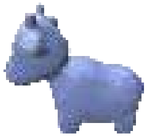
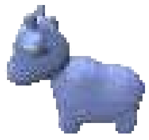
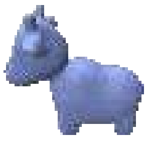
	grid size :	64x64x64	128x128x128	256x256x256	512x512x512
sphere	position error	 0.00583	 0.00289	 0.00138	 0.000688
	normal error	 0.0548	 0.0416	 0.0333	 0.0233
bunny	position error	 0.00457	 0.00235	 0.00118	 0.000184
	normal error	 0.239	 0.158	 0.0972	 0.0614
spot	position error	 0.00370	 0.00188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

TABLE 4.1: 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 l1 error.

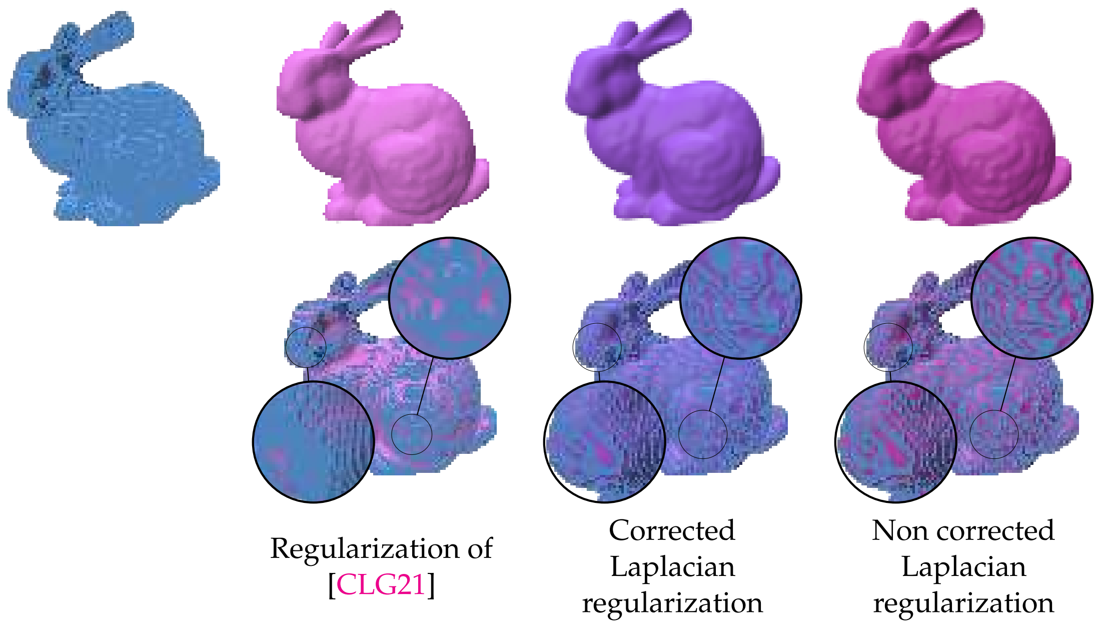

Regularization of [CLG21]
Corrected Laplacian regularization
Non corrected Laplacian regularization

FIGURE 4.2: We use several regularization methods on a voxelization of the bunny shape on a 256 × 256 × 256 grid. Left: original shape, middle-left: regularization of [CLG21], 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. [CLG21]. Here, the difference between corrected and non corrected is visually small.

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 arise 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.

A simple yet effective method for this task is to use the normal map in order to build a height field. The normal field, when projected onto the surface, gives the gradient of the height which we can then integrate (by computing its divergence and then solving a Poisson problem, for example). In order to build the geometry we sample normals and build operators on, we subdivide the original surface in 4 triangles for each triangle, without moving the vertices: (the process is repeated 4 times). Normals are then sampled at faces centers and then projected onto the face. This process does not need any corrected operators, since the height field exists on the discretized geometry. The height values, at vertices, are then added to the vertex along the normals of the low resolution surface.

Our approach consists in first subdividing the surface and then for each face sampling the normal at its center. We then use our method from the subdivided geometry and the corrected normal field: we first compute curvature, again using the corrected curvature measure, and then we integrate $-2H\mathbf{n}$ as before. A comparison between the two methods is shown on figure 4.7: while the height recovery methods show the best result in term of recovering details, it is also sensible to low quality

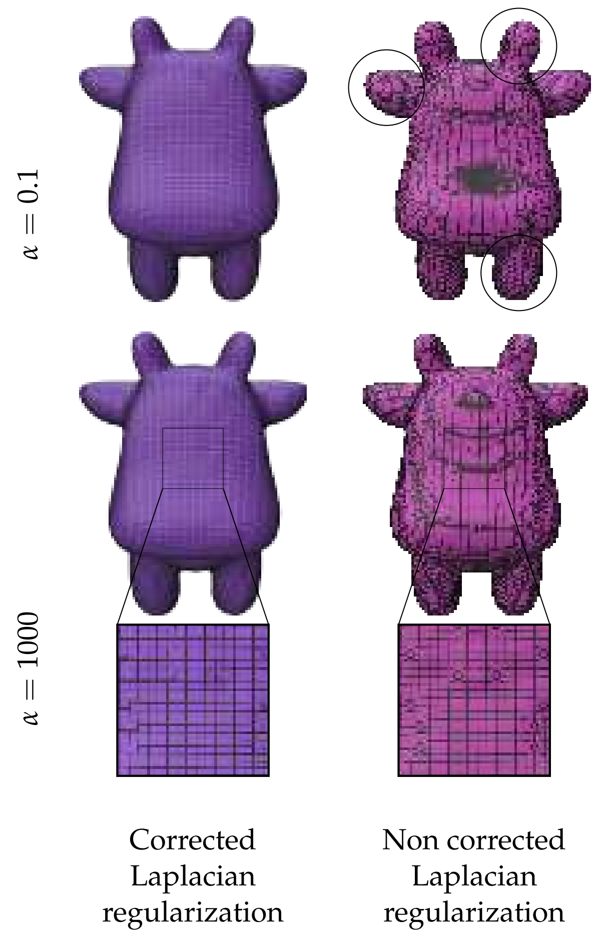

$\alpha = 0.1$
$\alpha = 1000$
Corrected Laplacian regularization
Non corrected Laplacian regularization

FIGURE 4.3: 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 α (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, α can be increased to decrease the influence of the Laplacian.

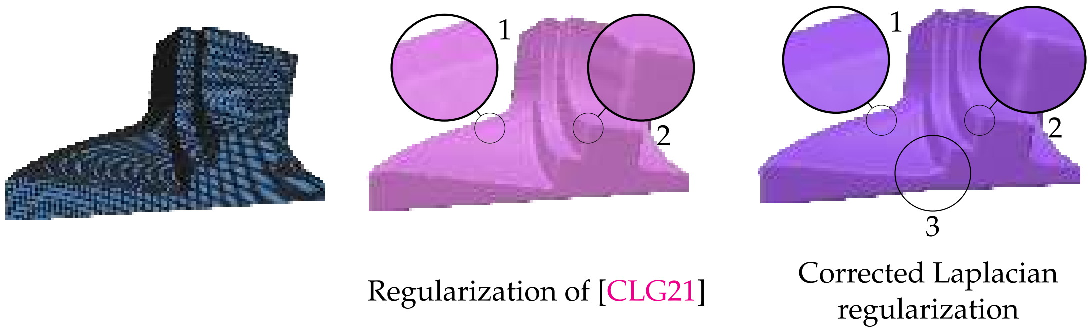


FIGURE 4.4: Top: original shape (voxelization of the fandisk shape on a $256 \times 256 \times 256$ grid). Middle: regularization from [CLG21]. Bottom: our regularization using a corrected Laplacian. Our method exhibits less sharp angles (2), but tends to be smoother and to avoid stair-like effects on 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 visible bump inside an angle (3).

normal maps with some artifacts showing at the border of the original triangles of the coarse surface. Using our regularization, since we smooth the curvature, the end result is smoother. In this case, the difference between corrected and non corrected methods is generally not noticeable.

The difference between corrected and non corrected operators is significant when doing this operation on digital surface. In this case, we also 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 coare mesh in order to sample the normal map on the coarse mesh, emulating having a normal map on the digital surface. The procedure is illustrated on figure 4.8. 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 somes points. Our method is still able to recover the original surface with some defects visible, and when using non corrected operators the results are significantly worse.

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 keep an attach to 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 4.9). 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

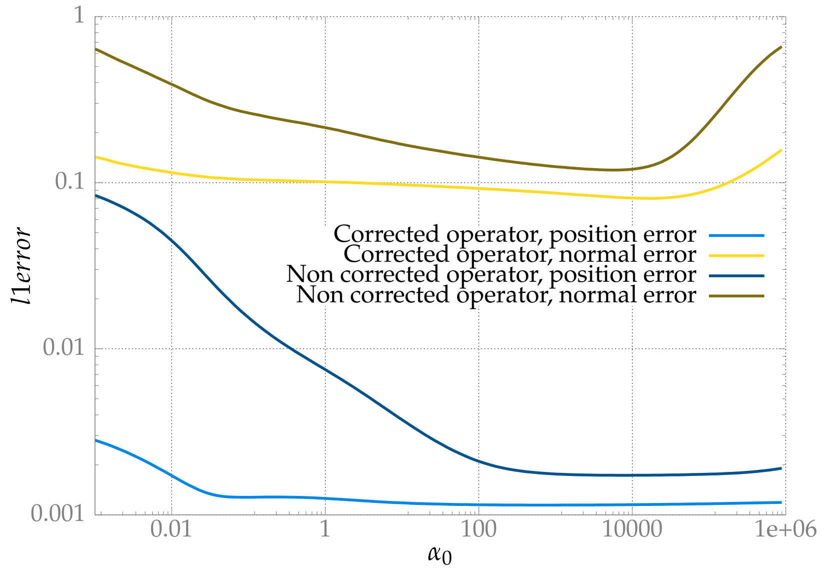


FIGURE 4.5: We plot the $l1$ error in position and normal for the regularization obtained on a voxelization of the bunny mesh on a $256 \times 256 \times 256$ grid. Using corrected operators lower the error, and makes the method more robust.

operator $S := d\mathbf{n}$. It is a symmetric 3×3 matrix, with at most two non-zeros eigenvalues k_1 and k_2 corresponding to the principal curvatures. Their eigenvectors corresponds to the principal curvatures 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 zerohowever (corresponding to the normal direction): this is fixed by computing instead $S + K\mathbf{n}\mathbf{n}^\top$, 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 $H' = \mathrm{Tr}(S')$. The procedure is summarized in algorithm 4.3.

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 $||M^{-1}LN' - dS||^2 + \alpha||N' - N||^2$
$N' \leftarrow (LM^{-1}L + \alpha M)^{-1}(-2LdS + \alpha MN)$;
$H \leftarrow \mathrm{trace}(S)$ //Compute mean curvature
$N' \leftarrow$ normalize($N'$);
//Minimize $||M^{-1}LP' - HN'||^2 + \alpha||P' - P||^2$
$P' \leftarrow (LM^{-1}L + \alpha M)^{-1}(-2LHN' + \alpha MP)$;
return $P'$;
```

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

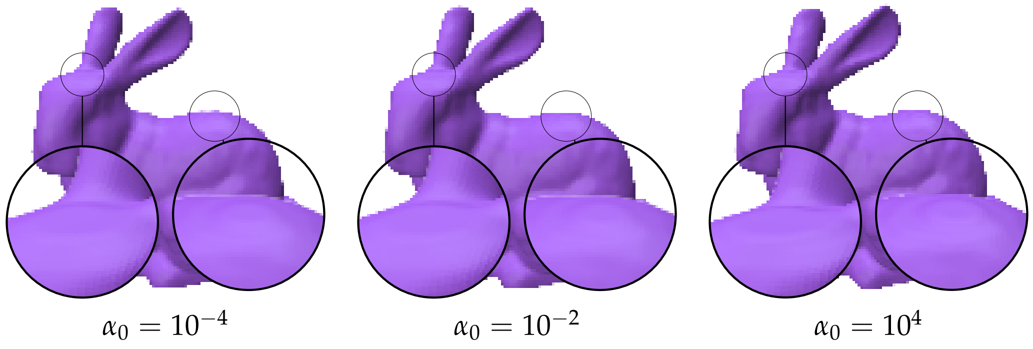


FIGURE 4.6: Regularization with various values of α_0. While on figure 4.5 we see that higher values tend to reduce the numerical error, the difference is only slightly visible here. High values tend to make the stair-like effect more noticeable, decreasing the visual quality: if the resulting surface is to be used on visual applications, lower values could be preferred.

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 4.10).

4.4 Conclusion

We have shown that our corrected operators could help us in order to help build a regularization, and we have seen that the use of corrected operators increases the quality of the output. This regularization also can include some user constraints in the form of curvature constraints. This further motivates the need for corrected operators directly on the digital surface, as opposed to building the operator after regularizing the surface: here we need the operators in order to obtain the regularization, and thus need corrected operators.

Further uses could also be investigated: for example functional maps [Ovs+12; Ovs+17], used for shape matching, usually rely on the Laplace-Beltrami operator.

The next chapter is focused on a topic less directly related to the Laplace-Beltrami operator, computing UV maps, although the operator has been used for this problem [Lév+02; Mul+08] and will intervene on the discretization of the functional we will introduce. Our goal is to develop a method based on the Ambrosio-Tortorelli functional for the simultaneous optimization of distorsion and seams.

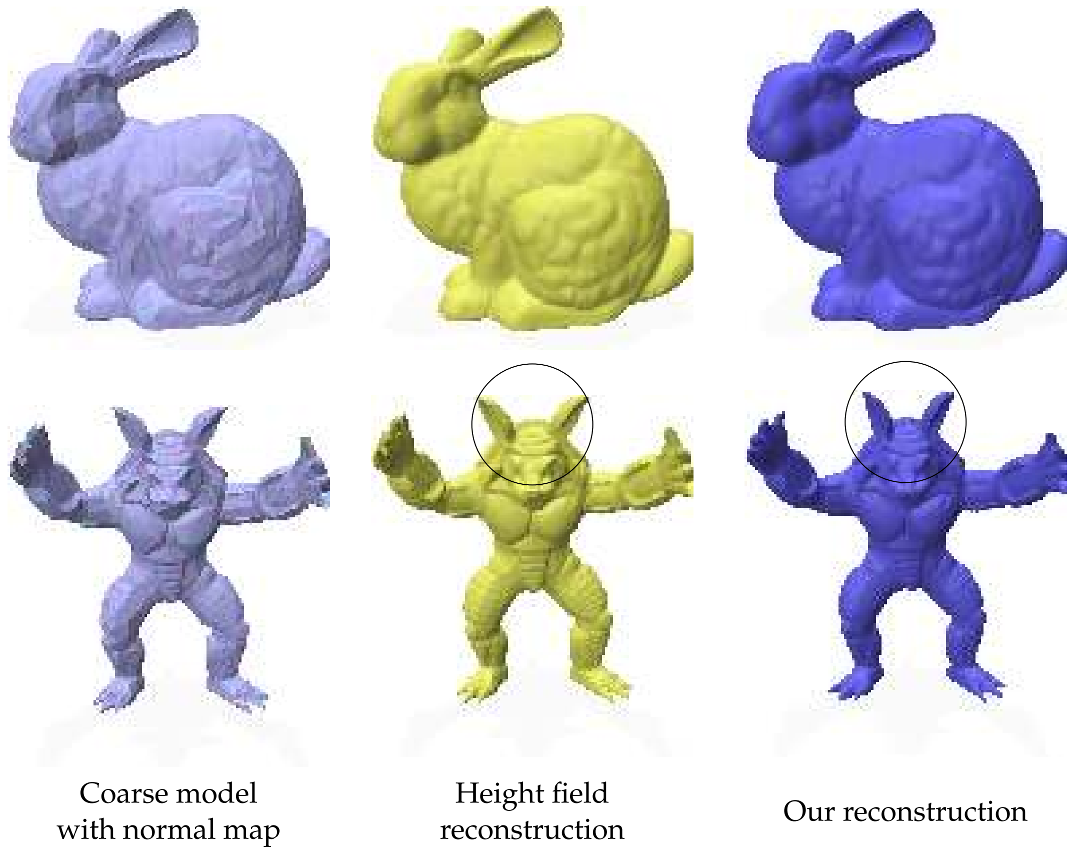
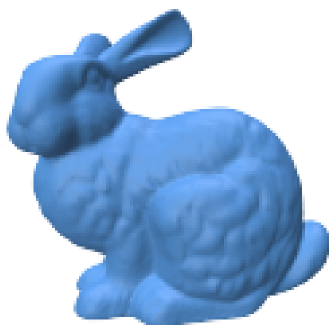
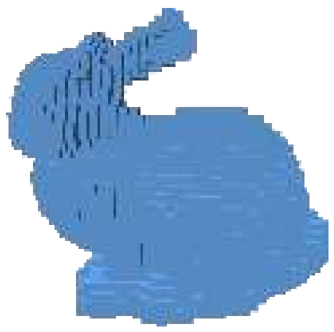
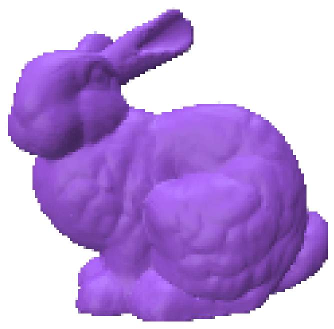
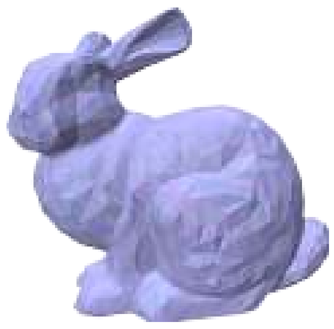


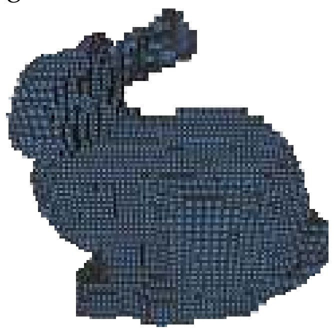
FIGURE 4.7: Left: coarse model with normal maps. Middle: reconstruction from height field. Right: reconstruction by normal and mean curvature estimation and integration. The height field reconstruction shows some defects due to the quality of the normal maps, which our reconstruction smooths out. However, our reconstruction also shows low frequency deviation from the original shape (especially noticable on the armadillo ears).


Original model


Voxelization of the coarse model on a $64 \times 64 \times 64$ grid


Regularization with corrected operator


Coarse model with normal map


Surfels subdivided

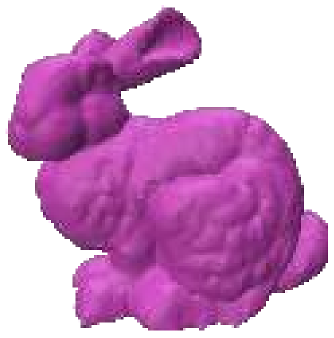
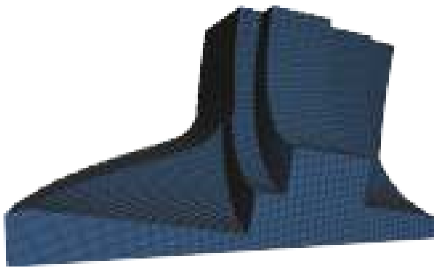
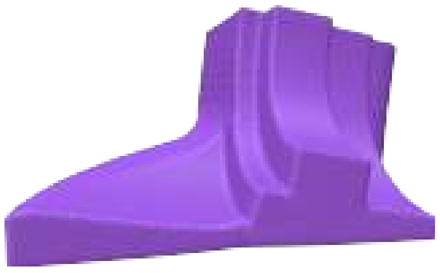
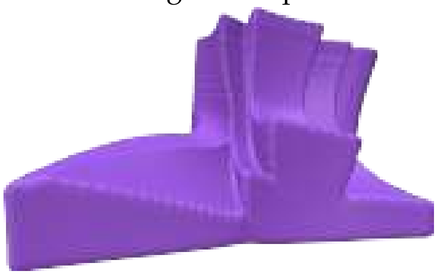

Regularization without corrected operator

FIGURE 4.8: 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.


Original shape


No modification


$H \leftarrow 2H$

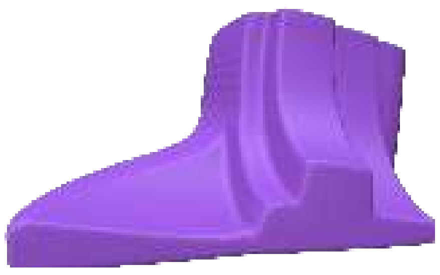

$H > 50 \Rightarrow H \leftarrow 50$

FIGURE 4.9: Various reconstructions on the fandisk shape with mean curvature manipulations.

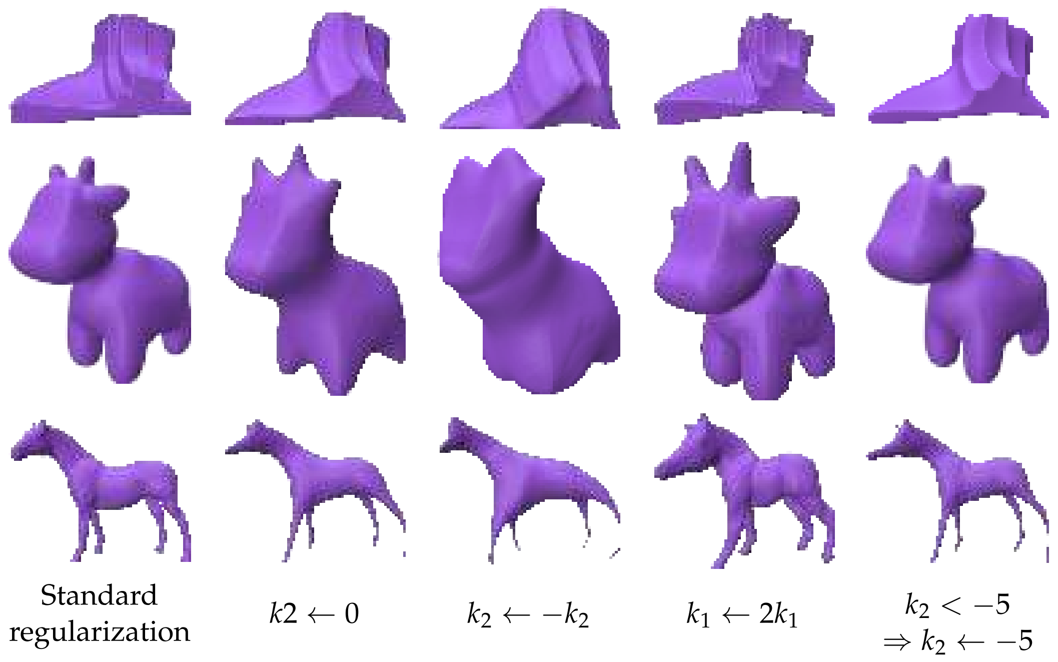


FIGURE 4.10: Various curvature editions on k_1, k_2 the principal curvatures, with $k_1 > k_2$. These are applied during the regularization process starting from voxelized shapes on a $256 \times 256 \times 256$ grid.

CHAPTER

UV-mapping using an Ambrosio-Tortorelli functional 5

Résumé

Les UV-maps sont des paramétrisations 2d de surfaces 3d auxquelles on permet des discontinuités. En d'autre terme, on cherche à prendre une surface, à la découper puis à l'aplatir. Cela permet notamment de créer ensuite des fichiers de texture, des images que l'on va vouloir plaquer sur la surface pour lui donner une certaine apparence. Il y a deux problèmes lors de la constructions de ces UV-maps: on veut d'un côté limiter la distorsion de la paramétrisation (une trop grosse distorsion résulte en des besoins de résolutions très différents entre plusieurs triangles, et rend la manipulation des images des textures plus difficile), et d'un autre limiter la longueur des coupes (qui rendent elles aussi la manipulation de l'image plus difficile).

De nombreuses approches existent pour essayer de résoudre ces deux problèmes. Cependant, ce sont généralements des approches qui séparent ces problèmes: les coupes sont calculées avant la paramétrisation. La qualité de la paramétrisation dépend cependant des coupes, et il y a donc un travail de prédiction de la qualité de paramétrisation depuis les coupes. Des travaux récents proposent une optimisation *simultanée* des deux, afin de ne pas avoir des coupes qui prédisent la paramétrisation mais qui s'y adaptent [Por+17; Li+18].

Nous présentons deux méthodes s'inspirant de la fonctionnelle d'Ambrosio-Tortorelli [AT90] pour réaliser une optimization simultanée de la distorsion et des coupes. L'une d'entre elles présentent des avantages en terme de premettre d'ajouter des contraintes sur le respect de l'injectivité de la paramétrisation, ainsi que de vitesse d'execution par rapport à des méthodes précédents auxquelles nous nous comparons.

Computing a UV map for a mesh is a classical problem in computer graphics. UV maps have many applications, the most direct being texture atlas creation for a 3D model. Given an input three-dimensional discrete surface, the task addressed in this paper is to define a low distortion mapping of the mesh geometry to the plane while minimizing the cut length. Such cuts, or seams, may be required for topological considerations (for shapes that are not homeomorphic to a disk), or may help to better minimize the UV distortion (different cuts may lead to different local minima of the distortion energy). The problem of finding cuts and the problem of computing the parameterization are often addressed separately: cutting can be considered as a topological discrete problem, while optimizing the distortion is a geometrical problem usually solved with variational approaches.

Our method aims at simultaneously optimizing those two problems using a variational model, so that instead of predicting the distortion, the cut can adapt itself to the distortion during the process. There already exist works exploring this direction: [Por+17] have proposed a variational method based on a per edge cut energy, [Li+18] have presented several edge merging/cutting operations and have used them during the optimization process to reduce a given functional.

Our contribution consists of employing a dedicated variant of Ambrosio-Tortorelli functional to achieve this joint optimization. This functional measures the smoothness of a function while allowing discontinuities: in our context, the function will be the parameterization itself (with a parallel between smoothness and distortion) and the discontinuities will correspond to the locations of cuts. Minimizing this functional gives us a simple framework to simultaneously optimize the distortion and find the cuts faster than previous simultaneous optimization methods. We explore two different discretizations of this functional, resulting in two different schemes relying on different operators.

The chapter is organized as follows. First, we briefly recall the challenges of surface parameterization (Section 5.1) and the Ambrosio-Tortorelli functional in the continuous setting (Section 5.2). Then, we relate the functional to the mesh parametrization problem and we propose a discretization scheme based on a corner/edges discretization and then one on a vertex/faces discretization which induces a fast joint distortion/cut optimization algorithm (Section 5.3). Finally, Section 5.4 details an experimental validation of the proposed approach.

The works presented in this chapter were presented at Shape Modeling International (SMI2023) [Wei+23].

5.1 Surface parameterization

Distortion minimization Given a mesh homeomorphic to a disk (to avoid any topological issues), there are several ways of defining a "good" parameterization depending on the distortion metric that is considered. The survey by [SPR06] has described a few of these methods. Some works focus on finding conformal parameterization [Lév+02], while other focus on finding a parameterization as isometric as possible, using energies measuring per triangle distortion such as ARAP, as-rigid-as possible, [Cha+10], Symmetric Dirichlet [SS15], or MIPS, most isometric parameterization, [HG12]. In the following, we focus on the Symmetric Dirichlet energy as introduced by Smith and Schaefer [SS15], whose expression is given by $\Psi_{symDir}(||F||^2 + ||F^{-1}||^2)/2$. Once the distortion metric has been specified, minimizing this energy is a problem in itself. Most optimization processes consist of two steps: (i) initialize the parameterization (with methods like [Tut63], which guarantees an injective parameterization) and (ii) minimize the distortion energy. To minimize these energies, we rely on quasi-Newton methods, and we need ways to approximate the Hessian with a positive symmetric definite matrix (PSD). Note that [SGK19] have provided a way to efficiently compute a PSD matrix approximating the Hessian for distortion energies.

Cutting methods There are several works focusing on computing cuts for a parameterization. The method [SH02] searches for some high curvature point (likely to cause distortion) and then approximates a minimal path between these points. Recently, [Zhu+20] have introduced a method that detects some feature points and connect them by approximating a Steiner tree problem. While these methods are efficient, our work aims at developing a method based on simultaneous optimization, in which little work has been done compared to these methods based on connecting high distortion points. [SC18] have also provided a variational approach to compute cuts minimizing the distortion. However, their work remains limited to conformal parameterizations, while we aim at reducing an isometric distortion.

Simultaneous cuts and distortion optimization Recent works aim at simultaneously finding the cut and the parameterization. Notable works are AutoCuts [Por+17], a variational model using an energy which is the sum of a distortion energy (such as Symmetric Dirichlet) and a per-edge energy approaching the cut length measure. The parameterization is treated as a triangle soup with the per-edge energy deciding whether neighbor triangles are attached to each other or not. The process can be automated, but user input is necessary in order to guarantee the bijectivity of the resulting parameterization. The weight between each term has to be defined by the user, and there is no guarantee that the result can reach a distortion or a cut length below a chosen threshold. The OptCuts method [Li+18] is based on a set of operations on the mesh topology such as cutting or merging, and uses a dual formulation of the energy problem to find the most appropriate operation to solve the problem. This method generally outperforms AutoCuts in terms of balance between distortion and cut length, with lower timings as well. This method can be adapted to guarantee bijectivity, and a distortion threshold to be reached can be set. Our method aims at providing a variational model, in a way similar to AutoCuts, but inspired by the Ambrosio-Tortorelli functional instead of using a handmade per-edge energy. We can then take inspiration from other Ambrosio-Tortorelli optimization related works to obtain an efficient and fast strategy to find our cuts.

Bijectivity One of the problems of surface parameterization is to guarantee the bijectivity of the result. If the embedding proposed in [Tut63] is guaranteed to be valid, optimizing an energy such as Symmetric Dirichlet has no reason to maintain it. One needs to enforce the bijectivity during the distortion optimization. [SS15] have decomposed the problem into two subproblems. First, we need to make sure that triangles are not flipped during the optimization, which is solved by adding a constraint on the distortion energy and a specific line search in the Quasi-Newton solver. Then we need to prevent overlaps of non-adjacent regions which can be detected for instance by preventing boundary edge crossings. [SS15] have solved this problem using a barrier energy term. [Su+20] have optimized this approach by building a shell around the initial parameterization to reduce the number of border edges and vertices. Alternatively, [JSP17] have proposed to triangulate the remaining space around the parameterization, which in turns guarantees local injectivity on the resulting triangulation. In our proposal, we mainly focus on speeding up the distortion/cut joint optimization. Our approach is fully compatible with existing strategies (either using [SS15] or [JSP17]), if global bijectivity is required.

Mumford-Shah related optimization The Mumford-Shah functional was first introduced for image processing [MS89], as a tool for image restoration and segmentation. It gives a way to represent an image as a piecewise-smooth function with a set of discontinuities. Since this functional is difficult to optimize (see next section), a lot of approximations of this functional have been proposed in the literature. We focus here on one relaxation of this functional, called Ambrosio-Tortorelli functional [AT90]. It has seen many applications in image processing since then, some direct such as segmentation, denoising, and some variants designed for inpainting, magnification, deblurring, or image registration. While this functional has remained largely ignored in geometry processing, it has been used recently for mesh processing applications [Coe+16; Bon+18; Liu+20].

5.2 The Ambrosio-Tortorelli functional

Our joint distortion/cut variational model takes inspiration from the Ambrosio-Tortorelli functional. As detailed below, it can represent in the same framework the parameterization itself and the cut loci as functions.

In order to describe the Ambrosio-Tortorelli functional, it is necessary to introduce the Mumford-Shah functional [MS89]. The Mumford-Shah functional allows us, given an arbitrary function g, to find a new function u that will try to be as close as possible to g while also trying to be as smooth as possible everywhere except along a set K of discontinuities. This functional is commonly used in image processing, with a direct use in denoising: we want a smooth output close to the original input image, but we do not want to oversmooth meaningful image contours, so the function should have discontinuities along these contours. Ideally, the set K delineates these features. The functional is defined as:

$$\mathscr{M}\mathscr{S}(K,u) := \alpha \int_{\Omega\setminus K} |u-g|^2 + \int_{\Omega\setminus K} |\nabla u|^2 + \lambda \mathscr{H}^1(K\cap\Omega)\,, \tag{5.1}$$

where Ω is the domain, g is the input data defined on Ω, u is the regularized data, K is the set of discontinuities, $\mathscr{H}^1$ is the Hausdorff measure, and (α,λ) are two real numbers corresponding to the weights given to the fitting and cut length terms respectively. Increasing coefficient α forces the output to be closer to the input, while increasing coefficient λ induces fewer discontinuities.

The first term of this functional describes the fitting of u to the input data g, the second term expresses the regularity of u while the third term measures the length of the discontinuities. Minimizing these three terms simultaneously lets us find a compromise between input fitting, smoothness of the result and how much discontinuity is allowed.

However, this functional is hard to optimize since it is difficult to manipulate a set of discontinuities. [AT90] have provided a relaxation of the Mumford-Shah functional, as it approximates its minimizer with two smooth functions, instead of one smooth function and one set of discontinuities. It can be defined as:

$$\mathscr{A}\mathscr{T}_\epsilon(u,v) := \alpha \int_\Omega |u-g|^2 + \int_\Omega |v\nabla u|^2 + \lambda \int_\Omega \left(\epsilon|\nabla v|^2 + \frac{1}{4\epsilon}(1-v)^2\right)\,,$$

where function v is from domain Ω to $[0,1]$, and ϵ is a real positive number.

[AT90] have proven that this functional Γ-converges toward the Mumford-Shah functional as ϵ tends to 0. Function v converges to an indicator function of the space where discontinuities are not allowed: it is set to 1 where u is smooth, and to 0 where u can have discontinuities. A large value for ϵ will give a diffuse result, and as we decrease ϵ the function will be less and less diffuse and takes values closer to 0 and 1. The main interest of this formulation is that $\mathscr{AT}_\epsilon$ minimizers are the same as $\mathscr{MS}$ ones when ϵ tends to zero. Furthermore, for a given ϵ, an efficient alternating minimization scheme can be designed (see Alg. 2): by fixing v (respectively u) the resulting expression is convex quadratic in u (respectively v). We can thus find the minimizing u or v with a single step of a Newton method. We clarify the gradients and Hessian operators for the discretization of this functional in the next section.

Algorithm 2: Optimization method for the Ambrosio-Tortorelli functional

```
Data: $\epsilon_1$ the starting epsilon, $\epsilon_2$ the ending epsilon, $n$ a fixed number of
      optimization steps
Result: $v$ and $u$ approximation of the functional minimizer
$\epsilon \leftarrow \epsilon_1$
while $\epsilon > \epsilon_2$ do
    for $i \leftarrow 0$ to $n$ do
        $v \leftarrow \min_v(\mathscr{AT}_\epsilon(u,v))$
        $u \leftarrow \min_u(\mathscr{AT}_\epsilon(u,v))$
    $\epsilon \leftarrow \epsilon/2$
```

5.3 Our discrete AT inspired model for joint parameterization/cuts

We work on a three-dimensional surface, specifically a triangle mesh, defined by a set V of vertices and a set F of triangular faces. We denote as n_f the number of triangles of the mesh, n_v the number of vertices, and n_e the number of edges. For each face f and edge e, the associated area is denoted A_f and A_e.

Functions u and v must be discretized. Standard discretizations make u and v defined per vertex, but this approach tends to smooth the functions. This is not desirable, especially for v. We thus prefer u and v be discretized not at the same place on the mesh. Since u represents our parameterization, it is natural to sample it at either vertices or triangle corners: u is then linearly interpolated on each triangle as expected.

5.3.1 A first approach

A first way to adapt AT to our problem is to try to set v at the edges, with the idea that v will control how the parameterization along the two faces adjacent to the edge must be continuous or not. In order to accomplish this, we have to view the surface as a "triangle soup", with each triangle free to distance itself from other triangles (with v controlling the links between triangles).

We speak of "triangle soup" for a parameterization where we attribute a position in the plane not only at each vertex but at each corner of every triangle of the surface. This gives us $3n_f$ corners to map in the plane instead of n_v vertices. This enables more degrees of freedom in the parameterization, by not binarizing the "cut" or

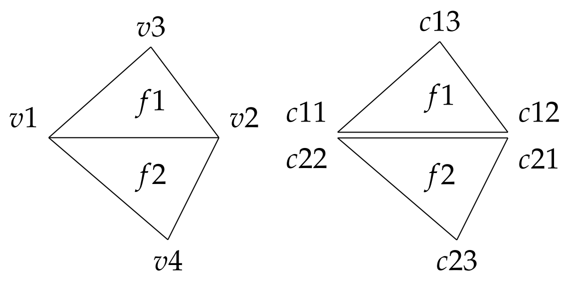


FIGURE 5.1: Differences between vertex and corner parameterization: instead of one value per vertex, we have one value for each triangle corner. As a result, each edge can be duplicated.

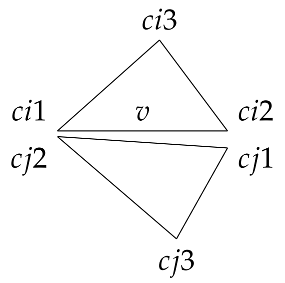


FIGURE 5.2: Values used in the gradient expression $\nabla\mathbf{u}$.

"not cut" status of each edge: in this aspect, our method is similar to the autocuts method [Por+17].

We then define v as a function which takes its values on each edge, with the intent that v will control the discontinuity: $v = 1$ will mean that the triangles will be attached (no cut), while $v = 0$ indicates a cut at this edge. This method has the advantage of intuitively defining the cuts: cuts will be edges where $v = 0$, and the associated u will directly give us the parameterization with the given cut.

As the matching term from Ambrosio-Tortorelli does not make sense here (we have no function to match to), we have to modify the formulation: we substitute the matching term by a term to measure the distorsion. We get the resulting functional:

$$AT_{modif}(\mathbf{u}, v) = \underbrace{\alpha \sum_f A_f \Psi(\mathbf{u}_f) +}_{\text{distorsion term}} \underbrace{\sum_e A_e((v\nabla\mathbf{u})^2}_{\text{gradient or smoothing term}} + \lambda\epsilon(\nabla v)^2 + \frac{\lambda}{4\epsilon}(1-v)^2), \tag{5.2}$$

where $\mathbf{u}_f$ denotes the corner values of $\mathbf{u}_f$ in face f, and $\mathbf{u}_e$ around edge e. We denote $\Psi(\mathbf{u}_f)$ the distorsion measure of triangle f (here corresponding to symmetric Dirichlet). Note that the gradient of $\mathbf{u}$ (in the gradient term) is undefined at discontinuities, as the edge has two different images. We propose the following expression instead, consisting of the integration of the linear interpolation of the distance between the two edges (see figure 5.2 for notations):

$$(v\nabla\mathbf{u})^2 = \beta v^2 \int_0^1 ||s(\mathbf{u}(c_{i1}) - \mathbf{u}(c_{j1})) + (1-s)(\mathbf{u}(c_{i2}) - \mathbf{u}(c_{j2}))||^2 ds, \tag{5.3}$$

Where β is a chosen constant that controls the strength of this term.

After integrating, the expression can be refactored in matrix form as:

$$(v\nabla \mathbf{u})_e^2 = \beta v^2 \mathbf{u}_e^\top G_e \mathbf{u}_e, \tag{5.4}$$

with, in one dimension: $G_e = \frac{1}{6}\begin{pmatrix} 2 & 1 & -2 & -1 \\ 1 & 2 & -1 & -2 \\ -2 & -1 & 2 & 1 \\ -1 & -2 & 1 & 2 \end{pmatrix}$. To go from one to two dimensions, we just have to substitute G_e by the block matrix $G_{2D} = \begin{pmatrix} G_e & 0 \\ 0 & G \end{pmatrix}$. This expression gives us an easy way to compute the energy: we sum this matrix for each edge in the surface (weighted with the correct v^2), resulting in a global G matrix yielding the total energy with $\beta \mathbf{u}^\top G \mathbf{u}$.
Once we have this matrix, it is also easy to compute the gradient and Hessian of this term: the gradient $b_{smoothing}$ and the Hessian $H_{smoothing}$ are given by:

$$b_{smoothing} = \beta G \mathbf{u}, \tag{5.5}$$

$$H_{smoothing} = \beta G. \tag{5.6}$$

Since G_e is SPD, it follow that G itself is SPD and can thus be directly integrated into a Newton method.

Since we now have a coefficient in front of each term, we can freely remove one. We remove the α from expression 5.2. We end up with the following expression:

$$AT_{modif}(\mathbf{u}, v) = \sum_f A_f \Psi(\mathbf{u}_f) + \sum_e A_e(\beta v^2 \mathbf{u}_e^\top G_e \mathbf{u}_e + \lambda\epsilon(\nabla v)^2 + \frac{\lambda}{4\epsilon}(1-v)^2). \tag{5.7}$$

In order to get an homogeneous expression and avoid mesh specific coefficients, we compute λ; ϵ and β using a characteristic length l as well as input values λ_0, ϵ_0 and β_0. The characteristic length here was taken as the average edge length on the mesh. The resulting expressions are, because of homogeneity constraints:

$$\lambda = l\lambda_0, \tag{5.8}$$

$$\epsilon = l\epsilon_0, \tag{5.9}$$

$$\beta = \frac{\beta_0}{l^2}. \tag{5.10}$$

We settled to using λ_0 between 1 and 10, and β_0 between 1 and 100 following the reasoning in Section 5.3.1.

Optimization

Optimizing this functional is done by alternating two steps, as seen previously: one at u fixed and one at v fixed.

- At v fixed, non constant terms are the distorsion term and the gradient one. We must simultaneously optimize these two terms: the first one will try to reduce the distorsion for each triangle while the other will try to keep neighboring triangles close to each other.
  We can draw a parallel between this step and the "standard" optimization of the distorsion. By placing a high enough coefficient in front of the gradient

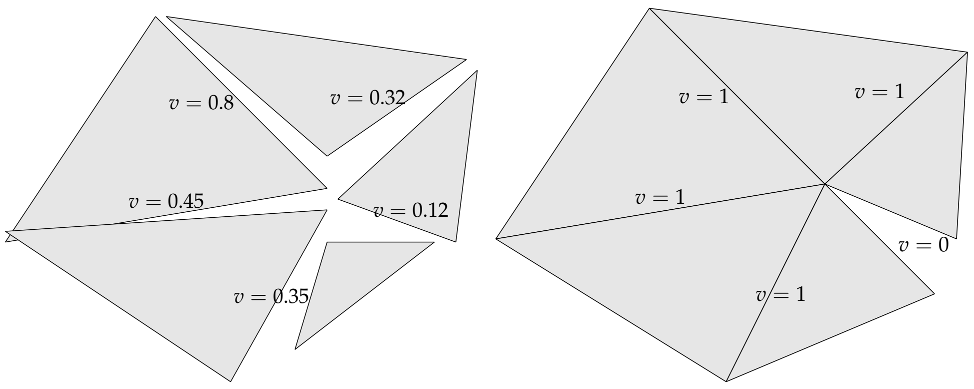
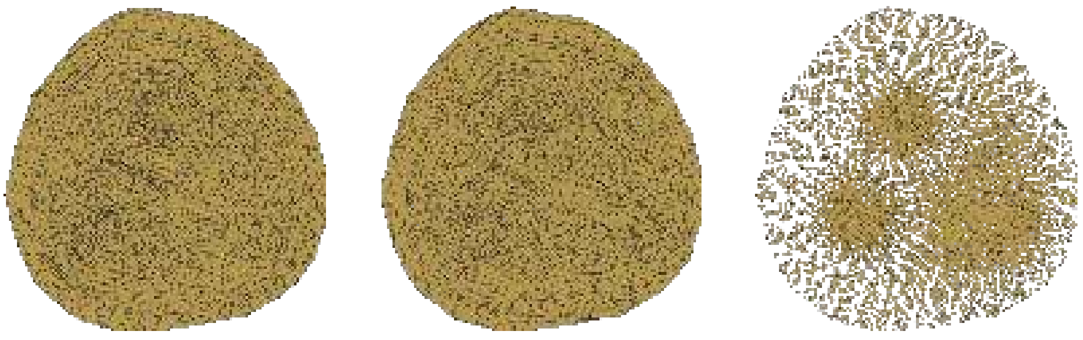


FIGURE 5.3: Illustration of our first method: we allow faces to optimize their distorsion, with v controlling the edge distances. As ϵ gets lower and lower, v takes value closer to either 0 and 1 and cuts get more precisely defined.

term (by increasing β), every corner will be attached to its neighboring corners (the one belonging to the same vertex). As shown in figure 5.4, the result is a parameterization similar to the ones obtained by a vertex parameterization. Removing the gradient term produces the opposite result: every triangle becomes free to optimize its distorsion and the result is a parameterization free of distorsion with no coherence between neighboring triangles.



(A) Original vertex parameterization

(B) Corner parameterization obtained with $\beta_0 = 10^3$

(C) Corner parameterization obtained with $\beta_0 = 10^{-1}$

FIGURE 5.4: Changing the value of β in equation 5.3 lets us control the balance between distorsion and triangle coherence.

- The second step (at u fixed) of the optimization is the same as seen for AT previously, as the distorsion term does not intervene. The energy is convex quadratic, and its minimum can be computed by computing $-H_v^{-1}b_v$, where

$$H = \frac{\lambda}{4\epsilon}M + \epsilon\lambda L + \beta M G(\mathbf{u}),$$

and

$$b_v = -\frac{\lambda}{4\epsilon}M * \mathbf{1}_{n_e}.$$

Here M denotes the edge mass matrix and L the edge stiffness matrix, $\mathbf{1}_{n_e}$ is a vector of size n_e filled with ones and $G(\mathbf{u})$ is a diagonal n_e by n_e matrix where each entry holds the value $\mathbf{u}_e^\top G_e \mathbf{u}$.

Note that, contrary to optimizing Ambrosio-Tortorelli, the first step does (with v fixed) not consist of simply optimizing a quadratic energy. We have to go through a quasi-Newton method that must be iterated until convergence (sometimes as much as a hundred iterations), instead of the single iteration from previously. As a result, these steps take most of the time of the optimization method.

Algorithm 3: Optimization method for the Ambrosio-Tortorelli functional

```
Data: $\epsilon_1$ the starting epsilon, $\epsilon_2$ the ending epsilon, $n$ a fixed number of
      optimization steps, $S$ the surface mesh $\beta_0$ and $\lambda_0$ balancing coefficients
      for equation 5.7, 5.10 and 5.8
Result: $u$ corner parameterization, $l_e$ list of cut edges
$v \leftarrow getInitCut(S)$
$u \leftarrow Tutte(S, v)$
$\epsilon \leftarrow \epsilon_1$
while $\epsilon > \epsilon_2$ do
    for $i \leftarrow 0$ to $n$ do
        $u \leftarrow getOptimalParameterization(S, u, v, \beta_0)$
        $v \leftarrow \min_v(AT_{modif}(u, v, \beta_0, \lambda_0))$
    $\epsilon \leftarrow \epsilon/2$;
$l_e \leftarrow \{\}$
for $e$ in edges do
    if $v(e) < 0.5$ then
        $l_e \leftarrow l_e + \{e\}$
```

The parameterization optimization is done with a QN method: we iterate the step $u = H^{-1}b$, until the gradient's norm drops below a certain threshold (chosen as 10^{-4} as per [SGK19]). Here b and H are defined as $b = b_{distorsion} + b_{smoothing}$, $H = H_{distorsion} + H_{smoothing}$, $H_{smoothing}$ being the Hessian defined in equation 5.6 and $H_{distorsion}$ being the SPD matrix approximating the distorsion energy's Hessian, computed using the method described in [SGK19].

Initial cut The initialization being done with Tutte's function, the initial surface must be isomorphic to a disk. It is necessary to go through an initial cut. We choose this cut using the *cut_to_disk* from libigl ([JP+18]) in the case of a surface of genus > 1. If the surface is of genus 1, we choose a minimal cut constituted of two edges from the same face. These edges are also set to $v = 0$.

Results Resulting cuts have a satisfying appearance (no patch of cut edge close to each other), and going through places with features that may create a lot of distorsion. The cut being given at the same time as the parameterization, there is no need to do any more computations in order to get the parameterization. We did observe a significant reduction of the distorsion (Figure 5.5).

The main difficulty after the optimization of the parameterization is to ensure that the result is bijective. The bijectivity is divided in two criteria: local injectivity and boundary injectivity.

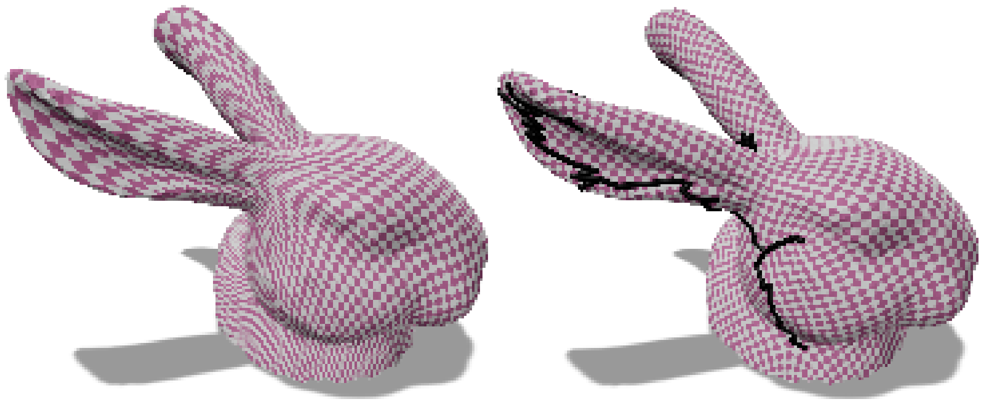

(A) Parameterization without additional cuts (B) Parameterization from our method

FIGURE 5.5: Comparison of parameterizations without and with cuts from our ethod: our has significantly reduced distorsion.

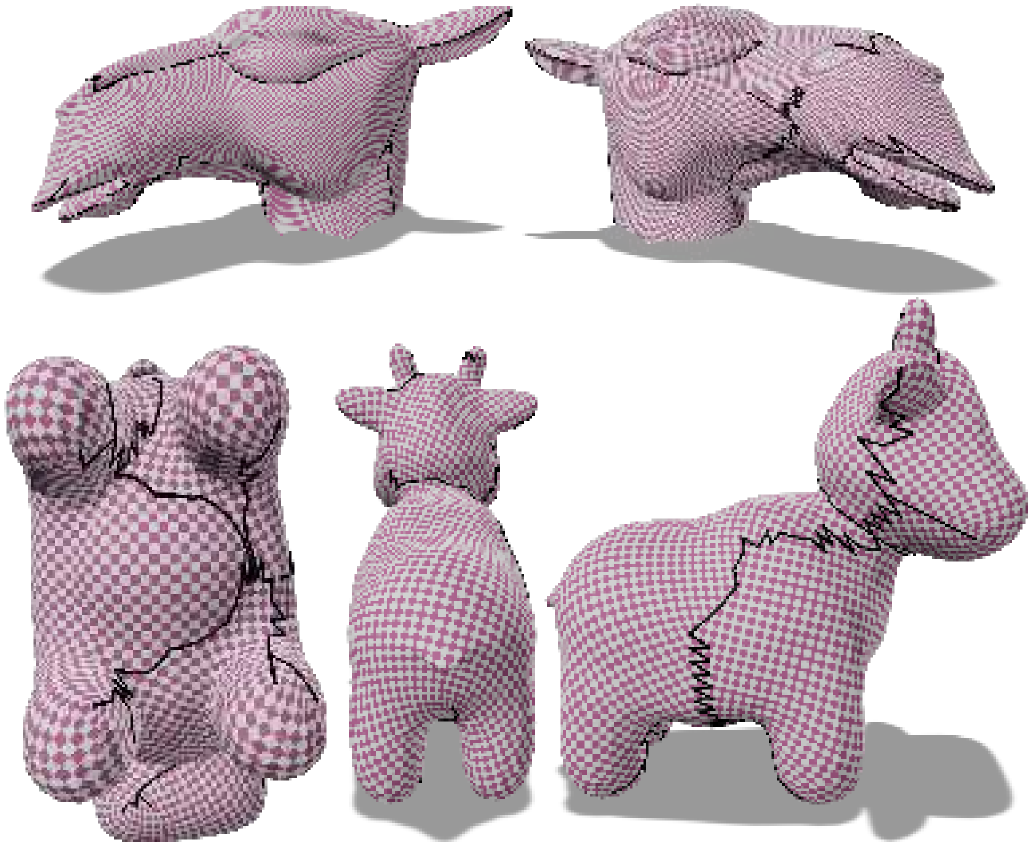


FIGURE 5.6: Resulting cuts generally do not form cycles, and follow the features of the mesh while reaching areas causing distorsion (the ears, the mouth, the legs...).

Parameter tuning

We have several parameters to input when running the method described in algorithm 3.

Convergence parameters We have three parameters related to Ambrosio-Tortorelli convergence: ϵ_1, ϵ_2 and n.

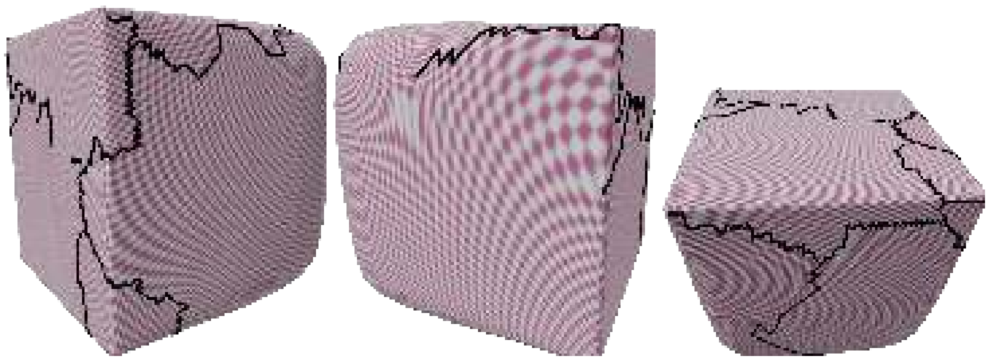


FIGURE 5.7: Cuts losely follow the features of the mesh, enabling a parameterization with a very low distorsion. We would however prefer straighter cuts as well as fewer cuts.

Convergence has a direct effect on the quality of the cut: if v does not have enough time to converge, it will become diffuse and lots of small cuts may appear (see figure 5.8). We prefer a few large cuts instead of several small ones.
In order to get a visually satisfying result, our experiments showed that the following values gave the best result (from a visual evaluation) at the lowest computing cost: $\epsilon_1 = 1$, $\epsilon_2 = 0.01$ and $n = 5$. This means we iterate the alternated optimization around 35 times, which is quite a lot.

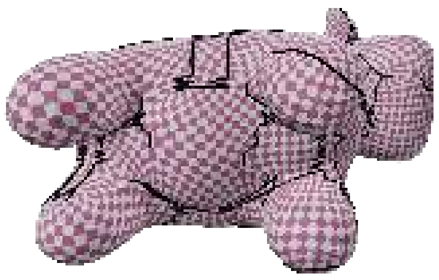


FIGURE 5.8: If the optimization of v does not use enough iterations, the cuts are fuzzy and present at many places.

Energy balance parameters These parameters are the β_0 and λ_0 intervening in equation 5.10 and 5.8 respectively, and thus in equation 5.7.
Parameter β_0 characterizes the strength of the gradient term, while parameter λ_0 is associated to the strength of the cut. Increasing β_0 gives a result with triangles closer to each others, but we observe that if it is too strong then no matter the value of λ_0 no cuts will appear: triangles do not separate enough for v to develop itself. Decreasing β_0 too much gives results with triangles far away from each other without necessarily having a cut between them: we could think that it just suffice to lower β_0 at the end, but then the cut may not actually be well suited to the result. We found that values balancing these two aspects were given for β_0 around 10.
We also found that the range of λ_0 values giving a good amount of cut with the corresponding convergence parameters was around $0.1 \leq \lambda_0 \leq 10$.

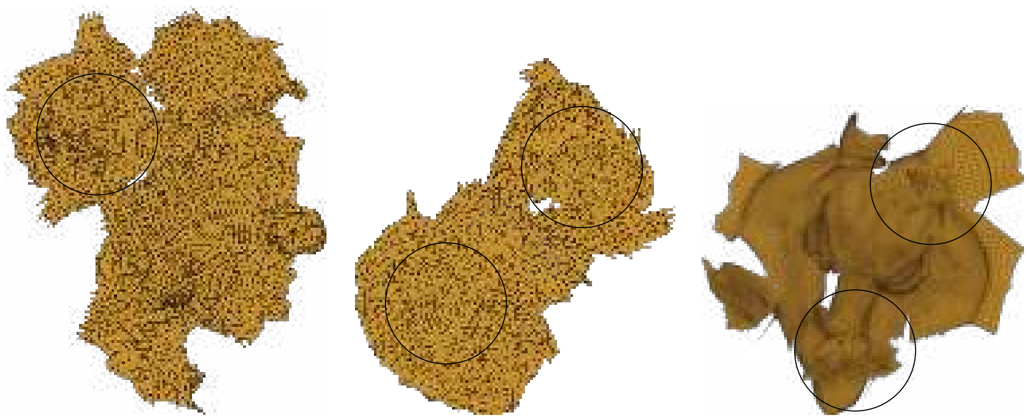


FIGURE 5.9: Parameterization obtained on spot, bunnyhead and camelhead. Due to the lack of bijectivity enforcement, many parts overlap.

Enforcing bijectivity

Local injectivity is assured by the use of the modified linesearch during the optimization. Boundary injectivity is not necessarily respected however (figure 5.9), and the method we choose for this problem is the one from Bijective parameterization with free boundaries. Indeed, SCAFFOLD [JSP17] did not seem adequate to our "triangle soup", where all the space between triangles should be triangulated instead of only the exterior space. Efficient bijective parameterization defines a shell around the border, and this step is done at the beginning of the parameterization as it requires the border to be fixed to the border of a convex surface: this is the same requirement as Tutte, so the shell is defined right after the Tutte embedding is computed. As our method redefines regularly new borders, we would need to restart the optimization from Tutte every time a new cut is defined instead of reusing the previously computed parameterization. This would make the method significantly slower (when it was already taking a few minutes).

The chosen method requires an energy depending on the distance between every border vertices and edges. We had to find a way to adapt our method, as in our "triangle soup" every edge is technically a border. In order to define this energy, we only take into account edges with a value v under a threshold. Different values were experimented for the threshold: while 0.5 seemed natural at first, but v being very diffused during the first iterations and many edges are classified as border edges. This would make the border energy explode and introduce too much instability.

Other threshold were then tried between 0.1 and 0.01, which marked less edges but would still make the method unstable, although instabilities did take more times to appear. The supposed reason for instability was the declaration of two edges as border edges during a step while they were very close next to each other. They could even sometimes be crossing each other: this was due to the fact that local injectivity was not verified well enough, since keeping individual triangles from flipping is not enough to maintain local injectivity on a triangle soup. However, we want to keep the possibility for two adjacent triangles to have edges crossing each other, or else we would not get the degrees of freedom we achieved by using a triangle soup.

To maintain a form of local injectivity, we defined a proxy corresponding to an embedding of the surface using a vertex parameterization, with its vertices defined as the barycenter of its corresponding corners in the triangle soup. The linesearch was then modified to prevent triangle flip on this proxy. We then have to guarantee that the distorsion parametrization is compatible with this modified linesearch: this means that the energy used must diverge when one of the proxy triangle degenerates. We replace the distorsion energy from just the distorsion on the triangle soup to the sum of this energy and the distorsion on the proxy. Keep in mind that it is not possible to give up the triangle soup energy, as it is the one that helps us create some space between triangles in the soup, which we need in order to have a non null gradient (else v is equal to 1 everywhere).

The proxy can be represented as a matrix P of size n_v by $3n_f$, where every line corresponds to a vertex and has non null coefficient for every corner it is associated to. The value for each coefficient is $\frac{1}{d_i}$, where d_i is the degree of the i^{th} vertex. This matrix lets us pass from the corner positions to the proxy vertices positions. It also helps us pass from the gradient and the Hessian of the symmetric Dirichlet energy defined in the proxy from the proxy vertices space to the triangle soup corner space.

$$G_{proxy} = PG_{proxy}$$

$$H_{corner} = P^\top H_{proxy} P$$

In order to guarantee that when two edges are defined as border (in the context of enforcing boundary injectivity) they cannot cross each other, every time an edge is marked as a border its vertices are displaced onto their proxy positions. Then, in order to put some space between border vertices and edges and avoid two newly defined borders to superpose each other, each triangle is slightly retracted onto its barycenter. A few reasons however lead us to abandon this formulation:

- The stability problem was not fixed: the retraction does not always lead to a sufficient distance between vertices and edges, and the bijectivity energy still diverges. Furthermore, a too strong retraction would lead to an explosion of the distorsion term, and the resulting correction would push the vertices onto the edges we tried to keep them away from, making the energy at the direction taken diverge and thus breaking the linesearch.

- The new expression of the energy leads to a lot more non null coefficient in the matrix to inverse: every corner now is linked to every corner of the vertices neighboring its associated vertex, resulting in around d^2 non null coefficients by row in the hessian, with d being the average degree of the vertices. In addition to the computing time required for the bijectivity energy, this leads to computing time an order of magnitude longer than previously (reaching tens of minutes or even hours for surfaces with a few thousand faces).

We thus searched for an other way to find the cuts.

5.3.2 A second approach

A second approach consists in sampling v at faces instead, and u at vertices. This means we let some faces free to have as much distortion as needed in order to let other faces place themselves freely, acting as if there was a cut along these faces: on these faces, v is close to zero. We then have to cut alongside the distorted faces. These faces can be seen as being pulled on, the cuts then releasing the tension (see figure 5.10).

We can also notice that the gradient term in the Ambrosio-Tortorelli formula serves as a measure of distortion, acting as a weighted Dirichlet energy formulation ($|v\nabla\mathbf{u}|^2$). It then makes sense to see Ambrosio-Tortorelli as a process trying

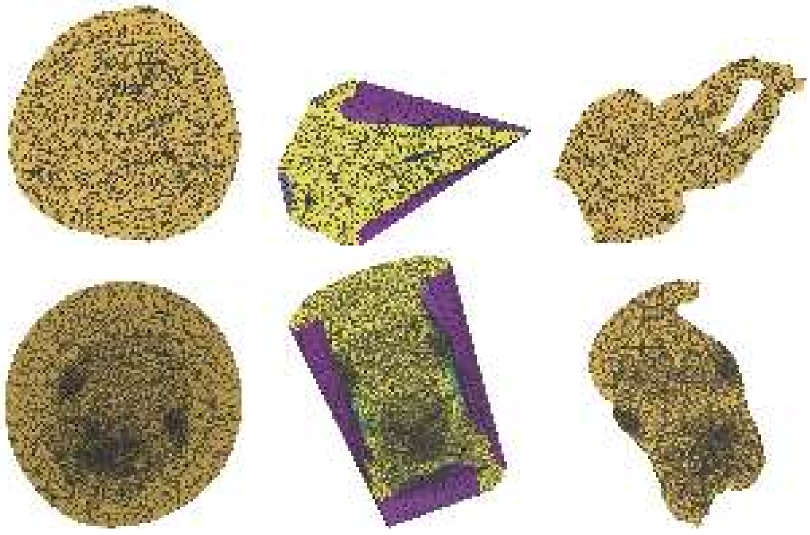


FIGURE 5.10: Parameterization at different stages of the method on the bunny head and face mesh: the initial parameterization (a, d), we then allow some faces (in magenta) to have distortion (b, e) so the other have more freedom of movement. We then compute cuts along these faces, giving us the result (c, f).

to reduce as much as possible this energy while allowing some exceptions: all we need then is a way to transform those exceptions into cuts. We also use Symmetric Dirichlet instead of Dirichlet, for local injectivity reasons, rescaled to be 0 when minimal. Since the gradient term already takes into account the distortion, and we do not have any data to be close to, we drop the first term of the functional, the fitting term.

The formulation below expresses the energy per triangle f:

$$
\begin{aligned}
E_f(\mathbf{u}, v) &:= A_f(v^2+\gamma)\underbrace{\left(\frac{|\nabla \mathbf{u}_f|^2 + |\nabla \mathbf{u}_f|^{-2}}{2} - 1\right)}_{\Psi_f(\mathbf{u})} + A_f\lambda\left(\epsilon|\nabla v_f|^2 + \frac{1}{4\epsilon}(1-v_f)^2\right) \\
&= A_f\left((v^2+\gamma)\Psi_f(\mathbf{u}) - \lambda\epsilon v_f(\Delta v_f) + \frac{\lambda}{4\epsilon}(1-v_f)^2\right), \qquad (5.11)
\end{aligned}
$$

where A_f denotes the area of face f, $\nabla \mathbf{u}_f$ is the usual constant gradient over a triangle of the linearly interpolated $\mathbf{u}$ function sampled at triangle vertices. Furthermore, v_f denotes simply the sampled value of function f on face f, $|\nabla v_f|^2$ is the squared norm of the gradient of v within f, whose integral over the face f is approximated as $-A_f v_f L v_f$, with Lv_f the DEC Laplacian of v on the dual mesh (see [Des+05]), γ is a fixed constant, necessary to avoid a null coefficient in front of the distortion term: it would lead to instability during the optimization. Results presented here use $\gamma = 10^{-7}$. Finally, $\Psi_f(\mathbf{u})$ is the distortion measure of the face f, and corresponds to the symmetric Dirichlet minus one, in order to have its minimum be 0.

Optimization

Optimizing this functional is done by alternating two steps, as seen previously: one step has $\mathbf{u}$ fixed and one step has v fixed:

Model	`bunnyhead`	`camelhead`	`bimba`	`hand`	`cat`	`armadillo`	`duck1`	`planck1`	`venus`	`tooth`	`circular box`
OptCut's time	2.5s	89s	13s	0.60s	0.65s	29s	2.08s	**1.19s**	0.61s	25.4s	22s
Our time	**1.6s**	**38s**	**6s**	**0.13s**	**0.11s**	**6.1s**	**0.77s**	1.32s	**0.36s**	**2.1s**	**3.8s**
Initial distortion	9.10	7.45	7.64	10.04	6.67	10.3	5.59	9.03	8.78	4.26	5.28
OptCut's distortion	**4.36**	4.26	4.50	5.2	4.38	5.22	4.49	4.54	4.37	4.24	4.53
Our distortion	4.39	4.26	4.50	5.3	4.50	5.22	4.49	4.56	4.39	4.25	4.54
OptCut's cut length	**2.88**	2.93	2.01	**4.5**	**1.28**	**2.07**	**1.09**	**1.61**	**1.70**	0.24	**1.13**
Our cut length	3.82	**2.56**	**1.90**	5.6	1.76	4.25	1.31	2.33	2.94	**0.10**	1.50

TABLE 5.1: Quantitative comparison between OptCuts method and our method, in terms of computation time, distortion and cut length. The `bimba`, `hand`, `cat`, `armadillo`, `duck`, `planck`, `venus`, `tooth` and `circular box` meshes come from the available input mesh in OptCuts [Li+18] repository.

- At v fixed, the only non-constant term is the distortion term. We optimize this term using the quasi-Newton method from [SGK19]: we compute a modified Hessian (guaranteed to be SPD) of the distortion energy as well as its gradient. We can then compute how we want to modify the current parameterization, and we use a line search to avoid triangle flip (following [SS15], using the implementation provided by [JP+18]). We repeat until the gradient norm falls under a threshold.

- At $\mathbf{u}$ fixed, the expression is convex quadratic as seen for AT previously in Eq.(5.2). We can put the energy to optimize at this step under the form

$$E := v^\top M\Psi(\mathbf{u})v + \lambda\left(\frac{1}{4\epsilon}(v^\top Mv - 2Mv) - \epsilon v^\top Lv\right) + c\,, \tag{5.12}$$

  which is of the form $E = v^\top Hv + 2b^\top v + c$ (with c a constant term that can be ignored). Its minimum can be found at $-H^{-1}b$. We need the expression of H and b:

$$H = \frac{\lambda}{4\epsilon}M - \epsilon\lambda L + M\Psi(u), \quad b = -\frac{\lambda}{4\epsilon}M\mathbf{1}_n\,, \tag{5.13}$$

  where L is a face Laplacian matrix (we used the DEC Laplacian on the dual mesh as in [Des+05]), $\Psi(u)$ is a diagonal matrix containing the distortion of each face (the i-th diagonal term corresponds to the i-th face), M is a face mass matrix, for example containing the area of each face on the diagonal, and finally I_n is the identity $n_f \times n_f$ matrix and $\mathbf{1}_n$ is a n_f-row vector filled with ones.

While on the second step finding the minimum only requires one iteration, the quasi-Newton of the first step has to be iterated multiple times before convergence (sometimes hundreds of times), making it much more time consuming than the second step.

Cutting

The scalar function v is used as a support for the cut extraction. Indeed, for triangles with values close to 1, the distortion term of the energy fully applies and is minimized during the optimization. On the contrary, triangles with values close to zero correspond to discontinuities in the energy minimization and thus will be used to delineate the precise location of cuts (see Figure 5.12).

We thus need a way to convert this scalar information into sequences of edges corresponding to cuts. To do so, we employ a simple strategy: we first define patches of triangles as simply connected components of traingles with values v lower than

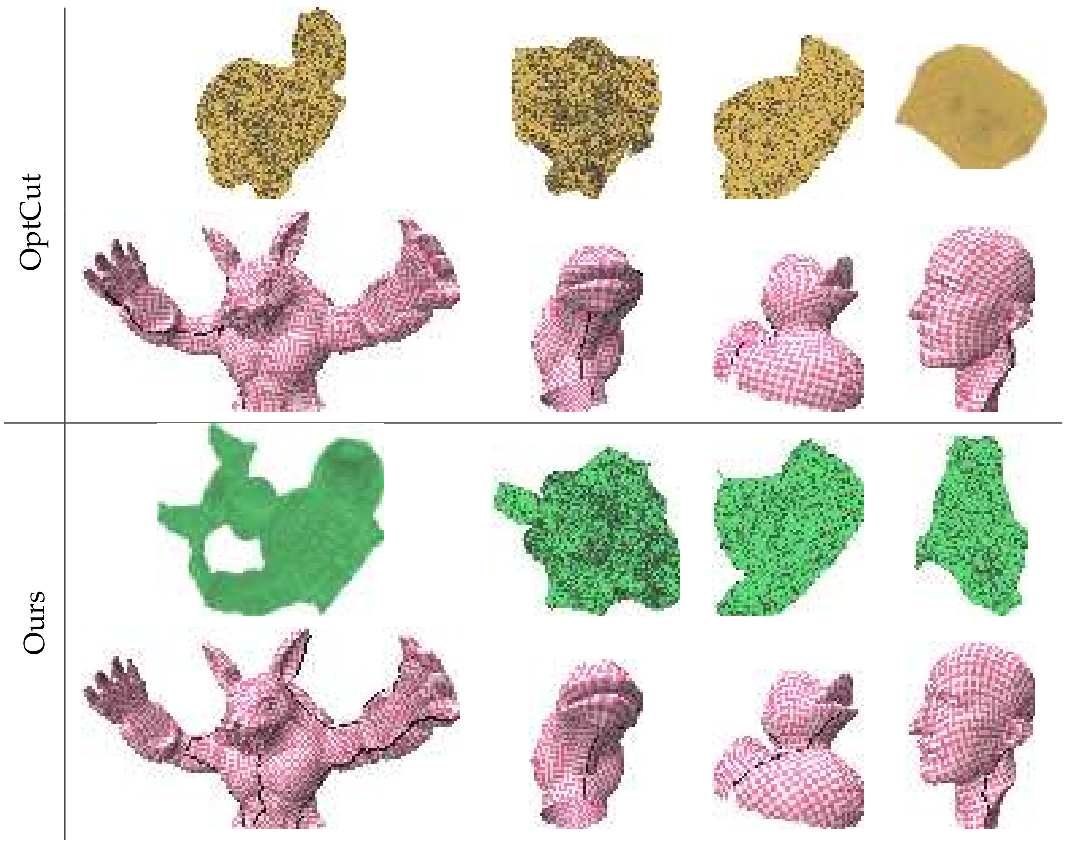


FIGURE 5.11: Visualization of the parameterizations evaluated in table 5.1 for the `armadillo`, the `camelhead`, the `duck` and `planck` meshes. Bottom row are our parameterizations, top are optcuts'.

0.5. Thanks to the Ambrosio-Tortorelli formulation with ϵ tending to zero, this simple process allows us to define thin strips of triangles. For each patch, we take the two farthest points belonging to this patch and find the shortest path between them: this defines the cut for this patch as a path of edges. The distance used is the distance on the underlying graph, with a weight of 0 for edges already belonging to the border. This trick forces the cut to go through the border if it is possible, to avoid having a cut parallel to an already existing border. Finally, we only keep cuts with lengths that are above 20% the length of the longest cuts, in order to avoid too small cuts.

This method was chosen for its simplicity and its effectiveness at capturing most of the areas to cut through without inducing too lengthy cuts, as seen on Figure 5.12. Depending on the application, one could be interested in iterating over this process: solve the AT functional to retrieve u and v, extract some long cuts, and iterate. However, in the following experiments, we only consider a single step for best performances.

Overall algorithm

Algorithm 4 gives an overall description of our method. This method can be run multiple times, for example if the cutting methods missed some places to cut. We can also use more complex energy when solving for u, for example, if we need to incorporate some border energy for border injectivity. This only changes the

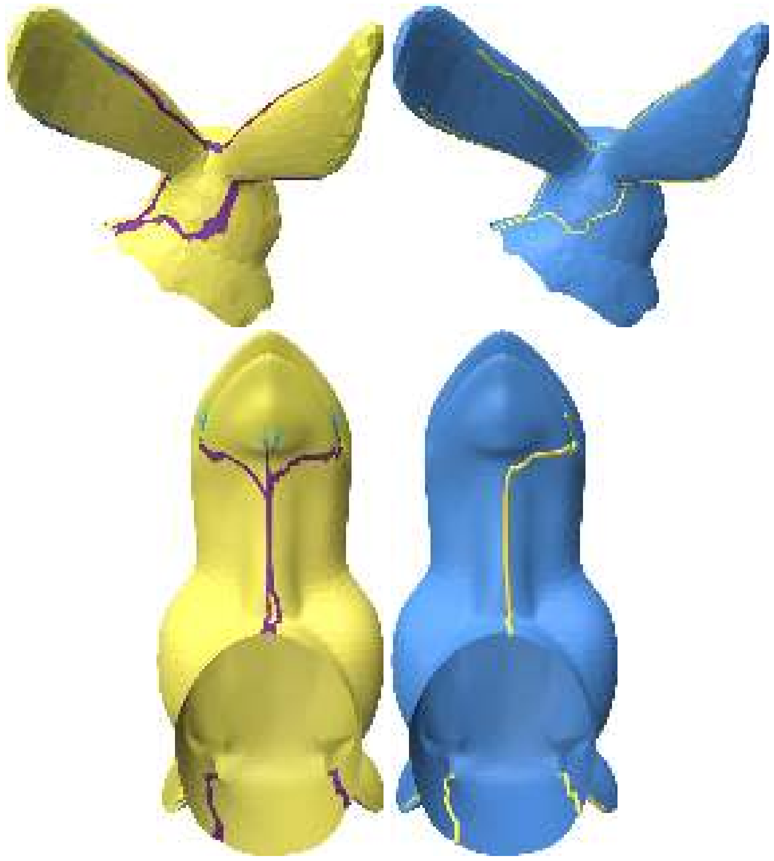


FIGURE 5.12: Our cutting method chooses the longest path among each patch of "deformable faces" (in magenta on (a) and (c)). While it may not capture all of the area suggested (such as on the `camelhead` (d)), it is quite effective at capturing most of it and may filter some artifacts such as the loop at the base of the neck of the camel.

`Projected_Newton_Solver` part, which needs to incorporate the Hessian and gradient of the additional energies.

5.4 Results

This section gathers several experiments that show the potential of our method for mesh texture mapping. We first compare our approach to the state-of-the-art OptCuts method [Li+18], both quantitatively and qualitatively. Then we examine the influence of the 4 parameters of our method. We finally show how our method can easily integrate user constraints in its formulation, like cutting in less visible zones or create a tighter UV map that fits a bounding box.

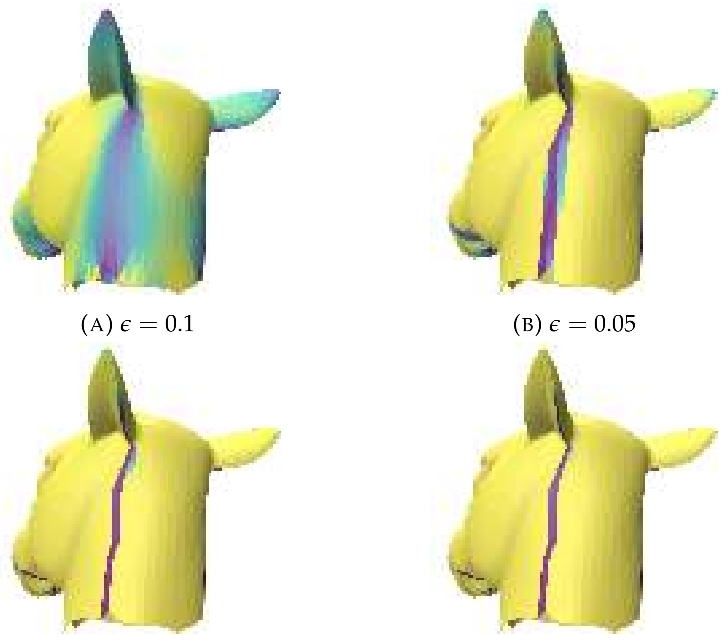
Note that the results presented here are of disk-like topology, as our method has a strong dependence on the initial cut. Indeed the method does not allow us to contest the edges included in the initial cut (we have not found a satisfying way to merge already cut edges). Furthermore the initial cut puts a strong constraint on the parameterization and consecutively on the evolution of v.

5.4.1 Influence of parameters

There are four parameters to tune: λ, ϵ_1, ϵ_2 and n the number of iterations between each change of ϵ (see the algorithm 2). We ended up using the following parameters

Algorithm 4: Pseudocode of our method for optimizing our model, which outputs the resulting parameterization

```
Data: $(V, F)$ a triangular mesh, $\epsilon_1$ the starting epsilon, $\epsilon_2$ the ending epsilon,
      $n$ a fixed number of optimization steps
Result: $U$ parameterization
Function Compute_AT_Parameterization(F, u)
    u ← Tutte(V, F);
    v ← $vec(1)$ ;
    $\epsilon \leftarrow \epsilon_1$ ;
    while $\epsilon > \epsilon_2$ do
        for $i \leftarrow 0$ to $n$ do
            u ← Projected_Newton_Solver( u, v) ;            // Compute
             parameterization using symmetric Dirichlet weighted by
             V
            v ← $H^{-1}b$ ;
        $\epsilon \leftarrow \epsilon/2$
    cuts ← Compute_Cuts( F, V, v) ;
    F, V ← Cut(F, V, cuts) ;
    u ← Projected_Newton_Solver( u) ;              // Compute final
     parameterization using non weighted symmetric Dirichlet
    return u;
Function Compute_Cuts( F, V, v)
    $S \leftarrow \{\}$;
    for $f$ in $F$ do
        if v (f) $< 0.5$ then
            $S \leftarrow S \cup \{f\}$;
    cuts ← $\{\}$;
    foreach $CC$ connected component in $S$ do
        $P1, P2$ ← Get_Farthest_Points( $CC$);
        cuts ← cuts ∪ Get_Shortest_Path( $CC$, $P1$, $P2$);
    return cuts;
```



(A) $\epsilon = 0.1$ (B) $\epsilon = 0.05$

(C) $\epsilon = 0.025$ (D) $\epsilon = 0.0125$

FIGURE 5.13: Evolution of v during the iterations of the function `Compute_AT_Parameterization` of Algorithm 4 as the ϵ decreases. Starting from a smooth discontinuity map (a), v becomes sharper as ϵ tends to zero.

by default: $\epsilon_1 = 0.1$, $\epsilon_2 = 0.01$ and $n = 3$. This results in 12 iterations of alternate optimization. We also use $\lambda = 1$.

Parameter λ can be considered as a cost for cuts and influence the final balance between distortion energy and the length of cuts. We choose $\lambda = 1$, but it may have to be changed depending on the number of cuts needed. Figure 5.15 illustrates the difference resulting from different λ. The three other parameters serve as convergence parameters: the wider the difference between ϵ_1 and ϵ_2, the more the problem starts from a diffuse and global point of view and ends at a local one. Parameter n governs how many iterations are necessary before convergence at a certain ϵ. Parameters ϵ_1 and ϵ_2 were chosen so that the starting problem is global and ends at a point where v seems reduced to one triangle wide lines along distorted faces, as seen on 5.13.

We choose $n = 3$, as it was the smallest value at which function v clearly distinguishes between rigid (v close to 1) and non-rigid faces (v close to 0). Higher values could be chosen in order to let v converge to a better solution. However, no significant difference was observed in tests with higher n. This could be due to the fact that, after ϵ changes its value for the first time, the number of steps for convergence is quite small (see figure 5.14): it seems that its impact happens at higher ϵ values and but ends up being erased with smaller ϵ values. This could also be due to the cutting method which is not refined enough to capture the difference a more precise v would make. It would make more sense to improve the cut before increasing the computation time significantly with a higher n.

We can also run the method several time to extend the cuts. Subsequent runs are usually faster than the initial ones, but the cuts added are also smaller. As seen on figure 5.16, they tend to prolong the initial choices instead of adding new isolated cuts.

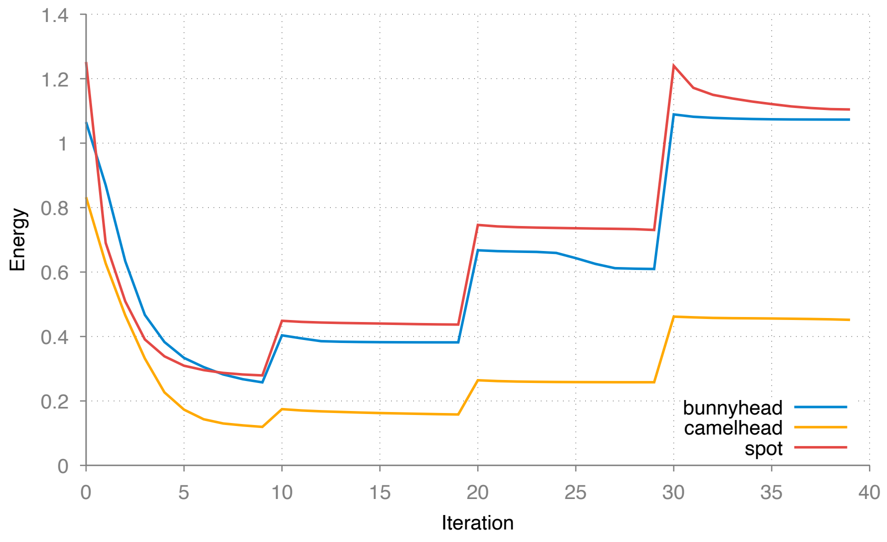


FIGURE 5.14: Evolution of our energy (defined in 5.11) when using $n = 10$ on various meshes (meaning ϵ changes every 10 iterations). After the first change of ϵ, it only takes a few iterations for the energy to converge.

5.4.2 Comparison to state-of-the-art methods

We choose to compare our method to the current implementation of OptCuts method [Li+18]. Note that it performs better than when described in the original paper due to some recent optimizations. We were unable to compare our method to AutoCuts [Por+17], since we failed to reproduce their results: the convergence speed was much too slow and was not producing results close to the ones displayed in their paper. After discussions with the author, we learned that their results were obtained using a proprietary solver, whose current version is not compatible with their code. However, it seems that the current implementation of OptCuts performs better than AutoCuts in most cases, so the comparison with OptCuts is a good test for our approach. Table 5.1 sums up parameterization results obtained by OptCuts and our approach, in terms of computation speed, final distortion and final cut length. We use OptCuts distortion measures and cut length, respectively:

$$E_d := \frac{1}{\sqrt{\sum_{t\in F} |A_t|}} \sum_{t\in F} |A_t|\Psi(t)\,, \quad E_s := \frac{1}{\sqrt{\sum_{t\in F} |A_t|}} \sum_{i\in S} |e_i|\,. \tag{5.14}$$

Our method almost always outperforms OptCuts in terms of speeds. However OptCuts is often better in terms of cut length for distortion reduction (better means less cut length for the lowest distortion). Since our method does not allow targeting for a fixed distortion (while OptCuts does), we used our method first and then set OptCuts targeting the same distortion value we obtained. Bear in mind that the lowest distortion that a mesh can attain is 4 as it is the minimum of Symmetric Dirichlet energy, so a change from 4.2 to 4.1 is actually quite significant.

5.4.3 Assisted cutting

We wish to take into account some user input when choosing cuts. For example, some edges may be favored over others based on their visibility (less visible edges are preferred). To take this constraint into account, we can define λ as a scalar over

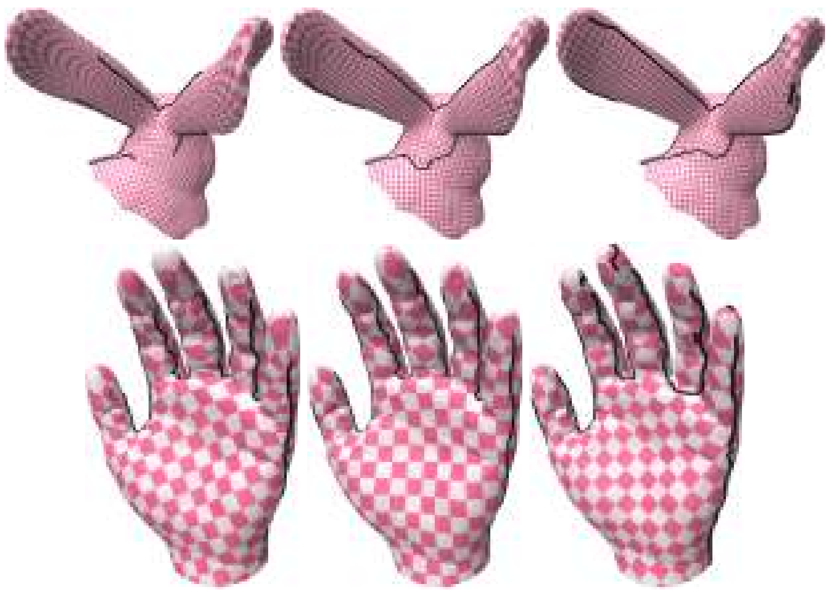


FIGURE 5.15: Results on the `bunnyhead` and `hand` meshes for $\lambda = \{3, 1, 0.1\}$ (reps. from left to right). Lower λ parameter implies longer cuts.

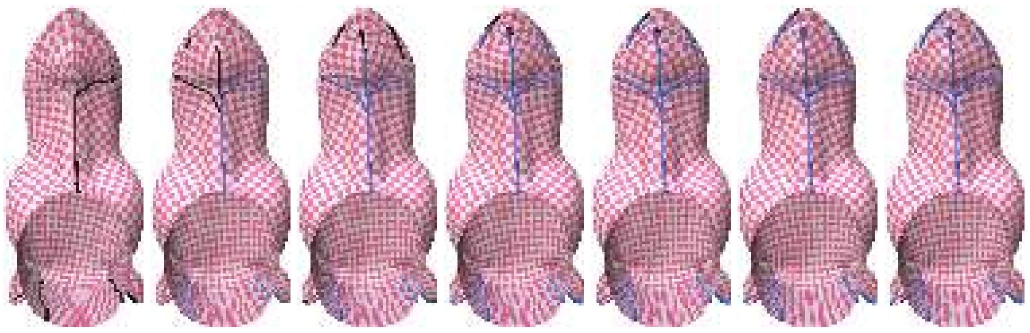


FIGURE 5.16: Results after multiple runs of our algorithm. This enables us to capture some natural cuts that our algorithm may have missed. After a few runs cuts our method converges, and the cuts stop expanding.

the surface instead of a constant, and with a value depending on the user input. We tested this approach using visibility computed with `libigl` ambient occlusion ([JP+18]). Figure 5.17 illustrates how it affects the resulting distribution of v: areas to cut are more prone to be placed in low visibility areas such as cracks.

5.4.4 Packing constraints for the atlas

It may be desirable for a UV map to be as tight as possible in order to take less memory space. We follow Autocuts method [Por+17] for fitting a parameterization within a predefined bounding rectangle. Figure 5.18 shows how UV map are then much more compact. However we observe a significant increase in the convergence time, depending on how tight the bounding box is: thee harder it is to fit the parameterization in the box, the longer it takes (sometimes taking 10 times as long as without).

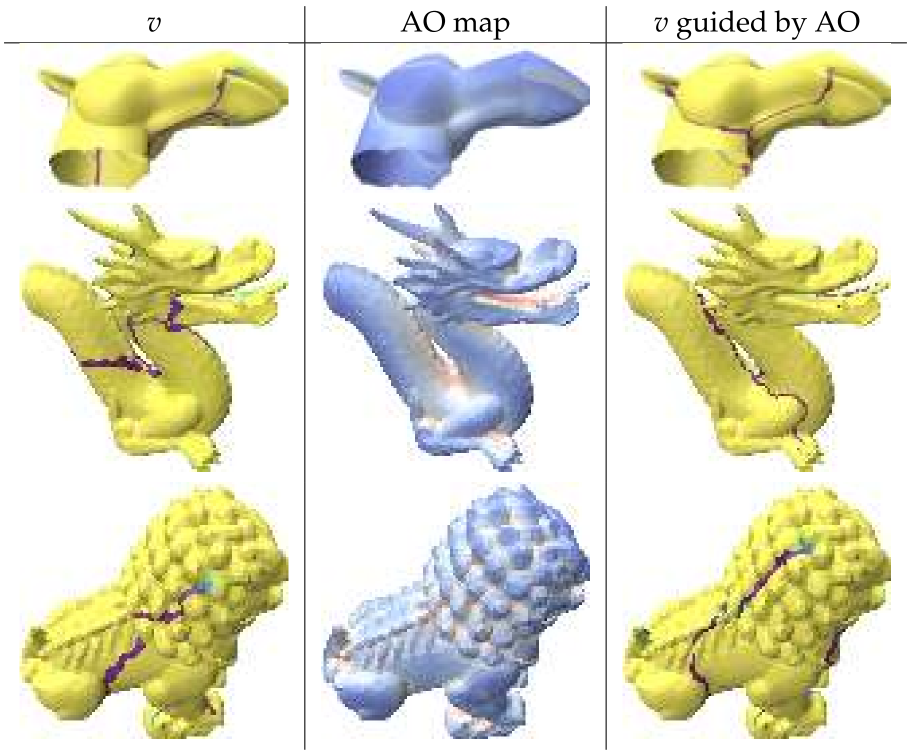


FIGURE 5.17: Optimized function v without (left) and with (right) the ambient occlusion (AO) taken into account (middle). When ambient occlusion is enabled, the faces identified as cut loci are guided to regions with lower visibility.

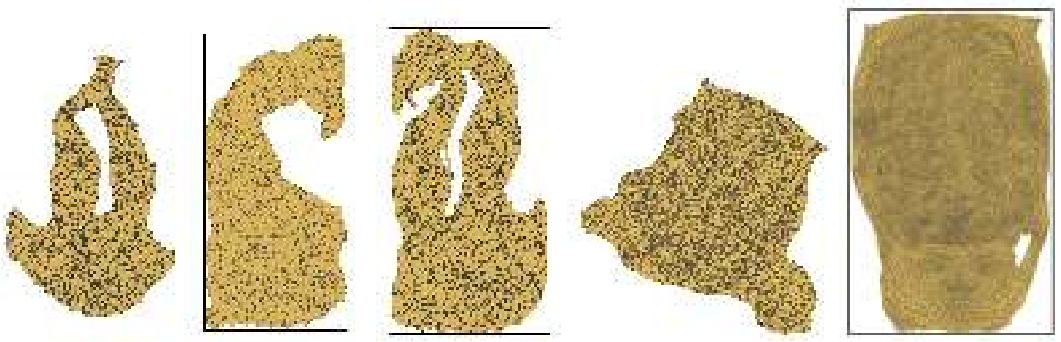


FIGURE 5.18: Resulting parameterizations without bounding box constraints (a, d) and with bounding box (b, c, e) on the `bunnyhead` and `david` mesh. The first example with `bunnyhead` (b) does not have border injectivity restrictions, enabling the many overlaps which are not present on the second example (c) (which has border injectivity restrictions). On the `david` mesh, an overlap free parameterization was obtained (e) without using our border injectivity restrictions, giving the same result with our without them.

5.5 Discussion and perspectives

We have presented a two new variationals model to compute a UV parameterization of a triangle mesh with a joint optimization of distortion and cuts. They are a discrete variants of the Ambrosio-Tortorelli functional, which offers theoretical guarantees for solving problems involving a penalization in the length of discontinuities. We have shown that the second approach is effective in practice, faster than other simultaneous optimization methods, with quality often close to the state-of-the-art Optcuts method, and incorporates easily other constraints in its formulation like border injectivity, assisted cutting or compact texture maps. We can also note that the loci of deformable faces may have an arbitrary topology, so cuts can take various shapes. Finally, the code is available at `https://github.com/Lieunoir/UV-AT`.

In order to measure the distortion, we only tested the Symmetric Dirichlet energy, which was the closest to the original gradient term in the Ambrosio Tortorelli functional. As future works, other energies could be evaluated (such as ARAP), as they may not behave in the same way as Symmetric Dirichlet.

A limitation of the current cutting method when using the second discretization, compared to the first, is that it does not preserve the topology defined by v: if any cycle is present, we ignore it. It also does not capture all of the region defined by the underlying patch (if the region is a tree it will only choose the longest branch), meaning we have to run the method again to capture it all. Our current cutting method is well adapted to a semi-supervised setting where the user can decide to refine further, but cuts following the topology induced by v could indeed be interesting in an unsupervised model.

In some cases, our method is not able to run properly because the energy diverges in the initialization. This problem comes from numerical issues of floating-point arithmetic. It is addressed by Shen et al. [She+19], giving a alternative to Tutte's embedding for initialization. We have not run our method with this initialization, but we can safely assume that it does not affect significantly the results.

One way to improve the convergence speed would be to include some speedup optimization techniques. In particular, we could use the methods proposed by Liu et al. [Liu+18] or [Liu+21]. We could also look for other ways to optimize the Mumford-Shah functional that would yield better results or give them faster: our alternate Ambrosio-Tortorelli optimization method is among the costliest ones. For instance, the recent competitive gradient and mirror descent algorithms [SA19; SAO20] seem to be particularly efficient in optimizing this type of functional, which are block-convex but not globally convex.

Conclusion

We have seen that the Laplace-Betrami operator can be built on digital surfaces, using a corrected normal field. Our experiments suggest that the operator is indeed convergent (at least in a weak sense): however, we have not provided a theoretical guarantee of this result. Since studies of the finite element method provide proofs for the weak convergence on triangle surfaces, and we have provided a finite element based method for its construction on digital surfaces, we could hope that a proof for the convergence on digital surfaces can be derived from the existing ones. One of the hard part may reside around the diffusion added to the process when computing the Laplacian of a function: this may be due to the fact that values are not sampled on the surface but around it, and the diffusion serves as a mean to smooth the prolongation of the function defined on the surface to ambient space. However, it is unclear how it may play a role in a formal proof of convergence.

Other discretization schemes could be explored. Notably, the finite element method can be used with second order basis, or with other element basis functions (such as Crouzeix-Raviart for functions valued on edges). It could then be explored how much the precision of the corrected surface matters compared to the precision of the element, for example if the use of second order basis on faces with a constant normal is useful. Then, the use of these methods for multi scales digital surfaces can be explored, by using polygonal operators or using higher order elements at the frontier between two scale levels.

We could also study more the use of these operators on normal maps. Our model of projection is not exactly the one used in normal maps: we assume that the projection operator is orthogonal to the real surface, while in normal maps the projection operator is orthogonal to the coarse surface. Interactions around sampling, mipmapping should have to be studied in order to avoid having to create an operator the same size as the texture map, and also perhaps as a way of working with multi scaling. The construction and use of these operators on the GPU could be studied as well, perhaps allowing the use of some methods in real-time contexts for effects such as deformation.

Using a corrected Laplace-Beltrami operator along with normal and mean curvature estimators, we have built a surface regularization method. We have explored the use of the method on triangle meshes with normal map, and the addition of a curvature edition step. The curvature edition could be further studied: in particular, we make no guarantee that the curvature that we try to obtain can actually be obtained from a real shape. Thus our strategy is one of finding something that approximates this: instead, we could try to make sure that the target curvature corresponds to the one of a real object. Several other uses for corrected operators could also be explored, such as shape matching for digital surfaces using functional maps.

As an example of the importance of the discretization, we have presented two ways to adapt the Ambrosio-Tortorelli functional for the generation of UV maps: one using a corner/edge discretization, which while being a natural way to obtain a parameterization and some cuts from the optimization process, was quite slow and was hard to adapt to current method to prevent overlaps. Another one based on vertex/face discretization ended up being more thoroughly investigated, as it was both faster than the first discretization and than other simultaneous optimization methods.

Both of the presented methods could benefit from faster distorsion solvers. Some works in this direction include Progressive Parameterizations [Liu+18], where the distorsion of highly distorted triangle is artificially bounded by changing the reference triangle to an intermediate one instead, and where the distorsion optimization switches between SLIM [Rab+17] and Composite Majorization [Sht+17] depending on the level of distorsion. It is however unclear how the first method (whose enegy include two terms, not juste the "pure" distorsion one) could benefit from it. For the second method, it is unsure if the benefits would be high, since some triangles need to have high variations of their shape in order for our optimization to converge quickly. Perhaps since these triangles usually have low weights, their choice of hessian does not matter and could simply not being taken into account for the choice of strategy used.

One of the parts of second method that could be improved is the cutting method: right now the method sometime is limited by the topology of the mesh until the cuts are done (figure 5.19). However, since we don't have a way to go back onto choices of cut, we cannot make these choices too early. A way to cirvumvent this problem could be to find ways to subdivide some triangles in order to enable finer displacement of triangles. We should be careful about these subdivision however, since adding triangle will add variables and slow the system, and changing the topology will require some recomputations of the operators and of the symbolic parts of the solvers.

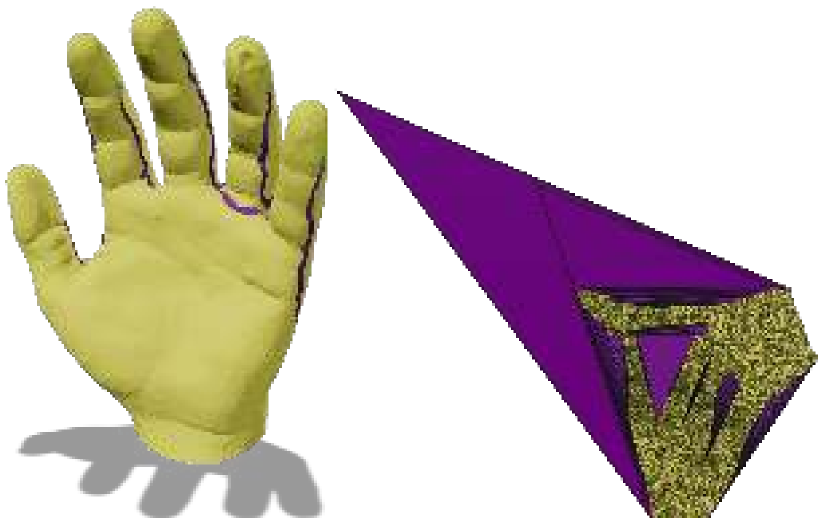


FIGURE 5.19: Parameterization of the hand mesh: some triangles are particularly stretched to allow for better placements of the other triangles. We could benefit from modifying the toplogy of the shape, perhaps by subdividing some triangles, in order to better allow the triangles around to reposition and represent more closely their placement after a cut would be added next to the triangle.

Publications

The works described in chapter 3 around corrected operators were presented in the following conferences and seminar:

- LAMA (Laboratoire de Mathématiques) LIMD team seminar
- DGMM2024 (Discrete Geometry and Mathematical Morphology) [WCL24]

The works described in chapter 5 around UV parameterization were presented in the following conferences and seminar:

- CoMeDiC 2021 (Convergent Metrics for Digital Calculus)
- JFIG 2021 (Journées Françaises de l'Informatique Graphique)
- SMI 2023 (Shape Modeling International) [Wei+23]

Bibliography

[ASC11] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. "The Wave Kernel Signature: A Quantum Mechanical Approach to Shape Analysis". In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011, pp. 1626–1633. DOI: 10.1109/ICCVW.2011.6130444.

[AT90] Luigi Ambrosio and Vincenzo Maria Tortorelli. "Approximation of Functional Depending on Jumps by Elliptic Functional via Γ-Convergence". In: *Communications on Pure and Applied Mathematics* 43.8 (1990), pp. 999–1036. DOI: https://doi.org/10.1002/cpa.3160430805.

[AW11] Marc Alexa and Max Wardetzky. "Discrete Laplacians on General Polygonal Meshes". In: *ACM Trans. Graph.* 30.4 (July 2011). ISSN: 0730-0301. DOI: 10.1145/2010324.1964997. URL: https://doi.org/10.1145/2010324.1964997.

[BB23] A. Bunge and M. Botsch. "A Survey on Discrete Laplacians for General Polygonal Meshes". In: *Computer Graphics Forum* 42.2 (2023), pp. 521–544.

[BBA21] A. Bunge, M. Botsch, and M. Alexa. "The Diamond Laplace for Polygonal and Polyhedral Meshes". In: *Computer Graphics Forum* 40.5 (2021), pp. 217–230. DOI: https://doi.org/10.1111/cgf.14369. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14369. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14369.

[BEL89] TED BELYTSCHKO. "The Finite Element Method: Linear Static and Dynamic Finite Element Analysis: Thomas J. R. Hughes". In: *Computer-Aided Civil and Infrastructure Engineering* 4.3 (1989), pp. 245–246. DOI: https://doi.org/10.1111/j.1467-8667.1989.tb00025.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8667.1989.tb00025.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8667.1989.tb00025.x.

[Bom+13] David Bommes, Bruno Levy, Nico Pietroni, Enrico Puppo, Cláudio Silva, and Denis Zorin. "Quad-Mesh Generation and Processing: A Survey". In: *Computer Graphics Forum* 32 (Sept. 2013). DOI: 10.1111/cgf.12014.

[Bon+18] Nicolas Bonneel, David Coeurjolly, Pierre Gueth, and Jacques-Olivier Lachaud. "Mumford-Shah Mesh Processing using the Ambrosio-Tortorelli Functional". In: *Computer Graphics Forum (Proceedings of Pacific Graphics)* 37.7 (Oct. 2018). DOI: 10.1111/cgf.13549.

[Bro+11] Alexander M. Bronstein, Michael M. Bronstein, Leonidas J. Guibas, and Maks Ovsjanikov. "Shape Google: Geometric Words and Expressions for Invariant Shape Retrieval". In: *ACM Trans. Graph.* 30.1 (Feb. 2011). ISSN: 0730-0301. DOI: 10.1145/1899404.1899405. URL: https://doi.org/10.1145/1899404.1899405.

[Bro+21] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *arXiv preprint arXiv:2104.13478* (2021).

[BS07] Alexander I. Bobenko and Boris A. Springborn. "A Discrete Laplace—Beltrami Operator for Simplicial Surfaces". In: *Discrete Comput. Geom.* 38.4 (Dec. 2007), pp. 740–756. ISSN: 0179-5376.

[BSW08] Mikhail Belkin, Jian Sun, and Yusu Wang. "Discrete Laplace Operator on Meshed Surfaces". In: *Proceedings of the Twenty-Fourth Annual Symposium on Computational Geometry*. SCG '08. College Park, MD, USA: Association for Computing Machinery, 2008, pp. 278–287. ISBN: 9781605580715. DOI: 10.1145/1377676.1377725. URL: https://doi.org/10.1145/1377676.1377725.

[BSW09] Mikhail Belkin, Jian Sun, and Yusu Wang. "Constructing Laplace Operator from Point Clouds in Rd". In: *ACM-SIAM Symposium on Discrete Algorithms*. 2009. URL: https://api.semanticscholar.org/CorpusID:11225723.

[Bud+20] Max Budninskiy, Ameera Abdelaziz, Yiying Tong, and Mathieu Desbrun. "Laplacian-Optimized Diffusion for Semi-Supervised Learning". In: *Computer Aided Geometric Design* 79 (2020), p. 101864. ISSN: 0167-8396. DOI: https://doi.org/10.1016/j.cagd.2020.101864. URL: https://www.sciencedirect.com/science/article/pii/S0167839620300510.

[Bun+20] Astrid Bunge, Philipp Herholz, Misha Kazhdan, and Mario Botsch. "Polygon Laplacian Made Simple". In: *Computer Graphics Forum* 39.2 (2020), pp. 303–313. DOI: https://doi.org/10.1111/cgf.13931. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13931. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13931.

[Bun+24] A. Bunge, D. R. Bukenberger, S. D. Wagner, M. Alexa, and M. Botsch. "Polygon Laplacian Made Robust". In: *Computer Graphics Forum* 43.2 (2024), e15025. DOI: https://doi.org/10.1111/cgf.15025. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.15025. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.15025.

[Cai+19] Thomas Caissard, David Coeurjolly, Jacques-Olivier Lachaud, and Tristan Roussillon. "Laplace–Beltrami Operator on Digital Surfaces". In: *Journal of Mathematical Imaging and Vision* 61.3 (2019), pp. 359–379.

[Cao+20] Wenming Cao, Zhiyue Yan, Zhiquan He, and Zhihai He. "A Comprehensive Survey on Geometric Deep Learning". In: *IEEE Access* 8 (2020), pp. 35929–35949. DOI: 10.1109/ACCESS.2020.2975067.

[CG16] Renjie Chen and Craig Gotsman. "On Pseudo-Harmonic Barycentric Coordinates". In: *Computer Aided Geometric Design* 44 (2016), pp. 15–35. ISSN: 0167-8396. DOI: `https://doi.org/10.1016/j.cagd.2016.04.005`. URL: `https://www.sciencedirect.com/science/article/pii/S0167839616300462`.

[Cha+10] Isaac Chao, Ulrich Pinkall, Patrick Sanan, and Peter Schröder. "A Simple Geometric Model for Elastic Deformations". In: *ACM Trans. Graph.* 29.4 (July 2010). ISSN: 0730-0301. DOI: `10.1145/1778765.1778775`.

[CL22] David Coeurjolly and Jacques-Olivier Lachaud. "A Simple Discrete Calculus for Digital Surfaces". In: *IAPR Second International Conference on Discrete Geometry and Mathematical Morphology*. Ed. by Étienne Baudrier, Benoît Naegel, Adrien Krähenbühl, and Mohamed Tajine. Springer, LNCS, Oct. 2022.

[CLG21] David Coeurjolly, Jacques-Olivier Lachaud, and Pierre Gueth. "Digital Surface Regularization With Guarantees". In: *IEEE Trans. Vis. Comput. Graph.* 27.6 (2021), pp. 2896–2907.

[CLL14] D. Coeurjolly, J.-O. Lachaud, and J. Levallois. "Multigrid Convergent Principal Curvature Estimators in Digital Geometry". In: *Computer Vision and Image Understanding* 129 (2014). Special section: Advances in Discrete Geometry for Computer Imagery, pp. 27–41. ISSN: 1077-3142. DOI: `http://dx.doi.org/10.1016/j.cviu.2014.04.013`. URL: `http://www.sciencedirect.com/science/article/pii/S1077314214001003`.

[CLR12] D. Coeurjolly, J.-O. Lachaud, and T. Roussillon. "Multigrid Convergence of Discrete Geometric Estimators". English. In: *Digital Geometry Algorithms*. Ed. by Valentin E. Brimkov and Reneta P. Barneva. Vol. 2. Lecture Notes in Computational Vision and Biomechanics. Springer Netherlands, 2012, pp. 395–424. ISBN: 978-94-007-4173-7. DOI: `10.1007/978-94-007-4174-4_13`. URL: `http://dx.doi.org/10.1007/978-94-007-4174-4_13`.

[Coe+16] David Coeurjolly, Marion Foare, Pierre Gueth, and Jacques-Olivier Lachaud. "Piecewise Smooth Reconstruction of Normal Vector Field on Digital Data". In: *Computer Graphics Forum (Proceedings of Pacific Graphics)* 35.7 (Sept. 2016). DOI: `10.1111/cgf.13013`.

[CPS13] Keenan Crane, Ulrich Pinkall, and Peter Schröder. "Robust Fairing via Conformal Curvature Flow". In: *ACM Trans. Graph.* 32.4 (2013).

[Cue+15] L. Cuel, J.-O. Lachaud, Q. Mérigot, and B. Thibert. "Robust Geometry Estimation Using the Generalized Voronoi Covariance Measure". In: *SIAM Journal on Imaging Sciences* 8.2 (2015), pp. 1293–1314. DOI: `10.1137/140977552`. eprint: `http://dx.doi.org/10.1137/140977552`. URL: `http://dx.doi.org/10.1137/140977552`.

[CWW17] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. "The Heat Method for Distance Computation". In: *Commun. ACM* 60.11 (Oct. 2017), pp. 90–99. ISSN: 0001-0782.

[DBD20] Fernando De Goes, Andrew Butts, and Mathieu Desbrun. "Discrete Differential Operators on Polygonal Meshes". In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 110–1.

[DE13] Gerhard Dziuk and Charles M. Elliott. "Finite Element Methods for Surface PDEs". In: *Acta Numerica* 22 (2013), pp. 289–396. DOI: 10.1017/S0962492913000056.

[Des+05] Mathieu Desbrun, Anil N Hirani, Melvin Leok, and Jerrold E Marsden. "Discrete Exterior Calculus". In: *arXiv preprint math/0508341* (2005).

[Des+99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. "Implicit Fairing of Irregular Meshes using Diffusion and Curvature flow". In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 317–324. ISBN: 0201485605. DOI: 10.1145/311535.311576. URL: https://doi.org/10.1145/311535.311576.

[DO05] Komla Domelevo and Pascal Omnes. "A Finite Volume Method for the Laplace Equation on Almost Arbitrary Two-Dimensional Grids". en. In: *ESAIM: Modélisation mathématique et analyse numérique* 39.6 (2005), pp. 1203–1249. DOI: 10.1051/m2an:2005047. URL: http://www.numdam.org/articles/10.1051/m2an:2005047/.

[Don+05] Shen Dong, Peer-Timo Bremer, Michael Garland, Valerio Pascucci, and John Hart. "Quadrangulating a Mesh using Laplacian Eigenvectors". In: (Jan. 2005).

[DUS55] G. M. DUSINBERRE. "Heat Transfer Calculations By Numerical Methods". In: *Journal of the American Society for Naval Engineers* 67.4 (1955), pp. 991–1002. DOI: https://doi.org/10.1111/j.1559-3584.1955.tb03171.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1559-3584.1955.tb03171.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1559-3584.1955.tb03171.x.

[EG04] Alexandre Ern and Jean-Luc Guermond. New York, NY: Springer New York, 2004. ISBN: 978-1-4757-4355-5. DOI: 10.1007/978-1-4757-4355-5_3. URL: https://doi.org/10.1007/978-1-4757-4355-5_3.

[EGH00] Robert Eymard, Thierry Gallouët, and Raphaèle Herbin. "Finite Volume Methods". In: *Solution of Equation in Rn (Part 3), Techniques of Scientific Computing (Part 3)*. Vol. 7. Handbook of Numerical Analysis. Elsevier, 2000, pp. 713–1018. DOI: https://doi.org/10.1016/S1570-8659(00)07005-8. URL: https://www.sciencedirect.com/science/article/pii/S1570865900070058.

[FC24] Nicole Feng and Keenan Crane. "A Heat Method for Generalized Signed Distance". In: *ACM Trans. Graph.* 43.4 (2024).

[Fli+05] F. Flin, J. B. Brzoska, B. Lesaffre, C. ColÃ©ou, P. Lamboley, D. Coeurjolly, O. Teytaud, G. Vignoles, and J. F. Delesse. "An Adaptive Filtering Method to Evaluate Normal Vectors and Surface Areas of 3D Objects. Application to Snow Images from X-Ray Tomography". In: *IEEE Transactions on Image Processing* 14.5 (2005), pp. 585–596.

[Flo03] Michael S. Floater. "Mean Value Coordinates". In: *Computer Aided Geometric Design* 20.1 (2003), pp. 19–27. ISSN: 0167-8396. DOI: https://doi.org/10.1016/S0167-8396(03)00002-5. URL: https://www.sciencedirect.com/science/article/pii/S0167839603000025.

[GW96] Carolyn Gordon and David Webb. "You Can't Hear the Shape of a Drum". In: *American Scientist* 84.1 (Jan. 1996), pp. 46–55.

[Her00] F. Hermeline. "A Finite Volume Method for the Approximation of Diffusion Operators on Distorted Meshes". In: *Journal of Computational Physics* 160.2 (2000), pp. 481–499. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.2000.6466. URL: https://www.sciencedirect.com/science/article/pii/S0021999100964660.

[HF06] Kai Hormann and Michael S. Floater. "Mean Value Coordinates for Arbitrary Planar Polygons". In: *ACM Trans. Graph.* 25.4 (Oct. 2006), pp. 1424–1441. ISSN: 0730-0301. DOI: 10.1145/1183287.1183295. URL: https://doi.org/10.1145/1183287.1183295.

[HG12] Kai Hormann and Guenther Greiner. "MIPS: An Efficient Global Parametrization Method". In: *Curve and Surface Design: Saint-Malo* 2000 (Nov. 2012), p. 10.

[Hir03] Anil Nirmal Hirani. "Discrete Exterior Calculus". AAI3086864. PhD thesis. USA, 2003.

[HP11] Klaus Hildebrandt and Konrad Polthier. "On Approximation of the Laplace-Beltrami Operator and the Willmore Energy of Surfaces". In: *Computer Graphics Forum* (2011). ISSN: 1467-8659.

[HS08] K. Hormann and N. Sukumar. "Maximum Entropy Coordinates for Arbitrary Polytopes". In: *Computer Graphics Forum* 27.5 (2008), pp. 1513–1520. DOI: https://doi.org/10.1111/j.1467-8659.2008.01292.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2008.01292.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01292.x.

[JP+18] Alec Jacobson, Daniele Panozzo, et al. *libigl: A Simple C++ Geometry Processing Library*. https://libigl.github.io/. 2018.

[JSP17] Zhongshi Jiang, Scott Schaefer, and Daniele Panozzo. "Simplicial Complex Augmentation Framework for Bijective Maps". In: *ACM Trans. Graph.* 36.6 (Nov. 2017). ISSN: 0730-0301. DOI: 10.1145/3130800.3130895.

[JZ07] Varun Jain and Hao Zhang. "A Spectral Approach to Shape-Based Retrieval of Articulated 3D Models". In: *Computer-Aided Design* 39.5 (2007). Geometric Modeling and Processing 2006, pp. 398–407. ISSN: 0010-4485. DOI: https://doi.org/10.1016/j.cad.2007.02.009. URL: https://www.sciencedirect.com/science/article/pii/S0010448507000504.

[Kac66] Mark Kac. "Can One Hear the Shape of a Drum?" In: *The american mathematical monthly* 73.4P2 (1966), pp. 1–23.

[KR04a] Reinhard Klette and Azriel Rosenfeld. *Digital Geometry*. Apr. 2004.

[KR04b] Reinhard Klette and Azriel Rosenfeld. *Digital Geometry: Geometric Methods for Digital Picture Analysis*. Morgan Kaufmann, 2004.

[Lac+20] Jacques-Olivier Lachaud, Pascal Romon, Boris Thibert, and David Coeurjolly. "Interpolated Corrected Curvature Measures for Polygonal Surfaces". In: *Comput. Graph. Forum* 39.5 (2020), pp. 41–54.

[LCL17] Jacques-Olivier Lachaud, David Coeurjolly, and Jérémy Levallois. “Robust and Convergent Curvature and Normal Estimators with Digital Integral Invariants”. In: *Modern Approaches to Discrete Curvature*. Ed. by Pascal Romon Laurent Najman. Vol. 2184. Lecture Notes in Mathematics. Springer-Verlag, 2017.

[Lee12] John M. Lee. New York, NY: Springer New York, 2012. ISBN: 978-1-4419-9982-5. DOI: 10.1007/978-1-4419-9982-5_1. URL: https://doi.org/10.1007/978-1-4419-9982-5_1.

[Lév+02] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. “Least Squares Conformal Maps for Automatic Texture Atlas Generation”. In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 362–371. ISSN: 0730-0301. DOI: 10.1145/566654.566590.

[LH14] Chunyuan Li and A. Ben Hamza. “Spatially Aggregating Spectral Descriptors for Nonrigid 3D Shape Retrieval: a Comparative Survey”. In: *Multimedia Syst.* 20.3 (2014), pp. 253–281. DOI: 10.1007/s00530-013-0318-0. URL: http://dx.doi.org/10.1007/s00530-013-0318-0.

[Li+18] Minchen Li, Danny M. Kaufman, Vladimir G. Kim, Justin Solomon, and Alla Sheffer. “OptCuts: Joint Optimization of Surface Cuts and Parameterization”. In: *ACM Trans. Graph.* 37.6 (Dec. 2018). ISSN: 0730-0301. DOI: 10.1145/3272127.3275042.

[Lia+11] Z Lian, A Godil, B Bustos, M Daoudi, J Hermans, S Kawamura, Y Kurita, G Lavoua, P Dp Suetens, et al. “Shape Retrieval on Non-Rigid 3D Watertight Meshes”. In: *Eurographics workshop on 3d object retrieval (3DOR)*. Citeseer. 2011.

[Liu+18] Ligang Liu, Chunyang Ye, Ruiqi Ni, and Xiao-Ming Fu. “Progressive Parameterizations”. In: *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: 10.1145/3197517.3201331.

[Liu+20] Zheng Liu, Weina Wang, Saishang Zhong, Bohong Zeng, Jinqin Liu, and Weiming Wang. “Mesh Denoising via a Novel Mumford-Shah Framework”. In: *Computer-Aided Design* 126 (May 2020), p. 102858. DOI: 10.1016/j.cad.2020.102858.

[Liu+21] Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. “Surface Multigrid via Intrinsic Prolongation”. In: *ACM Trans. Graph.* 40.4 (July 2021). ISSN: 0730-0301. DOI: 10.1145/3450626.3459768.

[LMS14] Konstantin Lipnikov, Gianmarco Manzini, and Mikhail Shashkov. “Mimetic Finite Difference Method”. In: *Journal of Computational Physics* 257 (2014). Physics-compatible numerical methods, pp. 1163–1227. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2013.07.031. URL: https://www.sciencedirect.com/science/article/pii/S0021999113005135.

[Lom+13] Herve Lombaert, Leo Grady, Jonathan R. Polimeni, and Farida Cheriet. “FOCUSR: Feature Oriented Correspondence Using Spectral Regularization–A Method for Precise Surface Matching”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.9 (2013), pp. 2143–2160. DOI: 10.1109/TPAMI.2012.276.

[LRT22] Jacques-Olivier Lachaud, Pascal Romon, and Boris Thibert. “Corrected Curvature Measures”. In: *Discrete & Computational Geometry* 68.2 (2022), pp. 477–524.

[LT16] Jacques-Olivier Lachaud and Boris Thibert. "Properties of Gauss Digitized Shapes and Digital Surface Integration". In: *Journal of Mathematical Imaging and Vision* 54.2 (2016), pp. 162–180.

[LV14] P. Lobaz and L. Váša. "Hierarchical Laplacian-Based Compression of Triangle Meshes". In: *Graphical Models* 76.6 (2014), pp. 682–690. ISSN: 1524-0703. DOI: https://doi.org/10.1016/j.gmod.2014.09.003. URL: https://www.sciencedirect.com/science/article/pii/S1524070314000502.

[LW15] Frederico Limberger and Richard Wilson. "Feature Encoding of Spectral Signatures for 3D Non-Rigid Shape Retrieval". In: Jan. 2015. DOI: 10.5244/C.29.56.

[LZ10] Bruno Lévy and Hao Zhang. "Spectral Mesh Processing". In: *ACM SIGGRAPH 2010 Courses*. 2010, pp. 1–312.

[Mac49] Richard Henri MacNeal. "The Solution of Partial Differential Equations by Means of Electrical Networks." Dissertation (Ph.D.) USA, 1949. DOI: 10.7907/PZ04-5290. URL: https://resolver.caltech.edu/CaltechETD:etd-04282004-143609.

[Mar+08] Sebastian Martin, Peter Kaufmann, Mario Botsch, Martin Wicke, and Markus Gross. "Polyhedral Finite Elements Using Harmonic Basis Functions". In: *Computer Graphics Forum* 27.5 (2008), pp. 1521–1529. DOI: https://doi.org/10.1111/j.1467-8659.2008.01293.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2008.01293.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01293.x.

[MS10] Josiah Manson and Scott Schaefer. "Moving Least Squares Coordinates". In: *Computer Graphics Forum* 29.5 (2010), pp. 1517–1524. DOI: https://doi.org/10.1111/j.1467-8659.2010.01760.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2010.01760.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2010.01760.x.

[MS89] David Mumford and Jayant Shah. "Optimal Approximations by Piecewise Smooth Functions and Associated Variational Problems". In: *Communications on Pure and Applied Mathematics* 42.5 (1989), pp. 577–685. DOI: https://doi.org/10.1002/cpa.3160420503.

[Mul+08] Patrick Mullen, Yiying Tong, Pierre Alliez, and Mathieu Desbrun. "Spectral Conformal Parameterization". In: *Computer Graphics Forum* 27.5 (2008), pp. 1487–1494. DOI: https://doi.org/10.1111/j.1467-8659.2008.01289.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2008.01289.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01289.x.

[Ovs+12] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. "Functional Maps: a Flexible Representation of Maps Between Shapes". In: *ACM Trans. Graph.* 31.4 (July 2012). ISSN: 0730-0301. DOI: 10.1145/2185520.2185526. URL: https://doi.org/10.1145/2185520.2185526.

[Ovs+17] Maks Ovsjanikov, Etienne Corman, Michael Bronstein, Emanuele Rodolà, Mirela Ben-Chen, Leonidas Guibas, Frederic Chazal, and Alex Bronstein. "Computing and Processing Correspondences with Functional Maps". In: *ACM SIGGRAPH 2017 Courses*. SIGGRAPH '17. Los Angeles, California: Association for Computing Machinery, 2017. ISBN: 9781450350143. DOI: 10.1145/3084873.3084877. URL: https://doi.org/10.1145/3084873.3084877.

[Por+17] Roi Poranne, Marco Tarini, Sandro Huber, Daniele Panozzo, and Olga Sorkine-Hornung. "Autocuts: Simultaneous Distortion and Cut Optimization for UV Mapping". In: *ACM Trans. Graph.* 36.6 (Nov. 2017). ISSN: 0730-0301. DOI: 10.1145/3130800.3130845.

[Rab+17] Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. "Scalable Locally Injective Mappings". In: *ACM Trans. Graph.* 36.2 (Apr. 2017). ISSN: 0730-0301. DOI: 10.1145/2983621. URL: https://doi.org/10.1145/2983621.

[Red19] Junuthula Narasimha Reddy. *Introduction to the Finite Element Method*. McGraw-Hill Education, 2019.

[Rus07] Raif M. Rustamov. "Laplace-Beltrami Eigenfunctions for Deformation Invariant Shape Representation ". In: *Geometry Processing*. Ed. by Alexander Belyaev and Michael Garland. The Eurographics Association, 2007. ISBN: 978-3-905673-46-3. DOI: /10.2312/SGP/SGP07/225-233.

[RWP06] Martin Reuter, Franz-Erich Wolter, and Niklas Peinecke. "Laplace–Beltrami spectra as 'Shape-DNA' of surfaces and solids". In: *Computer-Aided Design* 38.4 (2006). Symposium on Solid and Physical Modeling 2005, pp. 342–366. ISSN: 0010-4485. DOI: https://doi.org/10.1016/j.cad.2005.10.011. URL: https://www.sciencedirect.com/science/article/pii/S0010448505001867.

[SA19] Florian Schäfer and Anima Anandkumar. "Competitive Gradient Descent". In: *Advances in Neural Information Processing Systems* 32 (2019).

[SAO20] Florian Schäfer, Anima Anandkumar, and Houman Owhadi. "Competitive Mirror Descent". In: *arXiv preprint arXiv:2006.10179* (2020).

[SC18] Nicholas Sharp and Keenan Crane. "Variational Surface Cutting". In: *ACM Trans. Graph.* 37.4 (2018).

[SC20] Nicholas Sharp and Keenan Crane. "A Laplacian for Nonmanifold Triangle Meshes". In: *Computer Graphics Forum (SGP)* 39.5 (2020).

[SCT03] Olga Sorkine, Daniel Cohen-Or, and Sivan Toledo. "High-Pass Quantization for Mesh Encoding". In: *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. SGP '03. Aachen, Germany: Eurographics Association, 2003, pp. 42–51. ISBN: 1581136870.

[SGK19] Breannan Smith, Fernando Goes, and Theodore Kim. "Analytic Eigensystems for Isotropic Distortion Energies". In: *ACM Transactions on Graphics* 38 (Feb. 2019), pp. 1–15. DOI: 10.1145/3241041.

[SH02] Alla Sheffer and John C. Hart. "Seamster: Inconspicuous Low-Distortion Texture Seam Layout". In: *Proceedings of the Conference on Visualization '02*. VIS '02. Boston, Massachusetts: IEEE Computer Society, 2002, pp. 291–298. ISBN: 0780374983.

[Sha+20] Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. "Diffusion is All You Need for Learning on Surfaces". In: *CoRR* abs/2012.00888 (2020). arXiv: 2012.00888. URL: https://arxiv.org/abs/2012.00888.

[She+19] Hanxiao Shen, Zhongshi Jiang, Denis Zorin, and Daniele Panozzo. "Progressive Embedding". In: *ACM Trans. Graph.* 38.4 (July 2019). ISSN: 0730-0301. DOI: 10.1145/3306346.3323012. URL: https://doi.org/10.1145/3306346.3323012.

[Sht+17] Anna Shtengel, Roi Poranne, Olga Sorkine-Hornung, Shahar Z. Kovalsky, and Yaron Lipman. "Geometric Optimization via Composite Majorization". In: *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073618.

[SOG09] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. "A Concise and Provably Informative Multi-Scale Signature Based on Heat Diffusion". In: *Computer Graphics Forum* 28.5 (2009), pp. 1383–1392. DOI: https://doi.org/10.1111/j.1467-8659.2009.01515.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01515.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01515.x.

[Sor05] Olga Sorkine. "Laplacian Mesh Processing". In: *Eurographics 2005 - State of the Art Reports*. Ed. by Yiorgos Chrysanthou and Marcus Magnor. The Eurographics Association, 2005.

[SPR06] Alla Sheffer, Emil Praun, and Kenneth Rose. "Mesh Parameterization Methods and Their Applications". In: *Foundations and Trends in Computer Graphics and Vision* 2.2 (2006), pp. 105–171.

[SS15] Jason Smith and Scott Schaefer. "Bijective Parameterization with Free Boundaries". In: *ACM Trans. Graph.* 34.4 (July 2015). ISSN: 0730-0301. DOI: 10.1145/2766947.

[SSC19] Nicholas Sharp, Yousuf Soliman, and Keenan Crane. "The Vector Heat Method". In: *ACM Trans. Graph.* 38.3 (2019).

[Ste+20] Oded Stein, Alec Jacobson, Max Wardetzky, and Eitan Grinspun. "A Smoothness Energy without Boundary Distortion for Curved Surfaces". In: *ACM Trans. Graph.* 39.3 (Mar. 2020). ISSN: 0730-0301. DOI: 10.1145/3377406. URL: https://doi.org/10.1145/3377406.

[Su+20] Jian-Ping Su, Chunyang Ye, Ligang Liu, and Xiao-Ming Fu. "Efficient Bijective Parameterizations". In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392435.

[Tau95] Gabriel Taubin. "A Signal Processing Approach to Fair Surface Design". In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: Association for Computing Machinery, 1995, pp. 351–358. ISBN: 0897917014. DOI: 10.1145/218380.218473. URL: https://doi.org/10.1145/218380.218473.

[Tut63] W. T. Tutte. "How to Draw a Graph". In: *Proceedings of the London Mathematical Society* s3-13.1 (1963), pp. 743–767. DOI: 10.1112/plms/s3-13.1.743. eprint: https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/plms/s3-13.1.743. URL: https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/plms/s3-13.1.743.

[Var10] S. Varadhan. “On the Behavior of the Fundamental Solution of the Heat Equation with Variable Coefficients”. In: *Communications on Pure and Applied Mathematics* 20 (May 2010), pp. 431–455. DOI: 10.1002/cpa.3160200210.

[Váš+14] L. Váša, S. Marras, K. Hormann, and G. Brunnett. “Compressing Dynamic Meshes with Geometric Laplacians”. In: *Comput. Graph. Forum* 33.2 (May 2014), pp. 145–154. ISSN: 0167-7055. DOI: 10.1111/cgf.12304. URL: https://doi.org/10.1111/cgf.12304.

[Vei+12] L. Veiga, Franco Brezzi, Andrea Cangiani, G. Manzini, L. Marini, and Alessandro Russo. “Basic Principles of Virtual Element Methods”. In: *Mathematical Models and Methods in Applied Sciences* 23 (Nov. 2012).

[Wac75] E. L. Wachspress. *A Rational Finite Element Basis*. Vol. 114. Mathematics in Science and Engineering. Academic Press, 1975.

[War+07] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. “Discrete Laplace Operators: No Free Lunch”. In: vol. 07. Jan. 2007, pp. 33–37. DOI: 10.1145/1508044.1508063.

[War08] Max Wardetzky. “Convergence of the Cotangent Formula: An Overview”. In: *Discrete Differential Geometry*. Ed. by Alexander I. Bobenko, John M. Sullivan, Peter Schröder, and Günter M. Ziegler. Basel: Birkhäuser Basel, 2008, pp. 275–286. ISBN: 978-3-7643-8621-4. DOI: 10.1007/978-3-7643-8621-4_15. URL: https://doi.org/10.1007/978-3-7643-8621-4_15.

[WCL24] Colin Weill–Duflos, David Coeurjolly, and Jacques-Olivier Lachaud. “Digital Calculus Frameworks and Comparative Evaluation of Their Laplace-Beltrami Operators”. In: *Discrete Geometry and Mathematical Morphology: Third International Joint Conference, DGMM 2024, Florence, Italy, April 15–18, 2024, Proceedings*. Florence, Italy: Springer-Verlag, 2024, pp. 93–106. ISBN: 978-3-031-57792-5. DOI: 10.1007/978-3-031-57793-2_8. URL: https://doi.org/10.1007/978-3-031-57793-2_8.

[Wei+23] Colin Weill–Duflos, David Coeurjolly, Fernando de Goes, and Jacques-Olivier Lachaud. “Joint Optimization of Distortion and Cut Location for Mesh Parameterization using an Ambrosio-Tortorelli Functional”. In: *Computer Aided Geometric Design* 105 (2023), p. 102231. ISSN: 0167-8396. DOI: https://doi.org/10.1016/j.cagd.2023.102231. URL: https://www.sciencedirect.com/science/article/pii/S0167839623000638.

[XLH11] Yunhui Xiong, Guiqing Li, and Guoqiang Han. “Mean Laplace–Beltrami Operator for Quadrilateral Meshes”. In: *T. Edutainment* 5 (Jan. 2011), pp. 189–201. DOI: 10.1007/978-3-642-18452-9_15.

[Zhu+20] Tianyu Zhu, Chunyang Ye, Shuangming Chai, and Xiao-Ming Fu. “Greedy Cut Construction for Parameterizations”. In: *Computer Graphics Forum* 39.2 (2020), pp. 191–202. DOI: https://doi.org/10.1111/cgf.13923. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13923. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13923.

[ZRH11] Valentin Zobel, Jan Reininghaus, and Ingrid Hotz. “Generalized Heat Kernel Signatures”. In: (2011).